Fireworks: Evolutionary art project based on EvoSpace-Interactive

Leonardo Trujillo
and Mario García-Valdez
División de Estudios de Posgrado
Instituto Tecnológico de Tijuana, Mexico
Email: leonardo.trujillo@tectijuana.edu.mx
mario@tectijuana.edu.mx

Francisco Fernández-de-Vega
Grupo de Evolución Artificial
Universidad de Extremadura, Spain
Email: fcofdez@unex.es

Juan-J. Merelo
Departamento de Arquitectura y
Tecnología de Computadores
University of Granada, Spain
Email: jmerelo@geneura.ugr.es

***Abstract*—This paper presents a collaborative-interactive evolutionary algorithm (C-IEA) that evolves artistic animations and is executed on the web. The application is called *Fireworks,* since the animations that are produced are similar to an elaborate fireworks display. The system is built using the EvoSpace platform for distributed and asynchronous evolutionary algorithms. EvoSpace provides a central repository for the evolving population and remote clients, called EvoWorkers, that interact with the system to perform individual evaluation using and interactive approach. The artistic animations are coded using the Processing programming language that facilitates rapid development of computer graphics applications for artists and graphic designers. The system promotes user collaboration and interaction by allowing many users to participate in population evaluation and because the system incorporates social networking. Initial results show that the proposed C-IEA can allow users to produce interesting artistic artifacts that incorporate preferences from several users, evolving dynamic animations that are unique within evolutionary art.**

***Keywords*—Distributed algorithms, cloud computing, interactive evolutionary algorithm, linear genetic programming.**

I. Introduction

Artificial evolution has proven to be a powerful computational paradigm used to solve optimization, search and learning problems in diverse domains. Moreover, evolutionary algorithms (EAs) have found a wider application area than just traditional engineering or scientific domains, with promising contributions to art, music and design being developed at a steady pace over recent years [1], [2], [3], [4], [5]. The nature of EAs, which combine a balanced mixture of exploitative optimization with explorative search, has allowed EAs to become robust and versatile algorithms that solve difficult non-linear and discontinuos problems, even in cases where an analytic or automatic objective function cannot be derived [6].

In particular, when dealing with creative domains, fitness usually needs to incorporate some sort of human knowledge and preferences, either off-line (before the search) or on-line (during the search). The former case are referred to as supervised learning algorithms, while the latter case refers to interactive EAs (IEA). Human interaction is required in these domains because creativity and aesthetic principles are not yet fully understood, which limits the possibility of measuring them directly and objectively. That is why researchers have turned towards humans to provide an indirect and subjective evaluation of evolved artistic artifacts. While the IEA approach solves one problem, it also creates others, such as the following. First, a human can easily get tired of what can be a monotonous task of evaluating many individuals over many generations, most of which will only be marginally interesting. Secondly, a user can get bored, and start to provide feedback that deviates from his original goal. Finally, given the subjective nature of evaluating artistic design, a single user might not provide the required feedback; i.e., the fitness landscape might not present the necessary structure for the search to proceed in a non-random way. Nonetheless, IEAs have been used to generate impressive artistic designs, incorporating insights from computer science and artificial intelligence researchers, as well as artists from various domains.

Only until recently, however, has an EA tool been proposed that is specifically aimed at IEA for artistic design. EvoSpace-Interactive was developed with this goal in mind, providing a simplified development interface for IEAs [7], [8]. Moreover, EvoSpace-Interactive was designed to incorporate and combine the subjective opinions from different users of the system. EvoSpace-Interactive employs a cloud-based population manager that allows for asynchronous and distributed interaction from users, and employs social network integration to promote user collaboration and interaction. Therefore, we shall refer to EvoSpace-Interactive as a Collaborative-IEA (C-IEA).

In [8] the EvoSpace tool is presented and in [7] a proof-of-concept implementation was presented for a C-IEA for artistic design. In the present paper, on the other hand, the first advanced evolution of artistic artifacts using EvoSpace. Moreover, the application exploits the Processing programming language, that facilitates artistic development for non-computer scientists. Moreover, the system incorporates and promotes user collaboration by integrating with a social networking service. The proposed system is called *Fireworks,* that employs a linear genetic programming (LGP) approach [9] to evolve artistic animations of particle swarms that visually resemble a fireworks display.

The remainder of this paper proceeds as follows. First, Section II presents a discussion of related works. Then, Section III presents the EvoSpace platform for C-IEAs. The *Fireworks* system is described in Section IV, detailing the LGP algorithm and Processing based representation of individuals. Experimental results are summarized and discussed in Section V.

Finally, Section VI outlines the main conclusions of the work.

II. Related Work

The goal of IEAs is to use human preferences as the main (or only) source of selective pressure that guides an evolutionary search [10], [11]. IEAs pose an open-ended search where the objective function used by traditional EAs is replaced by the subjective preferences and interaction of human users of the system. Indeed, some of the earliest EAs were open-ended systems, such as the well-known Biomorphs algorithm developed by Richard Dawkins[12]. Therefore, IEAs should encourage high diversity and exploration given the dynamic and non-deterministic nature of the evaluation process [13].

