EvoShelf: A System for Managing and Exploring Evolutionary Data

Timothy Davison[1], Sebastian von Mammen[1], and Christian Jacob[1,2]

[1]Dept. of Computer Science, Faculty of Science
[2]Dept. of Biochemistry & Molecular Biology, Faculty of Medicine
University of Calgary, Canada
{tbdaviso,s.vonmammen,cjacob}@ucalgary.ca

Abstract. Systems that utilize evolutionary computation produce large amounts of data. Quite often, this data has a convenient visual representation. However, managing and visualizing evolutionary data can be a difficult and onerous task. By employing techniques used in photo management software, we have produced a system that helps to visualize and organize evolutionary data while retaining a complete record of a simulation. By means of a simple plugin architecture this system can be extended to import data produced by arbitrary evolutionary systems. We present the system's architecture, its features, and we provide a comprehensive example, highlighting its advantages in applied research.

1 Introduction

Evolutionary systems produce large amounts of data. Beyond the obvious data (such as the genotype and phenotype of an individual), there is a considerable amount of meta-data produced as well. Such data includes the hereditary data, fitness values, and other attributes of the evolutionary computation approach being employed.

It is common to manage experimental data by means of a file-system browser, such as the Finder in Mac OS X, and Windows Explorer in Microsoft Windows. Searching or organizing individuals according to various criteria is a laborious task in such systems. Consider a system that organizes the individuals produced by an experiment into sub-directories by generation, giving each individual its own file containing its genotype, and phenotype, along with meta-data such as fitness, or genealogy. Filtering these individuals by fitness value would be a difficult task with either file-system browser.

An evolutionary system may employ an interface of its own for browsing the data that it produces. In this case, the visualization procedures and the management of the genotype/phenotype data are typically implemented specifically for the one evolutionary system. However, the universality of evolutionary algorithmic approaches renders generic visualization and data management techniques valuable across various application domains.

In a way, the situation is very similar to managing individual (digitized) image and music collections. Such libraries can easily consist of thousands of

items. A number of applications have made the organization and management of such data much easier for the end user [1, 2, 5, 10]. We propose a system that can deal with evolutionary data with the same ease of use and flexibility as provided by these mainstream media management applications.

EvoShelf is an extensible system that allows for the importing and exploration of evolutionary data from evolutionary systems. It solves the challenge of organizing imported metadata by providing a navigable, and searchable image-based browser that uses interface design elements from Apple's photo and music management software iPhoto [1], and iTunes [2]. Furthermore, it provides a plugin framework for building additional import modules and visualizations.

In Section 2 we explore the topic of visualizing data in evolutionary systems. Section 3 presents the design of the *EvoShelf* system and its graphical user interface. It also touches upon some of the visualization techniques included in *EvoShelf*, as well as details about its plugin architecture. In Section 4 we use *EvoShelf* in coordination with an existing evolutionary system for semi-interactive evolutionary computing, and for analyzing the results produced by that system. We will conclude in Section 5 with a summary of our work, along with possible directions in which to take it in the future.

2 Related Work

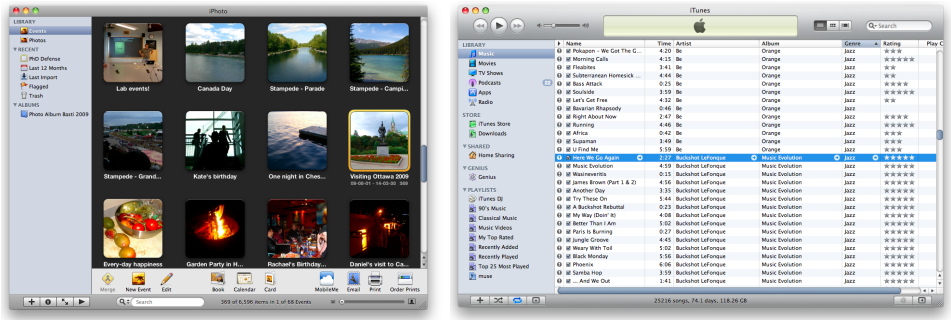
We briefly outline the data management and user-interface approach of various software that inspired the *EvoShelf* visualization and management system. Secondly, we outline various techniques that have been developed for visually supporting computational evolutionary experiments.

