University of Nevada, Reno

Interactively Evolving User Interfaces

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

by

Juan C. Quiroz

Dr. Sergiu M. Dascalu / Thesis Co-Advisor

Dr. Sushil J. Louis / Thesis Co-Advisor

May, 2007

UNIVERSITY
OF NEVADA
RENO

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

JUAN C QUIROZ

Entitled

Interactively Evolving User Interfaces

be accepted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

[illegible]

Sergiu M. Dascalu, Ph.D., Co-Advisor

[illegible]

Sushil J. Louis, Ph.D., Co-Advisor

Swatee Naik

Swatee Naik, Ph.D., Graduate School Representative

Marsha H. Read

Marsha H. Read, Ph.D., Associate Dean, Graduate School

May, 2007

ABSTRACT

Interactively Evolving User Interfaces

by

Juan C. Quiroz

We attack the problem of user fatigue in using an interactive genetic algorithm to evolve user interfaces in the XUL interface definition language. The genetic algorithm combines a set of computable user interface design metrics with subjective user input to guide the evolution of interfaces. User interface specifications are encoded as individuals in a genetic algorithm's population and their fitness is computed from a weighted combination of user interface design guidelines and user input. We show that we can reduce human fatigue in interactive genetic algorithms (the number of choices needing to be made by the designer), by 1) only asking the user to pick two user interfaces from among ten shown on the display and 2) by asking the user to make the choice once every t generations. Our goal is to provide user interface designers with a tool that can be used to explore innovation and creativity in the design space of user interfaces and make it easier for end-users to further customize their user interface without programming knowledge.

Acknowledgments

I would like to thank my family, but especially Katie, for putting up with me and keeping me sane through the countless hours of work.

I would like to thank my advisors, Dr. Louis and Dr. Dascalu, for giving me the opportunity to work with them, for their guidance, for the work station I have made my second home, and for the big fat check I get every month. I would also like to thank Dr. Naik for agreeing to endure 70+ pages of ... uh ... good stuff!

Finally, I would like to thank my ECSL lab members - Chris, Ryan, Anil, Adam, David, Nathan, and Mark, and I guess other random people that stray into the lab every so often, such as Sebastian and John.

This material is based in part upon work supported by the Office of Naval Research under contract number N00014-03-1-0104 and in part upon work supported by the National Science Foundation under Grant No. 0447416.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

User interface (UI) design is an expensive, complex, and time consuming process largely driven by documented style guidelines and design principles. Many of these guidelines and design principles are difficult to translate into code and good UI design is driven in large part by human aesthetics in their look and feel [1, 2, 3, 4]. Aesthetics are crucial to good user interface design because they can improve acceptability, learnability, comprehensibility, and productivity of users interacting with the interface [5]. We believe that the right balance of usability design principles and aesthetics is achieved through an iterative exploration of designs. The continuous search for creativity in user interface design is also fueled by the fact that *"very little knowledge in design generalizes beyond specific case studies"* [6]. Thus UI designers tend to be guided both by objective measures gleaned from UI style guidelines and design principles, and by subjective measures such as the *"look"* and *"feel"* of an interface. A technique or tool that streamlines user interface design and maintenance can revolutionize the industry and improve the end-user experience.

The central claim of this thesis is that

> *Interactive genetic algorithms can evolve user interface designs, combining computable expert knowledge metrics and human aesthetics.*

Interactive genetic algorithms (IGAs) differ from genetic algorithms (GAs) in that objective fitness evaluation is replaced with user evaluation. As such, they can incorporate intuition, emotion, and domain knowledge from the user. Through human subjective evaluation we can instill the UI designer's sense of aesthetics into the evolutionary process. While IGAs are a powerful tool, their reliance on user computation presents the issue of user fatigue. GAs usually rely on the use of large population sizes running for hundreds of generations to achieve satisfactory results [7]. Such computational dedication cannot be expected from the user due to psychological and physical fatigue. Thus, how best to incorporate user input into the IGA process remains a significant research challenge [8].