IEAs have been used in various domains and applications; for instance, [14] uses an IEA to evolve medical implants to improve hearing. However, the present work focuses on the evolution of artistic animations using a collaborative and distributed approach. IEAs that incorporate user collaboration and interaction are here referred to as Collaborative-IEAs or C-IEAs; some noteworthy works in this area are reviewed next.

An early example of a web based interactive system is the work by Langdon [15], where fractal representations of virtual creatures were evolved. He proposed a distributed EA using a global population that resides on a central website and uses Javascript to distribute portions of the population to remote clients. Users evaluate individuals locally and those a user prefers are returned to the server and can be distributed again to others, thus promoting a collaborative process during evolution. Similarly, Secretan et al. [3] and Clune and Lipson [4] use web-based IEAs to evolve artistic artifacts using compositional pattern producing networks as a generative encoding. In [3] images are evolved, while 3-D sculptures are evolved in [4]. In both cases, users (connected clients) are encouraged to collaborate. In both works a central websites was developed where users can select or create random individuals and evolve lineages of artifacts based on their preferences. In these systems a user can take a previously evolved artifact and continue the search process himself, building upon earlier design in a sequential manner. Therefore, the evolved artifacts can be the product of a collaborative evolutionary search. Another feature is that evolved artifacts can be rated by users, and given that users can create individual accounts, the ratings provide a way to rank users, or to select previously evolved artifacts based on the particular style of each user. Furthermore, the collaborative process is captured by the system, since it is possible to visualize how, and when, different users influenced a genetic lineage. Kowaliw et al. [5] present recent example based on evolving ecosystemic models, a generative encoding based on multi-agent systems that generate high quality artistic drawings. Again, users visit a website and interact with a Java applet, after which they can choose to add evolved images to a central collection such that other users can see them or them as seeds for their own evolutionary design.

The present work builds on previous proposals and extends the C-IEA approach. First, it promotes collaborative evaluations of artistic artifacts in a dynamic and parallel manner, instead of the sequential approach followed in [3], [4]. Second, it incorporates explicit user interactions by encouraging the use of social networking. Third, by exploiting the Processing programming language for graphics programming, it facilitates rapid development for artists with a limited computer science background. Fourth, it focuses on the evolution of artistic animations, not static pictures or paintings; another feature facilitated by the use of Processing. Fifth, it facilitates the ability to save and share promising artifacts. Finally, it emphasizes the use of a distributed model, to exploit current trends in software and hardware technology.

III. EvoSpace

EvoSpace is a population store for the development of evolutionary algorithms that are intended to run on a cloud computing model [8]. It is designed to be versatile, since the population is decoupled from any particular evolutionary algorithm. Client processes, called EvoWorkers, dynamically and asynchronously interact with the EvoSpace store and perform the basic routines of an evolutionary search. EvoWorkers can reside on remote clients or on the platform server itself. The EvoSpace platform is presented in [8], here we only provide a general outline of how the system functions.

EvoSpace consists of two main components. The first one is the EvoSpace container, used to store the EA population. The second component consists of the remote clients or EvoWorkers, which execute the actual evolutionary process. Figure 1 presents a schematic diagram of the EvoSpace system and data flow.

A. *The EvoSpace container.*

EvoSpace is based on the tuplespace model, a distributed shared memory (DSM) abstraction organized as a *bag* of tuples. A tuple t is the basic tuplespace element, composed by one or more fields and corresponding values. In this model the basic operations that a process can perform is to insert or withdraw tuples from the tuplespace. EvoSpace is composed by a set of objects ES and a set of interface methods provided by a central server. The current EvoSpace implementation offers the following interface methods ti withdraw and replace objects from ES.

Read(n): This method returns a random set A of objects from ES, with $|A| = n$ and $A \subset ES$, if $n < |ES|$, the method returns ES otherwise.

Take(n): Returns a random set A, following the similar logic used for $Read()$. However, in this case the sequence of $Take()$ operations provide a temporal dimension to the dynamics of set ES. We can define ES_i as the set at the moment of the $i-th$ $Take()$ operation and A_i as the output. The contents of EvoSapce are then given by $ES_{i+1} = ES_i \setminus A_i$; i.e., the objets taken are effectively removed from ES. The objects taken are also copied to a new set S_i of *sampled objects* and stored within a temporary collection $\mathcal{S}$ on the server, implemented as a priority queue. Sets $S_i \in \mathcal{S}$ can then be reinserted to ES if necessary.

ReInsert(i): This method is used to reinsert the subset of elements removed by the $i-th$ $Take()$ operation, such that the contents of EvoSpace are now $ES \cup S_i$ if $S_i \in \mathcal{S}$ and ES is left unchanged otherwise.

Insert(A): This method represents the union operation $ES \cup A$.