2.1 Digital Media Libraries

The framework presented in this article was mainly inspired by iPhoto, Apple's mainstream photo management application [1]. It is capable of organizing and browsing thousands of photos. Despite the large amounts of information that it is capable of presenting to the user, it maintains a very simple and intuitive interface. It consists of two primary views, an organizer view, and an image browsing view (Figure 1(a)). Multiple images, up to and including an entire library of photos, are displayed in the browsing view. The organizer view is used to filter this view into subsets of photos, such as those represented by a photo album containing the user's favorite photos. As photos are imported into the system they are grouped into events. Pictures taken during a certain period of time might have all been taken during a vacation and the respective group of photos could be labeled after the location of the recreational stay.

iTunes is another application from Apple Inc. that manages a large amount of data in a similar fashion to iPhoto. Unlike iPhoto, whose interface is focused on visualizing and managing photos, iTunes is targeted towards playing music and organizing large digital music collections. Visual cover art often decorates individual music files, but the iTunes library is mainly organized by sorting and

searching through textual meta-data such as artist name, music category, or album name (Figure 1(b)). Together, iPhoto and iTunes suggest an interface that combines visualization and meta-data management techniques that could be very powerful for organizing evolutionary data.



(a) (b)

Fig. 1. The user interfaces of the media management applications (a) iPhoto and (b) iTunes.

2.2 Evolutionary Visualization Techniques

Various data visualization techniques have been presented in the context of evolutionary computing. On the one hand, individuals can be compared at a glance based on their multi-dimensional genotypes, independent of the respective interpretation or phenotype. On the other hand, methods of visualization have been developed that capture characteristics of whole populations, allowing one to visually track the evolutionary process.

Pohlheim, for instance, presented a toolkit of convergence diagrams, 3D line plots, and 2D image plots, to visualize the evolution of fitness values and other individual attributes. Hart and Ross introduced a tree-based visualization to trace the ancestry of the best individual produced by an evolutionary run [6]. Daida et al. unfold genetic ancestry onto concentric circles on a 2D plane to create a compact and highly scalable visualization [4]. Wu et al. represent genotypes as sequences of color coded stripes whose colors correspond to different genes [12]. Keim et al. designed a system to visualize search queries on a (relational) database [8]. Data items that match the query most closely are arranged in the center of a spiral arrangement. This visualization technique can be used to relate individuals in an evolutionary system in arbitrary ways, e.g. by comparing fitnesses or individual attributes. In [9], Khemka and Jacob have closely investigated the possibilities to visualize population-based optimization processes at various levels of scale—from the individual to sets of experiments. They pro-

vide an easily adaptable user interface with various interactive manipulators to explore optimization processes across these scales.

3 The *EvoShelf* System

The interface of *EvoShelf* is divided into three window panes (Figure 2). The organization view on the left-hand side is used for selecting and grouping imported experimentation data (Section 3.1). The user's selection is shown in the browser view in the center pane. An inspector view (right-hand side) shows further details about an individual or an experiment. In addition to importing and inspecting functions, the toolbar at the top of the window gives access to built-in visualization methods which are explained in Section 3.2. Typically, a user of *EvoShelf* writes a plugin to import and visualize data for his respective evolutionary system, if it does not already exist. We provide details about plugins in Section 3.3.

EvoShelf makes use of lazy fetching of data. That is, images and attributes of an individual are not loaded until they are needed (such as when the user scrolls to them). When individuals go off screen, their data is unloaded. In this way, we have manipulated data sets with over $40,000$ individuals. Conceivably, *EvoShelf* can work with even larger datasets. To further increase the scalability of *EvoShelf*, high resolution images of individuals are loaded on demand—if no zoom is required, a low resolution image is displayed instead.

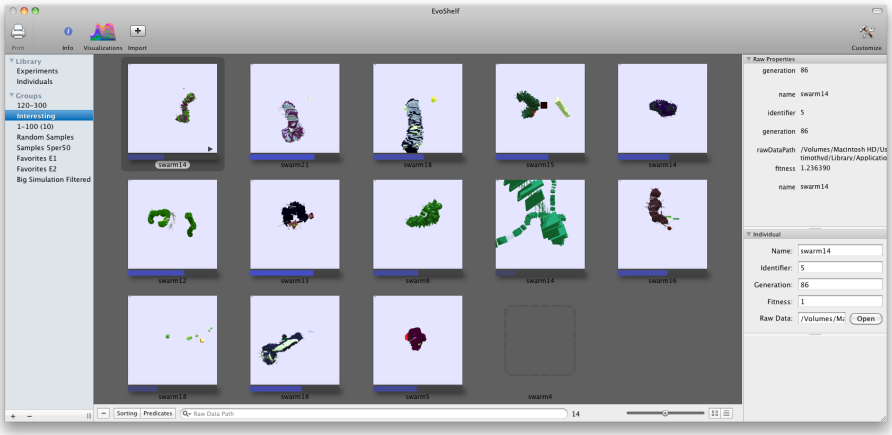


Fig. 2. The graphical user interface of *EvoShelf*.