Our work differs in that we use both a computable fitness criterion and user evaluation to compose a combined fitness. We encode user interfaces as individuals in an IGA, and run over a number of generations to help explore the space of UI designs. Periodically, the UI designer sees the phenotypes (the UIs) corresponding to a small subset of the population and picks two - the best and worst looking interfaces. Empirical observations tell us that we should not display more than nine or ten items to be judged by a user but the composition of the subset displayed for user evaluation creates rich dynamics that affect the convergence behavior of the population in the genetic algorithm [9]. In this work we combat fatigue by asking the UI designer to evaluate a small subset (e.g. nine) of the entire population (e.g. 100). Other methods used for user evaluation are ranking of individuals or assigning a numeric value to the individuals displayed for user evaluation [8]. Instead we effectively reduce user input by asking the user to make two picks (best and worst), and through interpolation assign a fitness to every other individual in the population. Not only does this reduce the amount of clicks made by the user, but it also lessens the cognitive load on the user. Furthermore, instead of asking for user input every GA generation, we only ask for input every t^{th} generation, where $t > 1$. This further reduces user fatigue. We also explore how the frequency of user input, by asking user input every t^{th} generation, affects the IGA performance and user fatigue.

We address two issues that affect convergence behavior. First, should we show the user the *top* n individuals in the population, a mixture of the n best and worst individuals in the population, or display a random set of n individuals in the population? Second, how often do we need to ask the user for feedback? Our results show that displaying the *top* n individuals results in faster convergence and better interfaces. In addition, we show that asking for user input frequently leads to effective user bias. On the other hand, less frequent user input reduces user fatigue. We show that the effect of varying the degree of user input, and hence the user control over the evolution, between a simulated user and three real users varies. With a simulated user less frequent user feedback introduces noise and leads to suboptimal solutions. With three real users, less frequent user input significantly reduced user fatigue than when asking for user input every generation.

1.1 Structure of this Thesis

Chapter 2 covers previous work on user interface evolution and user fatigue in interactive genetic algorithms (IGAs). We present background information on genetic algorithms and explain how interactive genetic algorithms differ. The challenges of UI design, design style guidelines, and our choice of XUL as the target language for our user interfaces are also discussed.

Chapter 3 describes the process by which we were able to evolve user interfaces. We illustrate how UIs are encoded as individuals in the population of an IGA, how we conduct the fitness evaluation of individuals, and the advantages and disadvantages of the representation used in our approach. We also explain in detail the integration of objective and subjective criteria during fitness evaluation to guide the evolution of UIs.

Chapter 4 describes the environment used to evolve user interfaces. We discuss our experiences in the trade-off between investing time on a research environment to improve productivity versus just conducting experiments.

Chapter 5 presents the first set of experiments conducted with a greedy simulated user. We explore the most effective selection algorithm for UI evolution, who to display from the population for user evaluation, and how varying the frequency of user input affects the population dynamics.

Chapter 6 discusses experiments conducted with three real users. We explore how users effectively bias UI evolution and how the frequency of asking for user input every t^{th} generation affects interactive evolution.

Chapter 7 summarizes the work presented and the main contributions of this thesis. We also outline promising directions of future work.

Chapter 2
Previous Work

In this chapter we introduce the foundations for the work presented in this thesis. We explore and explain how genetic algorithms and interactive genetic algorithms work. The use of interactive genetic algorithms presents us with the problem of user fatigue. We discuss how user fatigue arises and its effects on interactive genetic algorithms, the importance and challenges of user interface design, and related work on both evolutionary techniques for user interface design and on mitigating user fatigue. Finally, we explain our choice of XUL as the target language for our evolved user interfaces.

2.1 Genetic Algorithms

Genetic algorithms (GAs) are a search technique based on the Darwinian principles of natural selection and survival of the fittest [10, 11]. A genetic algorithm consists of a population of potential solutions to the problem to be solved. We determine the fitness of individuals in the population based on how well each possible solution solves the problem at hand. GA solutions are usually encoded as bit strings (110001010), which are referred to as chromosomes.

The canonical GA process is shown in figure 2.1. The GA starts with the creation of a population of random individuals, that is, random potential solutions to the problem to be solved. Then we evaluate the fitness of the randomly created individuals. Next parents are selected proportional to fitness. We crossover the selected parents, with the offspring replacing the parents. After crossover, we introduce a small probabilistic mutation to the offspring. The new population then repeats the cycle of selection, crossover, and mutation, for a number of generations until some terminating condition has been met.