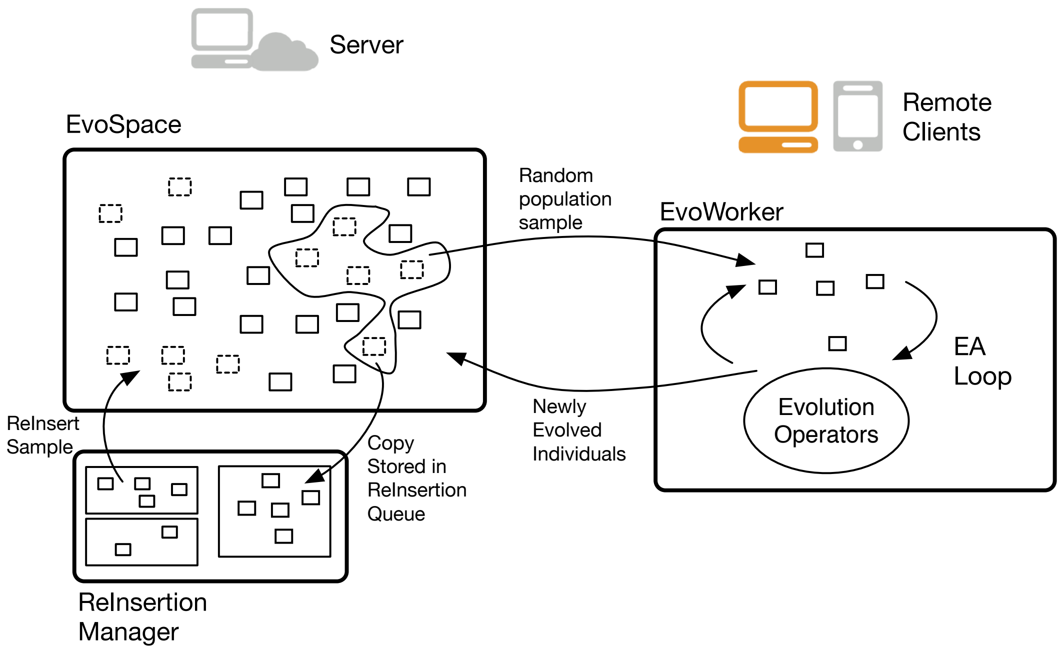


Fig. 1. Main components and dataflow within the EvoSpace platform. The figure illustrates the two main components, the EvoSpace container on the server and the remote EvoWorker clients. Notice how the population is samples and distributed to the clients where evaluation and other evolutionary operators can be applied.

Replace(A,i): Similar to $Add()$, however set A should be understood as a replacement for some $S_i \in \mathcal{S}$, hence $|A| = |S_i|$, but the objects in A can be different (evolved) objects from those in S_i. Moreover, if S_i exists it is removed from $\mathcal{S}$. However, if S_i does not exist this means that a $ReInsert(i)$ operation preceded it, this increases the size of ES.

Remove(A): This method removes all of the objects in A that are also in ES, in such a way that the contents of EvoSpace are now set to $ES \cup (A \cap ES)$.

The sub-components of EvoSpace are summarized below.

Individuals. The objects contained within ES represent the evolving individuals of an EA. Individuals in ES are stored as *dictionaries*, a collection of unique keys and values with a one to one association.

The EvoSpace Server. On the server side the `EvoSpaceServer` process creates and activates a new EvoSpace container object and waits for requests to execute interface methods. Three other server processes are also executed: `InitPopulation`, `ReInsertionMgr` and `EvolutionMgr`. `InitPopulation` initializes the population with a total of $popsize$ random individuals. `ReInsertionMgr` periodically checks (every wt seconds) if the size of the population in ES falls below a certain threshold min or if the time after the last reinsetion is greater than $next_r$. If any of these conditions are satisfied, then rn subsets $S_i \in \mathcal{S}$ are reinserted into ES using the $ReInsertOld()$ method. Finally, `EvolutionMgr` periodically checks if a termination condition is satisfied according to the needs of the evolutionary search. For an open-ended C-IEA a termination criteria is usually not implemented, and is left up to the researcher to halt evolution when he consideres it necessary.

EvoWorkers. The `EvoWorker` process requests a set of ew_{take} individuals from the ES container using the $Take()$ interface method provided by EvoSpace. In the case of an IEA, the individuals are presented to a user of the system which is prompted to evaluate them.

Evolve Process. Finally, in the case of an C-IEA, after a pre-specified number of $Take()$ operations $evolve_{threshold}$ the $Evolve()$ function is invoked on the server side, which proceeds to request a set of ev_{take} individuals using the $Take()$ method, on which the specified set of genetic operators (selection, mutation, crossover, etc.) are applied to generate offspring which are then returned and reinserted into the population.

Implementation of the population store. Individuals are stored in-memory, using the Redis key-value database. Redis was chosen over a SQL-based management system, or other non-SQL alternatives, because it provides a hash based implementation of sets and queues which are natural data structures for the EvoSpace model. For example, selecting a random key from a set has a complexity of O(1). The logic of EvoSpace is implemented as a Python module exposed as a Web Service using cherrypy and Django HTTP frameworks. The EvoSpace web service can interact with any language supporting json-rpc or ajax requests. The EvoSpace modules and workers in JavaScript, JQuery and python are available with a Simplified BSD License from http://evospace.org.