3.1 Individuals, Experiments, and Groups

The organizational view to the left of Figure 2 is divided into a library section and a groups section. In the library section, the user can select either *Individuals*

or *Experiments*. In particular, the *Individuals* selection displays the images of all the individuals in the library, whereas *Experiments* shows representative thumbnails of all the imported experiments. The user can browse through the set of individuals of any experiment by hovering with the mouse over its thumbnail. The individuals of an experiment are revealed when the user double clicks on the experiment.

The data in the browser view can be sorted or filtered by the experiments' and individuals' attributes. Once the user has formed a suitable selection he can save his selection in a group, which would be equivalent to photo albums or playlists (as in[1, 2]). In Figure 2, a group labelled *Interesting* is selected, which hosts individuals from multiple experiments that the authors found of interest. Groups can be organized hierarchically. That is, one can form groups containing groups. When such a group is selected a union is formed from all of the individuals contained within the subgroups.

The controls at the bottom of the interface allow the user to remove individuals from a group or from the library, to sort individuals, to search for individuals according to arbitrary attributes (such as fitness value or generation), to scale the size of the images displayed, and to change the display mode. One display mode shows individuals as a collection of images, another one lists them in tabular format. The latter view is convenient for sorting and searching through individuals based upon numeric or textual attributes.

One individual is selected in the browser view in Figure 2. The image representing the individual was generated by the evolutionary system used as a test run for *EvoShelf* (see Section 4). In the given case, a play button (a right pointing arrow) allows one to re-run the simulation that produced and/or evaluated the selected individual. The button is not shown if the plugin for the particular evolutionary system does not support this option, and it only appears when the user hovers the mouse over the image.

Below the images in the browser view in Figure 2, blue bars represent the individuals' fitnesses. The bar is scaled to the minimum and maximum fitness of all the individuals currently displayed in the browser view. The higher the fitness, the brighter and longer the bar. No image is provided for *Swarm35* indicating that the genotype data was successfully imported but no image was found—in the given case, the simulation was terminated before a screenshot would have been taken.

The inspector view displays several default properties about the imported data, such as the file name of an individual or experiment. A custom interface for the inspector can be defined via the plugin architecture (Section 3.3).

3.2 Built-in Visualization Techniques

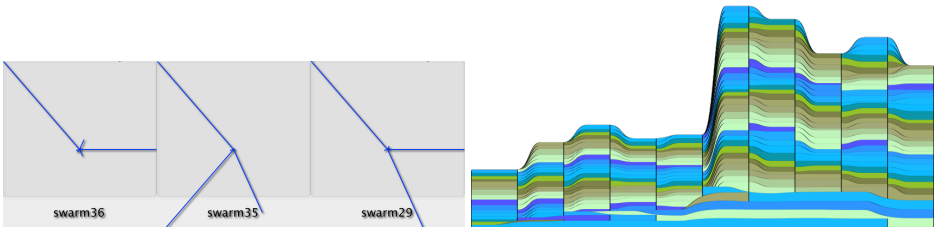
EvoShelf employs two basic built-in visualization techniques: star plots of name-value pairs [9] and FitnessRiver, a derivative of the ThemeRiver^{TM} method, which integrates local numeric values with global trends [7].

A star plot in *EvoShelf* visualizes a set of name-value pairs as a series of radially arranged line segments (Figure 3(a)). The length of a line segment is

representative of an attribute's value and it is normalized to the attribute's maximum value in respect to the selected individuals. An attribute's line segment will consistently appear at the same location in a star plot to render individuals comparable.