Suppose that we are using a GA to solve the OneMax problem, where the goal is to maximize the number of 1s in a bit string. We can assign fitness to an individual directly proportional to the number of 1s in an individual's chromosome. An individual with a chromosome 001011 of length 6 would get a fitness of 3. During the initialization step we would create a population of random chromosomes of length 6, with a maximum possible fitness of 6. We would then count the number of 1s in each chromosome to figure out the fitness of each individual. Then parents with the most 1s would be more likely chosen for crossover, due to their higher fitness. For example, an individual with a chromosome 111110 would be more likely selected than an individual with a chromosome 001000. The generational process would continue until the population has converged to the optimal chromosome 111111.

Selection algorithms used to pick parents have been extensively studied as to their effects on population dynamics [12, 13, 14]. Two widely used selection algorithms, and of relevance to the work presented here, are roulette wheel selection and tournament

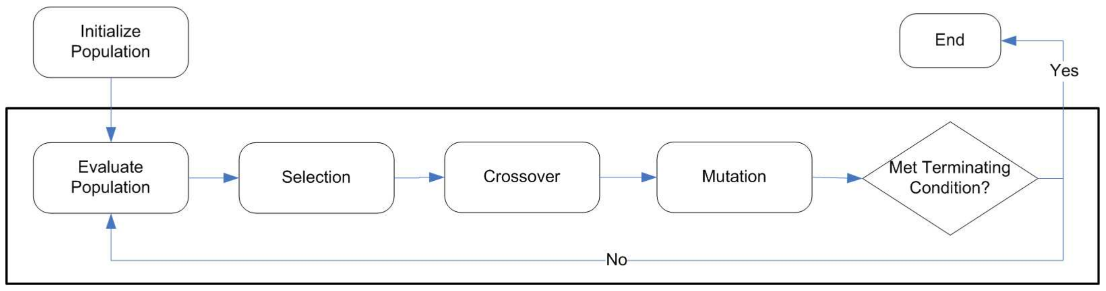


Figure 2.1 Canonical GA

selection. In roulette wheel selection, or fitness proportional selection, individuals are assigned to a pie slice, whose size is proportional to the individual's relative fitness value. We then pick a random number to land in a point in the *"roulette."* The individual whose pie slice contains the random number is selected to be part of the mating pool. Hence, higher fitness individuals will have bigger pie slices, increasing their probability of being selected multiple times for crossover. Nevertheless, lower fitness individuals have a small probabilistic chance of being selected to be part of the mating pool. In contrast, tournament selection samples n individuals at random from the population, where n is the tournament size. The individual in the tournament with the highest fitness wins and gets selected to be part of the mating pool. Larger tournament sizes increase convergence rate by killing off diversity in the population [14].

Crossover combines the genetic material of two individuals. Single point crossover, which is the method used in the canonical GA, is shown in figure 2.2.We choose a random crossover point, we slice the chromosomes of the two parents along the crossover point, and we glue the complementary slices of the parents to form the offspring. Crossover reflects the principle of individuals passing their good traits onto their offspring. By crossing over individuals with high fitness, we hope to get offspring which achieve an even higher fitness by building on their parents' genes.

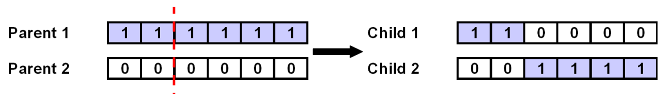


Figure 2.2 Single point crossover

After parents have been crossed over, we introduce a small probability of mutation to the offspring. Bit flip mutation, shown in figure 2.3, picks a random point in a chromosome, and flips its value from 0 to 1 or from 1 to 0. Mutation is a source of diversity in the gene pool of the population. While crossover shuffles the gene pool, mutation modifies it by introducing new genes.

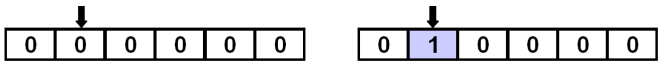


Figure 2.3 Bit flip mutation

2.2 Interactive Genetic Algorithms

Usually we can compute the fitness of individuals in a GA based on a math equation, some computation, or a model [7]. However, a user cannot be trivially modeled; user preferences are relative and subject to change with time and context. Interactive genetic algorithms (IGAs) incorporate user subjective evaluation by replacing the fitness evaluation with the user, where the user provides the fitness to individuals in the population by assigning a number on a subjective scale, ranking individuals, or choosing the best individual from a displayed subset [7, 8]. Because of the nature of IGAs, they have been used for a variety of applications which incorporate creative human input, including editorial design, industrial design, image processing, database retrieval, graphic art and computer graphics animation, control and robotics, among others (see the work of Takagi for a survey on IGAs [8]). IGAs fuse the power of evolutionary computation and human subjective evaluation by providing a mapping from psychological space to parameter space [8]. By doing so, IGAs incorporate human knowledge, emotion, intuition, and preference into GAs.