B. *EvoSpace-Interactive.*

This paper presents a C-IEA system based on an extension of the EvoSpace platform called EvoSpace-Interactive [7], depicted in the block diagram of Figure 2. With EvoSpace-Interactive developers can exploit a platform for distributed user collaboration and interactive evolution that offers the following: a central repository for the population implemented as an EvoSpace service; a Web Application script implemented using Django; and a mature full stack Web Framework with a BSD license developed in Python. EvoSpace-Interactive is responsible for user authentication and session handling through the popular Facebook social network using the OAuth 2.0 protocol. Also, the storage of collections, where users can store individuals they want to share with friends, is persisted using the PostgreSQL DBMS. Most of the interactive functionality is programmed in the client side using Javascript libraries. The communication between components is implemented using json-rpc, a lightweight remote procedure call protocol and common ajax and http transactions. Overall functionality is decomposed in specialized services, adding flexibility to the framework since services can be interchanged. The framework is built using only open source components from libraries to servers. Users interact with the system through a GUI implemented on a Responsive Web Design (RWD) front end framework, an approach to web design in which a graphical user interface is crafted to provide a satisfactory viewing experience in a range of mobile devices. This approach enables designers to tailor the look and feel of the application with minimum intrusion, only changing CSS definitions.

Three components must be specified by the application developer, marked with double lines in Figure 2; these are: an *individual* representation; a *processing script* that renders each individual; and a *worker* script that encodes the evolutionary operators depending upon the proposed representation. Each of these points are discussed below.

C. User Interface

The users interact with the web interface composed of the five elements proposed in [7], see Figure 3. First, at the top left corner user login, where users can link to their Facebook account or decide to participate as an anonymous user. Second, to encourage user interaction when a user chooses to login to his Facebook account a list of friends that have also linked to the C-IEA application is presented on the left.

The third element is a central *Wall* area, where ew_{take} randomly sampled individuals by a $Take()$ operation are shown to the user. Here, the user can interact with the system by clicking on the individuals he prefers, which counts as a ”like" for the individual, or the user can add an image to a *collection* of individuals, a special directory to store individuals a user prefers and wishes to save.

After the user finishes interacting with the current crop of individuals he can choose to retrieve a new sample from EvoSpace. This is done with the fourth element of the interface, located at the top of the screen, the *Get More* button. The button returns the current group of individuals to EvoSpace, and brings back a new one.

The *collections* section is the fifth element of the interface, shown at the bottom left corner. The user can create as many collections as he wants to organized his preferred individuals. Moreover, a user can browse the content of each collection and from there share images through the social network. When a user browses over an individual a detail pane shows the list of users that have liked the individual.

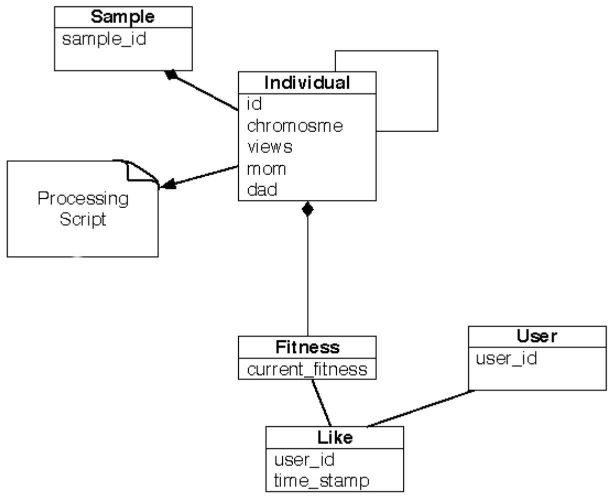


Fig. 4. Individual internal representation.

D. Individual Representation

As stated above, individuals are represented internally as a dictionary. The properties stored by EvoSapce-Interactive applications are: a unique `id`; a user defined `chromosome`; the number of times the individual has been selected in a sample and returned to the population, stored in property called `views`; the genetic operators that generated the individual; ids of the parents; `current Fitness` that stores the most resent fitness value; and a `fitness` dictionary where each key is a concatenation of a `userid`, a `timestamp` and a numerical value that represents the rating given by the corresponding individual. A UML representation of an individual is presented in Figure 4.

E. Processing Language and HTML Canvas Element

Processing is a programming language and development environment initially created to serve as a software sketchbook and as a tool to teach fundamentals of computer programming within a visual context. Currently is used by artists, designers, architects, and researchers for visualization applications, games and interactive animations projects [16]. Processing is a subset of Java directed to novice programmers and generative artists [17], which are the intended users of the EvoSpace-Interactive framework. As a complement there is a javascript library *processingjs* that allows Processing scripts to be run by any HTML5 compatible browser. Processing scripts are render individuals on the web interface, which can involve static images, animations, sound or even interactive artifacts. Before calling the $draw()$ method of the processing script initial place holding or fallback parameters are replaced with an individual’s chromosome. Each individual’s script has its own Canvas entity, defined by the HTML5 standard as an element that provides scripts with a resolution-dependent bitmap canvas which can be used for rendering graphics on the fly. Although the combination of an HTML5 Canvas element and a Processing script is supported by default, other combinations could be used. For instance, images, embedded audio, or other libraries capable of drawing in the Canvas. Also, a fallback implementation must be considered for applications intended for non-HTML5 capable browsers.