The ThemeRiver^{TM} visualization method produces a stream diagram that is read from left to right. Currents in the stream represent individual themes that occur, grow and decay over time. Instead of separating equivalent attributes into individual currents of a stream diagram, our FitnessRiver visualization method stacks the fitness values of individuals on top of each other. The fitness of an individual is proportional to the width of its current. Different colors are used to distinguish between successive individuals. Discontinuing currents indicate the removal of an individual from the evolutionary process. In the FitnessRiver visualization the x-axis represents the sequence of generations. A flat baseline is used so that the user has a greater sense of the progression of the fitness evolution (Figure 3(b)).

In Figure 3(b) we can see a large jump in the overall fitness at about the middle generation. When we look closely, we see that there are a few very successful individuals in the previous generation. We can see how these individuals likely contributed to the next generation. Furthermore, the majority of individuals in the new generation have noticeably more fitness than those in the previous generation.



(a) (b)

Fig. 3. (a) Individuals are comparable based on their star plots. (b) The FitnessRiver visualization shows the evolution of local and the global fitness. It is an adaptation of ThemeRiver^{TM} [7].

3.3 Plugins

A user can define additional import modules, visualization modules, data models, and finally custom inspector views for custom data models[1]. A few basic classes are provided for these modules and models that serve as plugin templates. The importing process, including control over import dialogue windows, can be adapted and alternative visualization modules can be subclassed from the *EvoShelf* visualization view controller class.

[1] *EvoShelf* plugins are written in Objective-C and should use the Cocoa API [3].

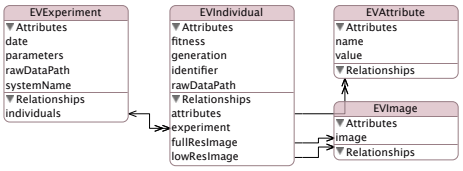


Fig. 4. The default data model for importing and managing *EvoShelf* data.

The default data model (Figure 4) is well suited for evolutionary algorithms (EA) and other forms of heuristic computation, such as particle swarm optimization (PSO). For instance, each step in a PSO simulation could correspond to a generation in an EA. This could however, generate a significant amount of individuals, in which case one might prefer to only import the final individuals from the system. In order to adapt the data model for differently organized information, the *EvoShelf* data model needs to be adapted. For instance, the attributes for the classes *EVExperiment* and *EVIndividual* need to be adjusted to fit the given experiment. The new attributes automatically determine the searching and sorting options in *EvoShelf*, as well as the information provided by the inspector view. In case a more elaborate inspector view is desired, an interface constructed in Apple's WYSIWG Interface Builder application can be loaded.

4 Example Scenario

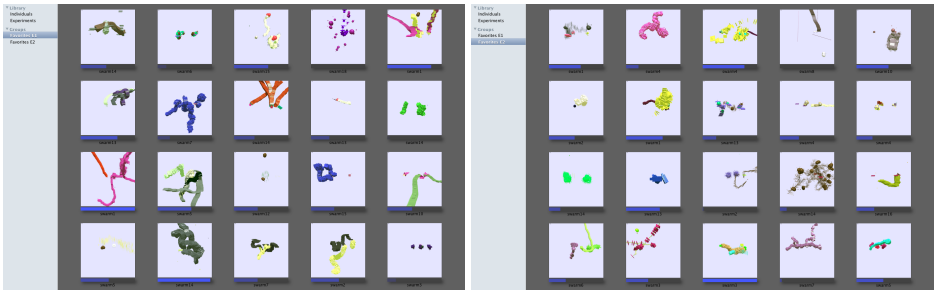
In this section, we explore the use of *EvoShelf* with a preexisting evolutionary system. In the evolutionary system of choice, Swarm Grammars (SGs) are bred by means of a Genetic Programming algorithm to produce architectural idea models [11]. SGs are a swarm-based developmental model in which production and interaction rules guide the movements, constructions and the reproduction of agents in 3D space.

In a subdirectory for each generation, the genotypes are stored as text files and snapshots of the corresponding phenotypes as images. Fitness evaluations for the individuals are stored in an additional file. When importing all the individuals, including their image representations and their meta-data into *EvoShelf*, the original directory structure is automatically copied into *EvoShelf*'s database.

Figure 5(a) shows a set of interesting SG specimens. We want to emphasize that due to their partially very low fitness values (swarms 3, 6, 7, and 18), we would have very likely not inspected these phenotypes without relying on *EvoShelf*'s visual browsing functionality. Based on these undervalued, interesting phenotypes, we were able to improve the fitness function that drives the SG evolution. In particular, we shifted the geometrical focus of the fitness evaluation in respect to the SGs' constructions to better suit the favored ones.