The IGA process is shown in figure 2.4. After the population has been initialized randomly, we display a visual representation of each solution (each individual) to the user. Improper visualization of the problem can penalize IGA performance, because complex or ambiguous representations make it harder for the user to understand and quantitatively compare the solutions presented [7]. If the user is satisfied with one of the solutions displayed, then the process stops. Otherwise, we use user feedback to assign fitness to individuals. Once fitnesses are assigned, we go through the standard process of selection, crossover, and mutation. Finally, the new population is displayed to the user for evaluation. In figure 2.4, nine individuals are presented to the user for evaluation, where each individual represents a color. Alternatively, we could have displayed to the user the color of each individual in the form of an RGB [15] vector, such as (240, 10, 98), but rendering the color to the user, instead of the vector representation, is a more effective and intuitive problem visualization.

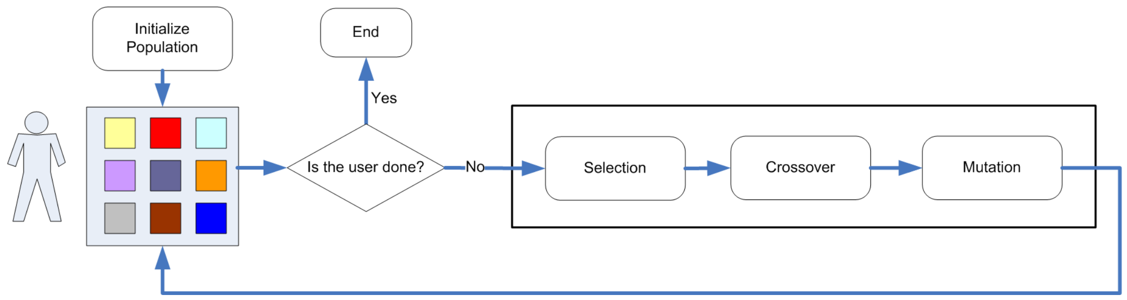


Figure 2.4 Interactive genetic algorithm

Effective IGAs have to overcome several issues. GAs usually rely on large population sizes running for many generations, but asking a user to make hundreds or thousands of choices may be a little unrealistic. A user would rapidly fatigue and/or lose interest. Furthermore, the subjective nature of human input can lead to users changing their goals through the IGA run, leading to noisy landscapes - which coupled with user fatigue can lead to suboptimal solutions [7].

UI design, discussed more in detail in the next subsection, is a process which combines objective and subjective heuristics. As such, an IGA is a suitable tool which uses evolutionary techniques, coupled with human input to help guide the evolution of the population.

2.3 User Interface Design

User interface design is a complex process critical to the success of a software system; designing interactive systems that are easy to use, engaging, and accessible is a challenging task. Consequently, the design of a user interface is a major and costly part of any software project.

Graphical user interface development toolkits and libraries help user interface designers to develop graphical user interfaces (GUIs) faster by providing basic widget elements, such as menus, buttons, and textboxes. Because GUI toolkits and libraries facilitate the design activities at too low a level, they may allow the designer to create a bad or poor design quickly [16]. UI designers therefore also use style guidelines and design principles to guide their designs and this hopefully leads to more usable interfaces. In addition, such guidelines and design principles provide a means with which to evaluate a generated design. Style guidelines not only define the look and feel of a user interface, but they also address the organization of widgets, the use of color, the use of font, the use of spacing and margins, among other properties. Some prominent style guidelines are Apple's Human Interface Guidelines, Microsoft's User Interface Guidelines, Sun's Java Look and Feel Guidelines, and the GNOME Human Interface Guidelines [1, 2, 3, 4]. The problem lies in that *"interpreting the guidelines unambiguously and applying generic principles to a particular design problem is itself a major challenge"* [16]. There is also the problem that guidelines are either too specific or too vague, so they do not always apply to the problem at hand. For example, an excerpt from Apple Human Interface Guidelines specifies: *"use color to enhance the visual impact of your widgets,"* but no detail is given as to which color to use for a given widget and context [1]. Therefore, user interface designers are forced into making subjective decisions and evaluations to fill in the details that guidelines omit.