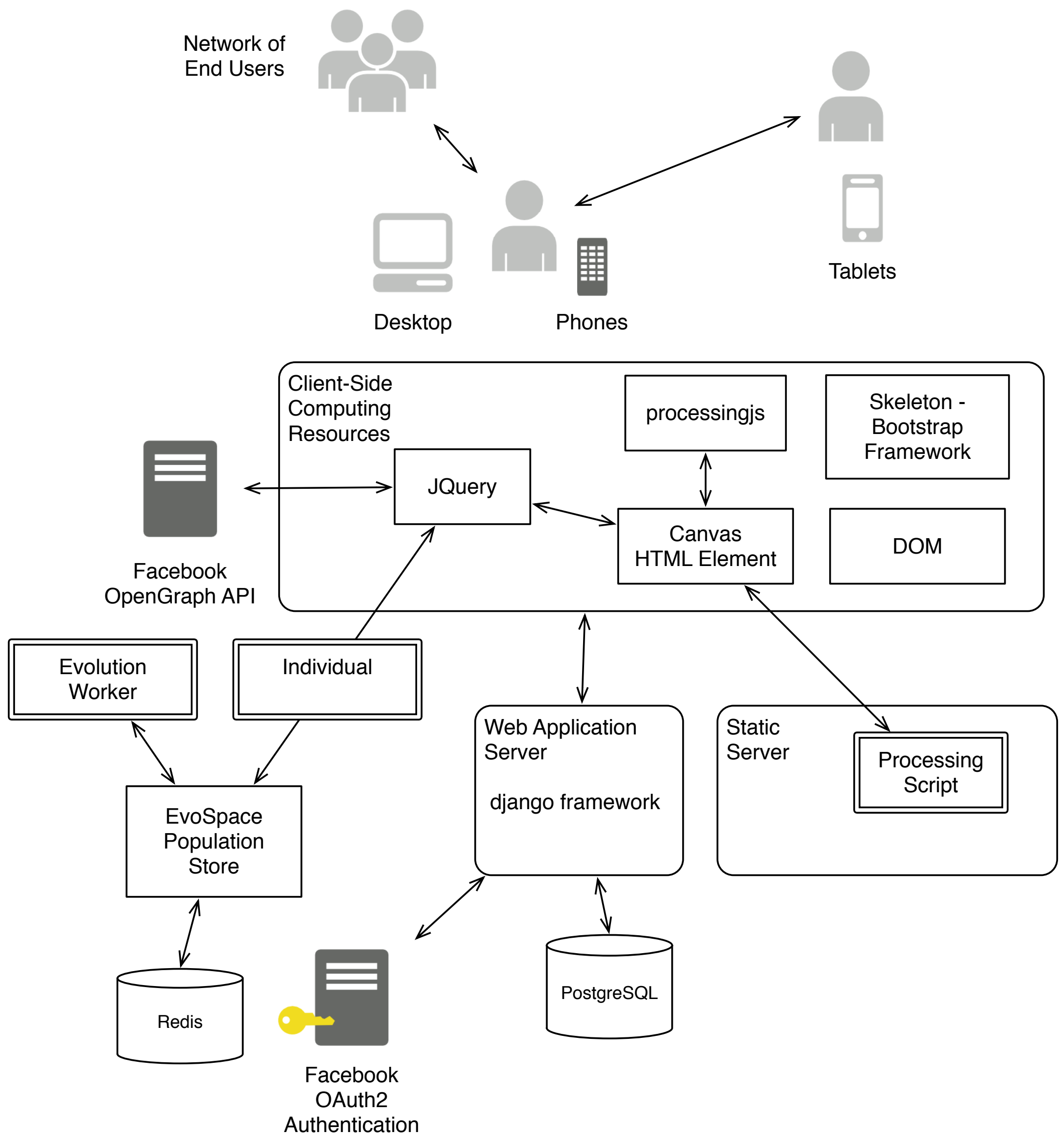


Fig. 2. Conceptual design of the proposed C-IEA. The figure presents a block diagram of the main software components of the system and how they interact with each other on the cloud, to provide a collaborative interactive application for remote client users.

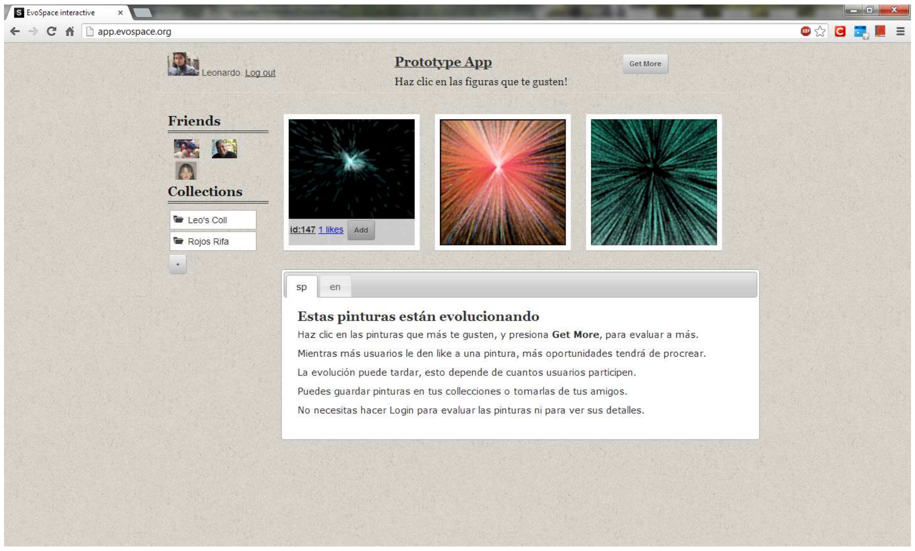
IV. Fireworks Application

This section presents the evolutionary art application proposed in this work called Fireworks. First, the artistic artifacts are described, developed using Processing scripts that can produce complex graphics in a simple development language. Afterwards, the linear GP encoding of individuals is given. Finally, the genetic operators for selection, crossover and mutation are described, as well as the fitness assignment process.