We also used *EvoShelf* for a semi-interactive evolutionary process by repeatedly selecting and exporting interesting individuals, modifying the fitness function and parameters to the GA, breeding their offspring for a fixed number of generations and importing the outcome (Figure 5).

We discovered that the SG GP evolution usually converged prematurely after at most several hundred iterations. Figure 6 shows the FitnessRiver plot over



(a) (b)

Fig. 5. (a) 20 interesting individuals are selected from an experiment and served as the initial generation for a (b) follow-up experiment.

300 generations. Overall 20,000 individuals were computed and imported into *EvoShelf*. We noticed that the overall fitness of our individuals had stagnated by the $100th$ generation (there is a very slight improvement in fitness past this point). Figure 7 confirmed our assumption of over-fitting: Up to the fitness stagnation at around generation 100, we randomly chose and plotted one of the ten best individuals every ten generations. For the period afterwards, we plotted one of the ten best individuals at random every 20 generations. And indeed, the phenotype images in combination with the star plots reveal a one-sided development, most easily recognizable by the inverted T-shaped star plots. Upon closer investigation, this similarity corresponds to the deployed amounts of two out of three construction elements provided to the SG agents (rods and layers), and the amount of construction elements that were placed outside of the intended target area. As the latter construction elements reduce the fitness of an individual, their increase might explain the fitness fluctuation as observed in Figure 6.

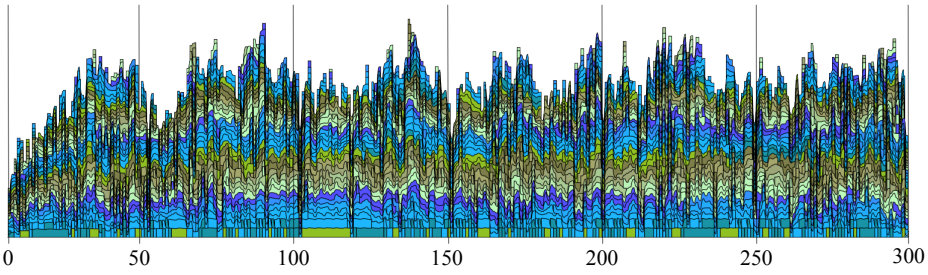


Fig. 6. The FitnessRiver plot shows stagnating and fluctuating fitness development after about 100 generations. The vertical lines denotes each $50th$ generation.