We define usability as *"the extent to which a computer system enables users, in a given context of use, to achieve specified goals effectively and efficiently while promoting feelings of satisfaction"* [17]. We are interested in the development of aesthetically pleasing UIs, because aesthetics can improve acceptability, learnability, comprehensibility, and productivity of users interacting with the interface [5]. A study had demonstrated that *"designers are biased towards aesthetically pleasing interfaces, regardless of efficiency"* [17]. This presents a problem, because the right balance between aesthetics and efficiency in a UI design leads to the best end-user experience. An aesthetically pleasing UI with an obscure menu organization can make it frustrating to the user to find and explore the utilities of the UI. Through our evolutionary approach, we incorporate both expert knowledge and user bias, leading to the fusion of guidelines of style and user preference, to make beautiful, efficient UIs.

2.4 Related Work

Our work addresses research challenges from two fields: the incorporation of user input into IGAs, and the use of evolutionary techniques for UI design.

2.4.1 Evolution of User Interfaces

Oliver et al. and Monmarché et al. explored the evolution of the appearance and layout of websites [18, 19]. The user evolves either the style or the layout of a webpage; these two optimizations are separated in order to simplify the evaluation of individuals. The user guides evolution by picking the individuals the user likes, then replacing the rest of the individuals by mating and applying high mutation rates to the user selected individuals. CSS parameters like font size, font color, font family, link color, and text alignment were evolved in their experiments. We expand on this work in two ways. First, our research incorporates expert knowledge (in the form of style guidelines) in addition to incorporating the subjective evaluation by a user. Second, they used a population size of 12 individuals in order to display and fit all individuals on a screen. Instead we use large population sizes and display a small subset for user evaluation, allowing us to sample the space of UIs more effectively. Finally, we are interested in WIMP interfaces (window, icon, menu, pointing device) instead of web interfaces. WIMP interfaces have a more interactive nature, in contrast to web interfaces which tend to be content oriented [17].

2.4.2 User Fatigue in IGAs

Interactive genetic algorithms are a suitable tool for problems where *"there is no clear measure to give the evaluation of fitness other than the one in the human mind"* [20]. This applies to the evolution of UIs because users will be evolving UIs based on a mental model. Takagi identifies reducing human fatigue in the use of IGAs as the major remaining problem [8]. We show that users can guide the evolution of user interfaces, and are able to evolve interfaces to their liking by only selecting the best and worst individuals from a small subset of the entire population, instead of having to evaluate or rank all individuals in the population.

Llorá et al. make the user pick the best solution from a small subset of the population displayed [7]. The displayed subset is a tournament used to define partial ordering of solutions; given that $s1$ and $s2$ are shown to the user, and the user picks $s1$, then we assume that the fitness of $s1$ is greater than the fitness of $s2$ [7]. The partial ordering of solutions, from the winners and losers of the tournaments, is used along with the dominance concepts of multi objective optimization to induce a complete ordering of solutions, which is subsequently used to train a support vector machine (SVM) to learn the user's preferences [21, 7]. For an in-depth discussion and applications of support vector machines see the work of Gunn, Burges, and Bennett and Campbell [22, 23, 24].

In the work presented in this thesis we do not attempt to do any user modeling with machine learning techniques. Instead, we use a simple interpolation based on the user selection of the best and worst UIs to determine the fitness of every other individual in the population. Thus we reduce the user input to two decisions every generation. Furthermore, as in Kamalian's, work we have the user evaluate a subset of the population every t^{th} generation, putting the user in a supervisory role and thus reducing the amount of feedback needed from the user [25]. How to choose a good value for t is addressed in chapters 5 and 6. The work presented by Kamalian et al. also allows the user to give either a promote or demote reaction to individuals displayed for user evaluation [25] . In addition, they use a validity constraint to determine viable and meaningful designs to be displayed to the user. While individuals matching the validity constraint can be numerous, we explore the effects of displaying a small subset of the population for user evaluation and how the individuals selected as part of the subset affect the IGA's performance.

2.5 XUL User Interfaces

The target language used for the UIs being evolved is XUL, the XML User-interface Language, a cross-platform markup language for user interfaces [26]. XUL is a powerful and extensive language allowing the defining of widget appearance through CSS style sheets and the use of JavaScript to implement widget behaviors [26]. XUL was chosen as the target language because of its flexibility and the ease with which widgets can be manipulated. XUL is also suitable for the manipulation necessary to evolve the structure of UI layouts. The syntax and structure of XUL allow us to create a wide range of applications, from a simple layout consisting of two buttons, to a full fledged application consisting of a menubar, toolbar, and other common widget elements. Lastly, as a subset of XML, we can use XML parsers and libraries to handle the manipulation of our XUL UIs.