A. Fireworks Swarm Animations

The goal of the Fireworks C-IEA is to evolve artistic animations of particle swarms. It is noteworthy to point out that evolving animations is a unique application within current evolutionary art literature that mostly focuses on static images or sculptures. The animations are based on the open-source Processing script developed by Claudio Gonzales called Galactic Dust available at http://www.openprocessing.org/sketch/8062, licensed under Creative Commons Attribution-Share Alike 3.0 [1] and GNU GPL license[2]. The Galactic Dust script presents a virtual 3D canvas where a set of N particles can move about. In the original version of the script the particles are randomly positioned within the 3D canvas (or placed in an a priori pattern) and remain static. Then, when the user left-clicks on a point within the canvas this specifies the position of a gravitational point P on the canvas and produces an attractive force on all of the particles towards P, proportional to their distance to P and mass which is also specified initially. Similarly, when a user right-clicks on the canvas a repulsive force is produced. Moreover, the particles continuously change color using small random steps and the user can toggle a tracing effect on or off, in which each particle leaves a visual

[1]ttp://creativecommons.org/licenses/by-sa/3.0/
[2]http://creativecommons.org/licenses/GPL/2.0/



Prototype App

Haz clic en las figuras que te gusten!

Estas pinturas están evolucionando

Haz clic en las pinturas que más te gusten, y presiona **Get More**, para evaluar a más.

Mientras más usuarios le den like a una pintura, más oportunidades tendrá de procrear.

La evolución puede tardar, esto depende de cuantos usuarios participen.

Puedes guardar pinturas en tus colleciones o tomarlas de tus amigos.

No necesitas hacer Login para evaluar las pinturas ni para ver sus detalles.

Fig. 3. User interface of the EvoApp C-IEA.

trail of its movements that gradually disappears with time.

The Galactic Dust script is the inspiration and basis for the artistic animations evolved by the Fireworks application. In particular, the goal of Fireworks is to begin with randomly placed particles and to evolve a pattern of movements and behaviors, searching for interesting visual animations. These animations are similar to popular screensavers or background visualizations for music players, however the authors feel that an illustrative description is to say that they resemble elaborate fireworks displays, thus the name. In the following subsection, the search space representation is described.

B. Linear Genetic Programming Representation

The C-IEA uses a LGP representation [9] for the evolution of the Fireworks' swarm animations. Initially, all individuals begin with the central pixel of the 3D canvas as the gravitational point P. The chromosome of each individual is encoded as a variable-length list. Each element, or gene, of the chromosome represents one of five basic operations, these are: $Attract_{flag}$, $Move_{flag}$, $Trace_{flag}$, $ForwardX_{flag}$, $ForwardY_{flag}$ and $Step_m$. The first four instructions are toggle operations for different script parameters that determine the characteristics of particle movements. $Attract_{flag}$ instructs the script to toggle between attraction or repulsion towards P. $Move_{flag}$ toggles between particle movement or particle deceleration at each frame of the animation. $Trace_{flag}$ determines if particle movement will produce tracing or if it will not. $ForwardX_{flag}$ and $ForwardY_{flag}$ flags determines the orientation of particle movements in the horizontal and vertical axis respectively. Finally, the $Step_m$ function determines the magnitude of step movements in each direction given by m pixels, with a random decision uniformly chooses from $u \in [1, 13]$.

C. Genetic Operators

The LGP search employs several types of genetic operators. Firstly, two crossovers are implemented: $2pCross$ and $appendCross$. $2pCross$ chooses two random crossover points on each parent separately and swaps the corresponding sublists between the parents, generating two offspring. If the sublists are of different size then one of the offspring will increase in size while the size of the other will decrease. $appendCross$ produces two offspring that are the concatenation of both parent chromosomes, each in different order. Secondly, five mutation operators are implemented: $Shuffle$, $Move$, $Delete$, $Replace$ and $Insert$. The $Shuffle$ mutation randomly permutes the genes in a parent chromosome. The $Move$ mutation randomly selects a chromosome sublist and then moves it to a new random position within the parent chromosome. The $Delete$ mutation randomly selects and deletes a sublist from the parent chromosome. The $Replace$ mutation randomly selects and deletes a sublist from the parent chromosome and replaces it by a new random list. Finally, $Insert$ mutation generates a new list and splices it into a randomly chosen position within the parent chromosome.