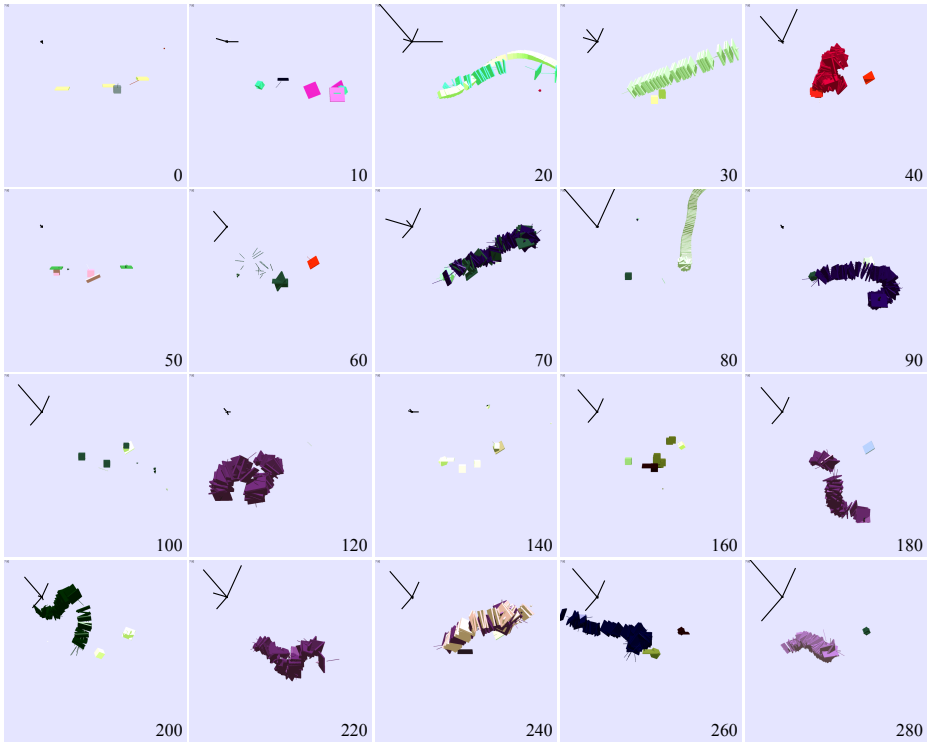


Fig. 7. First, every ten generations, then (2*nd* half) every 20 generations, a star plot and phenotype of a randomly selected individual is shown.

5 Summary and Future Work

We presented *EvoShelf*, an easy-to-use application for managing experimental data produced by arbitrary evolutionary systems. *EvoShelf*'s user interface is similar in its simplicity to mainstream media-browsers. Fast browsing of supplementary images associated with each specimen or of generic visualizations of the individuals, for instance by means of star plots, enables the user to retrace and interactively explore vast amounts of data produced evolutionary experiments. Storing, retrieving and ordering experimental data is facilitated by a simple yet powerful search function that considers the specimens' attributes and meta-data (generation, fitness, etc.). Hierarchical grouping structures further facilitate the management of large amounts of experimental data. In addition to the built-in management and visualization methods, *EvoShelf* can be extended by plugins that implement the required import, visualization or introspection functionalities. According programming templates are provided that can be easily adjusted or majorly extended, depending on the user's demands.

We applied *EvoShelf* on an evolutionary application that breeds Swarm Grammars to generate architectural idea models [11]. Due to the convenient and

fast browsing functionality of *EvoShelf*, we have been able to identify specimens that received low fitness values despite their appeal. As a consequence, *EvoShelf* helped us to adjust the fitness function of the SG GP system to better suit our expectations. By means of the visualization techniques that come with *EvoShelf*, FitnessRiver and star plots, we have been able to track and investigate an over-fitting process in our evolutionary runs. Finally, by using the selection and storing capabilities of *EvoShelf*, we have been able to introduce interactivity into an otherwise autonomous evolutionary process.

In the future, we would like to add more visualization plugins, as well as extend the current visualizations. Several improvements are possible in respect to the deployed visualization techniques. For instance, it should be possible to overlay different individual-based visualizations as we have done in Figure 7. The star plot visualization that we applied should be extended to improve its readability— possibly by an underlying, grayed out, partitioned circle, different coloring schemes, or line strengths. Overall, we found it would be useful to automatically associate representative specimens with global trends, as attempted by the combination of Figures 6 and 7. The FitnessRiver visualization could possibly be extended to also track the application of genetic operators and the course of inheritance. We would also like to explore importing data from an evolutionary system as it runs. Taking this a step further, one could also use *EvoShelf* as the basic user interface for controlling a system that uses interactive evolution as found in [9].

References

1. Apple Inc. Apple - iphoto. `http://www.apple.com/ilife/iphoto/`, April 2010.
2. Apple Inc. Apple - itunes. `http://www.apple.com/itunes/`, April 2010.
3. Apple Inc. The cocoa framework for mac os x. `http://developer.apple.com/cocoa/`, April 2010.
4. J. Daida, A. Hilss, D. Ward, and S. Long. Visualizing tree structures in genetic programming. *Genetic Programming and Evolvable Machines*, Jan 2005.
5. Google. Picasa photo editing. `http://picasa.google.com/`, April 2010.
6. E. Hart and P. Ross. Gavel-a new tool for genetic algorithm visualization. *Evolutionary Computation*, Jan 2001.
7. S. Havre, B. Hetzler, and L. Nowell. Themeriver tm: In search of trends, patterns, and relationships. *IEEE Transactions on Visualization and Computer Graphics*, Jan 2002.
8. D. Keim and H. Kriegel. Visdb: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, Jan 1994.
9. N. Khemka and C. Jacob. Visplore: a toolkit to explore particle swarms by visual inspection. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 41–48, New York, NY, USA, 2009. ACM.
10. Nullsoft. Winamp media player. `http://www.winamp.com/`, April 2010.
11. S. von Mammen and C. Jacob. Evolutionary swarm design of architectural idea models. *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, Jul 2008.
12. Wu. Visual analysis of evolutionary algorithms. *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 2, 1999.