Chapter 3
User Interface Evolution

How do we encode user interfaces into a representation that can be used by the IGA? How do we layout widgets in a panel? Should we evolve the x and y coordinates of each widget, or do we evolve the position of widgets relative to their neighbors? We address these questions and how we implemented the solution to these questions in our IGA. We will discuss the representation used for our user interfaces, how widgets are laid out on a panel, and the advantages and disadvantages of our implementation.

3.1 User Interface Representation

We encode the UI representation in two chromosomes (figure 3.1). One chromosome encodes widget layout organization, and the second chromosome encodes widget characteristics (such as widget color). We organize the widgets on a 10 rows by 2 columns grid. In user interface design a sizer usually manages widgets, and a grid sizer allows efficient widget organization in a layout. The grid layout also enforces alignment of widgets, which is a style guideline in UI design. We avoided widget encoding as a bit string since standard genetic operators such as crossover or mutation could potentially destroy the representation by introducing duplicate widgets. To avoid this problem, we encode the widgets in an integer permutation string, of size 20 (10 rows by 2 columns), where each integer represents a unique identifier for each widget and 0s represent empty cells filled with spaces. The integer string maps to the 2D grid representation in a row major fashion. We chose the 10x2 grid because this results in UIs able to fit in the available space in our sample application: the Lagoon UI for the *MoveTo* panel explained in more detail in section 3.3.

To preserve the integer representation of the layout chromosome, we use PMX, partial mapped crossover [11]. PMX prevents duplicate widget insertion during crossover. We use swap mutation, where we randomly pick two genes in the integer chromosome and swap their values. The integer permutation representation used for the layout of the widgets also saves us from having to compute whether widgets overlap, a computational save of l^2 for each individual (widget layout chromosome of length l) in the population (of size n), and a total save of l^2n computation every generation. Hence we can explore widget layouts by permuting widget identifiers.

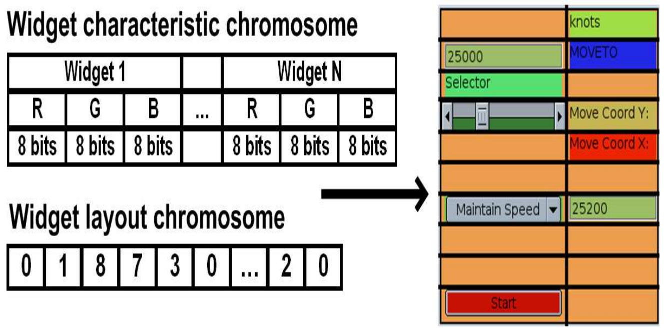


Figure 3.1 UI encoding consists of two chromosomes. The widget characteristics chromosome encodes the color of each widget in a bit format. The widget layout chromosome encodes the position of the widgets in the grid. Widgets are identified by integer IDs greater than 0 and empty cells in the grid are identified with 0s.

The second chromosome encodes widget characteristics (widget color) for each individual. This chromosome is a standard bit string and we use standard one point crossover and bit flip mutation on this part of an individual.

3.1.1 Widget Layout

We layout our widgets on a grid construct provided by XUL which allows us to organize our widgets in rows and columns.

We have tried using other layout organizations, including absolute positioning and positioning relative to other widgets. In absolute positioning we encoded the cardinal coordinates of our widgets, where the coordinates specified where in the panel the widgets were placed. While this was simple to implement, it resulted in widgets being placed on top of each other. This added another level of complexity to be resolved by the user by providing input into the system specifying that the UIs the user liked the best were the UIs with widgets not stacked on top of each other, instead of having the user concentrate on more useful characteristics, such as the actual widget organizations and the look and feel. We may return to this representation in the future.

Next we tried using relative positioning, where we encoded the relative positions of widgets with respect to the previous widget in the chromosome. The four positions allowed were left, right, up, and down. The first widget in the chromosome was placed on the middle of the panel, with each subsequent widget being placed relative to its predecessor in the chromosome. Without any bounds or overlap checking, we got cases where the widgets in the UI would almost line up in a straight line, resulting in elongated UIs that wasted screen space. Finally, the IGA still placed widgets on top of each other, since a widget placed to the left of a widget with a neighboring widget already on the left results in stacked widgets.