The C-IEA uses tournaments for parent selection, and a crossover and mutation rate determines whether or not the genetic operators are applied. Crossover is applied first and then the resulting offspring are mutated. The crossover and mutation operator that is actually applied is chosen uniformly at random for each pair of parents. The survival strategy is a steady-state implementation, where the offspring generated by the genetic operators replace the two worst individuals from the selection tournament.

D. Fitness Evaluation

The fitness of each individual is computed based on the number of users who liked it relative to the number of times if has been seen. In the current implementation, users can only give positive evaluations explicitly when they select an individual, a *like*. However, the system also keeps track of the number of times an individual has been sent to an EvoWorker client for evaluation, the *views* property. When a user evaluates a sample of individuals, some (or all) of them will not receive a vote, but in all cases the *views* property will be incremented by 1. If an individual has a high number of *views* with a relatively

TABLE I. PARAMETERS OF THE C-IEA USED BY THE *Fireworks* APPLICATION.

Parameter	Value
Population size	100 individuals
Sample size taken by EvoWorkers ew_{take}	3 individuals
Number of samples before $Evolve()$ process ev_{take}	10 samples
Tournament size of the $Evolve()$ process	6
Number of offspring generated by the $Evolve()$ process	2
Survival	Elitist steady-state survival
Operator probabilities (crossover and mutation)	$p_c = 1$ and $p_m = 0.3$

small number of *likes*, then the individual is considered to be less fit then an individual with basically the same number of *views* and *likes* if if the values are low. In other words, the ratio $\frac{likes}{views}$ is more informative, but it does not distinguish between an individual with many *views* and another with only one *view* if they both have zero *likes*. Additionally, *views* must be >=1 to avoid a division by zero. Therefore, fitness is assigned based on $\frac{likes+1}{views+1}$.

V. EXPERIMENTS

This section provides details regarding the experimental setup and the results obtained until now. First, Table I summarizes the parameters of the C-IEA used by the *Fireworks* application. Currently, the *Fireworks* applications has been evolving individuals for over one week, progressively generating novel animations. In general, it is clear that the genetic representation and evaluation criteria has allowed the algorithm to progressively incorporate the preferences of several users. There is no time-table to halt the open-ended evolutionary process, and the system is expected to run for at least one more month. It is now possible to access the application and interact with it at http://app.evospace.org/. It is important to point out that the Processing scripts evolved in this work are computationally costly, and push browser resources to the limit. Only the Chrome web browser offers an acceptable frame rate, using 200 swarm particles within each animation with canvas elements of 200 by 200 pixels in size. Figure 5 presents snapshots of individuals that have been added to a personal collection of one of the authors. Finally, Figure 6 shows a series of frames from a singled evolved animation.

VI. CONCLUDING REMARKS

This paper presents a collaborative interactive evolutionary algorithm (C-IEA) developed on top of the EvoSpace platform. The algorithm provides a store that hosts the evolving population, which can be accessed by remote clients over a web browser. Connected clientes, called EvoWorkers, provide an interactive application to end users that evaluate the individuals produced by the algorithm. Thus, fitness evaluation and genetic variations are carried out asynchronously over a distributed evolutionary process. The algorithm is able to incorporate user collaboration by considering fitness evaluations provided by all connected users of the system.

The C-IEA application is called *Fireworks*, that is designed to produce artistic animations of 3D particle swarms. The animations are coded using the Processing programming language, that eases development for artists and other non-computer scientists, that are more interested in the creative and artistic process. Individuals are coded using a linear GP approach, that facilitates the representation of diverse particle behaviors, that give rise to interesting patterns and displays.

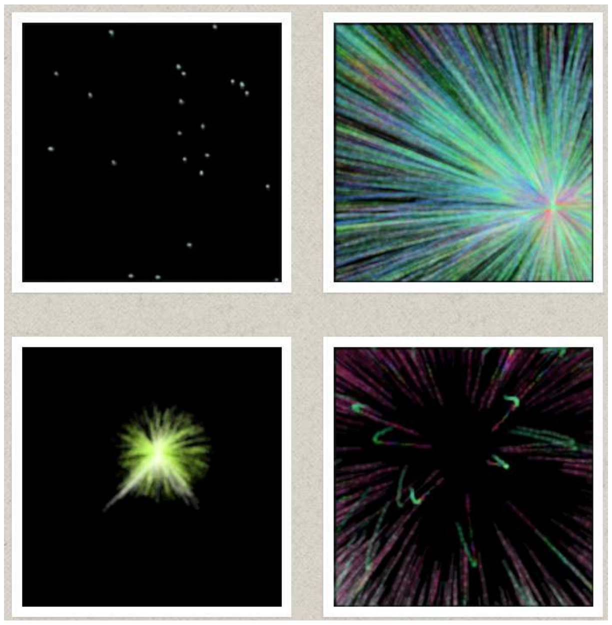


Fig. 5. Snapshots of evolved individuals with the Fireworks application stored in the collection of a system user.

The system continues to run and evolve in an open ended manner, continuously searching and evolving towards solutions that can better integrate diverse user preferences.

REFERENCES

[1] S. Todd and W. Latham, *Evolutionary art and computers*. Academic Press, 1992.

[2] J. McCormack, “Open problems in evolutionary music and art,” in *Proceedings of the 3rd European conference on Applications of Evolutionary Computing*, ser. EC’05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 428–436.

[3] J. Secretan, N. Beato, D. B. D’Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley, “Picbreeder: A case study in collaborative evolutionary exploration of design space,” *Evol. Comput.*, vol. 19, no. 3, pp. 373–403, Sep. 2011.

[4] J. Clune and H. Lipson, “Evolving three-dimensional objects with a generative encoding inspired by developmental biology,” in *Proceedings of the European Conference on Artificial Life*, 2011, pp. 144–148.

[5] T. Kowaliw, A. Dorin, and J. McCormack, “Promoting creative design in interactive evolutionary computation,” *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 4, pp. 523 –536, 2012.

[6] K. A. D. Jong, *Evolutionary computation - a unified approach*. MIT Press, 2006.

[7] M. Garcia, L. Trujillo, F. Fernández-de Vega, J. Merelo, and G. Olague, “Evospace-interactive: A framework to develop distributed collaborative-interactive evolutionary algorithms for artistic design,” in *Proceedings from the 2nd International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design, EvoMUSART ’12*, ser. LNCS, P. Machado and J. McDermott, Eds. Springer-Verlag, 2013, pp. 121–132.

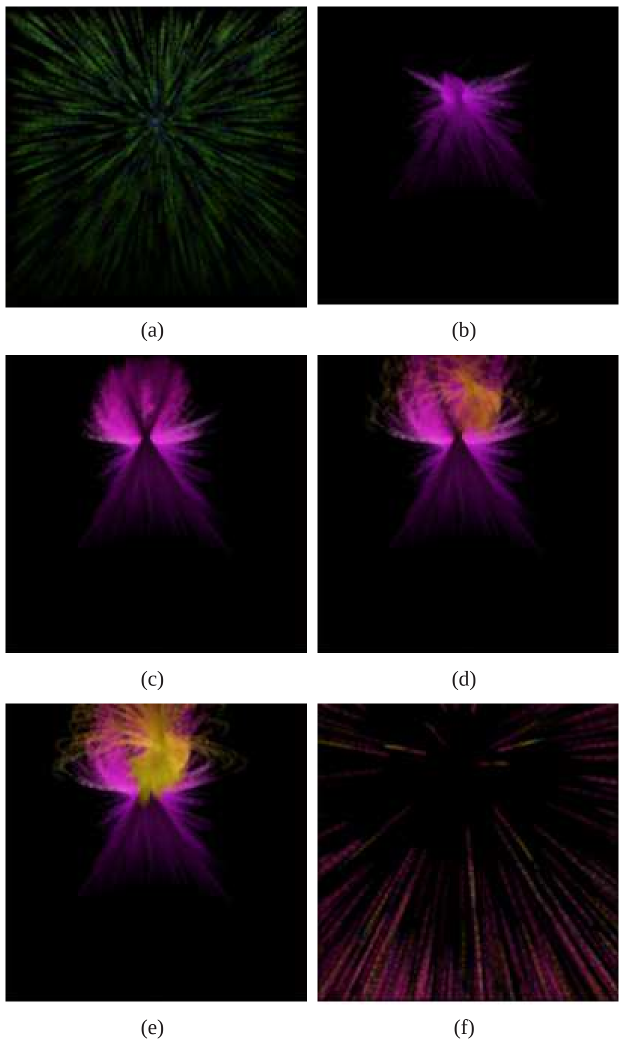


Fig. 6. Sample frames of of an evolved animation.

[8] ——, “Evospace: A distributed evolutionary platform based on the tuple spacemodel,” in *Proceedings from EvoApplications ’12*, ser. LNCS, A. Esparcia-Alcázar, Ed. Springer-Verlag, 2013, pp. 499–508.

[9] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[10] H. Takagi, “Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation,” *Proceedings of IEEE*, vol. 89, no. 9, pp. 1275–1296, 2001.

[11] Y. Semet, “Evolutionary computation: a survey of existing theory,” University of Illinois, Tech. Rep., 2002.

[12] R. Dawkins, *Climbing Mount Improbable*. W.W. Norton & Company, 1996.

[13] B. G. Woolley and K. O. Stanley, “Exploring promising stepping stones by combining novelty search with interactive evolution,” *CoRR*, vol. abs/1207.6682, 2012.

[14] P. Legrand, C. Bourgeois-République, V. Péan, E. Harboun-Cohen, J. Lévy-Vẽhel, B. Frachet, E. Lutton, and P. Collet, “Interactive evolution for cochlear implants fitting,” *Genetic Programming and Evolvable Machines*, vol. 8, no. 4, pp. 319–354, 2007.

[15] W. B. Langdon, “Global distributed evolution of l-systems fractals,” in *Genetic Programming, Proceedings of EuroGP’2004*, ser. LNCS, M. Keijzer, U.-M. O’Reilly, S. M. Lucas, E. Costa, and T. Soule, Eds., vol. 3003. Springer-Verlag, 5-7 April 2004, pp. 349–358.

[16] C. Reas and B. Fry, *A programming handbook for visual designers and artists*. The MIT Press, 2007.

[17] M. Pearson, *Generative Art*, pap/psc ed. Manning Publications, Jul. 2011.