Although for the two previous representations we expect a GA to eventually untangle the layout, the permutation representation seems to be a more effective and elegant solution to the layout of the widgets.

3.1.2 Widget Color

We encode widget color on the widget characteristics chromosome. For the color we use the RGB color model, where each color is a combination of various degrees of red, green, and blue. The RGB components vary from 0 to 255 respectively. So red is $(255, 0, 0)$, green is $(0, 255, 0)$, and blue is $(0, 0, 255)$. Hence, we require 8 bits for each of the three main color components, with a total of 24 bits to represent the color of a single widget. This representation allows us to explore the 2^{24} space of colors for each widget.

The RGB model was chosen because of its support in CSS, which is how the characteristics of widgets are specified in XUL, the target language for our UIs. We could have used the HSV color model, but its gamut is the same as RGB, and experiments have shown that there is no significant efficiency difference in the RGB and HSV color models [15, 27, 28]. Therefore, we decided to stick to RGB, however we treat RGB colors as vectors in a 3D color-space.

3.2 Fitness Evaluation

Our IGA's fitness evaluation consists of two steps: (1) user input evaluation, and (2) objective metric conformance checking. In the first step we have the user make two selections, the UI the user likes the best and the UI the user likes the least. We use these two selected UIs to evaluate the subjective fitness component of all other individuals in the population through interpolation. In the second step the GA looks through the UIs in the population and checks to see how well they adhere to or violate coded guideline metrics. We then add the subjective and objective fitness components in a linear weighted sum. For this experiment we used equal weights for the subjective and objective fitness components.

3.2.1 Subjective Evaluation

We compute the similarity between two individuals in two steps. In the first step, we calculate color similarity of the two UIs, in terms of the widgets and the panel background. To determine color similarity, we calculate the euclidean distance between two colors. We reward a small distance between the widget color in individual i and the user selected best individual b. On the other hand, a large distance between the widget color in individual i and the user selected worst individual w is rewarded. Next, we compute widget layout similarity. Here we compute the hamming distance between the permutation layout chromosomes of the two individuals. This fitness is inversely proportional to the hamming distance between individual i and the user selected best b and directly proportional to hamming distance between i and the user selected worst. Finally, we scale the subjective component to make it comparable to the objective component.

We compute similarity between the best individual b and individual i and between the worst invididual w and individual i in the population as follows:

$$b_s = \sum_{k=1}^{m} \frac{M - dist(e_{b,k}, e_{i,k})}{M} + (MH - hamming(b, i))$$

$$w_s = \sum_{k=1}^{n} \frac{dist(e_{w,k}, e_{i,k})}{M} + hamming(w, i)$$

The term within the summation computes color similarity and the second line, the layout similarity. b_s is the subjective fitness component computed with reference to the user-selected best individual while w_s computes the subjective fitness component with reference to the user-selected worst individual. In the formulas above, M is the maximum distance between any two colors, $\sqrt{255^2 \times 3} = 441.68$ and $dist(e_{b,k}, e_{i,k})$ is the euclidean distance between the k^{th} widget of the best individuals and the k^{th} widget of individual i. MH is the maximum hamming distance ($l = 20$). We finally scale the subjective fitness to lie between 0 and 1000.

Lastly, we compute the subjective component as the sum of the color and layout similarity of individual i compared to both the best individual b and the worst individual w.

$$subjective = b_s + w_s$$

3.2.2 Objective Evaluation

We compute the objective fitness component by checking how well UI individuals in the population adhere to and respect coded style guidelines. Our first coded color style guideline checks whether a UI has a high contrast between background panel color and widget foreground colors. Maintaining a low contrast between widget colors is our second coded color style guideline. We prefer the high contrast between background and widget colors to ensure legibility. The low contrast between widget colors ensures that widgets have a similar shade of color, instead of having each widget in a UI with an independent color. The use of the grid positioning to layout widgets enforces their alignment, which is a style guideline too.

We iterate through the widgets of each UI layout and compute the euclidean distance from each widget color to background panel color to check high contrast between the background panel color and widget colors. We consider a large distance between widget j and the panel background color as a high contrast value. We sum all the euclidean distances, rewarding individuals that have a high euclidean sum. Next, we compare each widget j in a UI layout to every other widget (an l^2 computation) in the layout, taking their euclidean distances and adding them up. Large euclidean distance values between two widgets means that the widgets do not have a similar shade of color. We do this to cluster the colors in 3D space into a center of gravity which defines the color shade that all these colors should share in common. A large sum of the euclidean distances means that all widgets have very different colors, and hence they are spread out far from each other thereby violating our style guidelines. We therefore assign a low reward to such an individual. A small sum of the euclidean distances means that the widgets are clustered together and share a similar shade of color. This individual fulfills our style guideline and we therefore assign a high reward. We sum the rewards from the high contrast between widget colors and background color and low contrast between widget colors. Finally, as with the subjective fitness, we scale this objective value to also lie between 0 and 1000.

We compute how similar the color of widgets in a panel are as follows:

$$obj_1 = \sum_{k=1}^{m-1} \sum_{j=k+1}^{m} \frac{dist(e_{i,k}, e_{i,j})}{M}$$

We compute the contrast of widgets to the background color with the formula:

$$obj_2 = \sum_{k=1}^{m} \frac{M - dist(e_{i,k}, window\_bg_i)}{M}$$

Finally, we add the two objective computable metric values to obtain the objective metric:

$$objective = obj_1 + obj_2$$

After we compute the subjective and objective fitness components, we take a linear weighted sum of the two to determine the fitness of each individual:

$$fitness = w_1 * objective + w_2 * subjective$$

where w_1 is the objective component weight, w_2 is the subjective component weight, *objective* is the fitness objective component and *subjective* is the subjective fitness component. The weights w_1 and w_2 are complements of each other, with values between 0 and 1. We used values of 0.5 and 0.5 for w_1 and w_2 respectively for the experiments discussed in chapters 5 and 6.

3.2.3 Parasitism

We are evolving and trying to optimize the layout and the look of the widgets in a panel. Consequently, we have multiple criteria that we are trying to optimize. This has led to parasitic behavior on the evolution of UIs. The user picks the UI the user likes the best and the UI the user likes the least. However, the user does not specify these in terms of what exactly the selection is being made on. When the user picks a UI as the best, this leads to the GA attributing a high fitness to both the look and the layout of the widgets. For example, if the user picks a UI because of the vibrant blue colors the widgets have, then a high fitness will be attributed to whatever layout the widgets have.

In the current implementation we have not incorporated a means with which to prevent the emergence of this parasitic behavior. This could be suppressed by fixing either the layout or the look of the widgets, and evolving the other non fixed parameter. Alternatively, the user could be asked to select the best UI based on widget layout and the best UI based on widget look. However, this adds to the number of selections that have to be made by the user, thus increasing user fatigue.

3.3 Case Study: MoveTo Panel

Users participating in our IGA sessions, to be discussed in chapters 5 and 6, guided the evolution of the *MoveTo* interaction panel that controls combat ships in *Lagoon*, a real-time 3D naval combat simulation game developed in our lab [29]. The MoveTo panel (figure 3.2) consists of five text labels, a button, a drop-down menu, a slider, and two textboxes, all written in XUL and loaded into the IGA. We chose the MoveTo panel because it has a variety of widely used widgets, yet it is simple enough for our initial experiments.

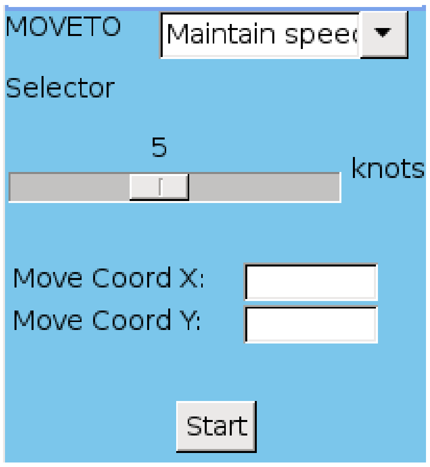


Figure 3.2 MoveTo panel in Lagoon

3.4 Summary

We have presented the encoding of our UIs using two chromosomes, one for the widget characteristics and one for the layout of the widgets. We discussed how individuals are assigned a fitness by combining objective and subjective heuristics. The user selection of the best and worst UI are used to interpolate the fitness of every other individual in the population, where interpolation is done via similarity to the user selected best and worst UIs. A shortcoming of our representation and user interaction is the emergent parasitic behavior, which can be resolved at the expense of increased user interaction.

Chapter 4
User Interface Evolution Environment

Our research software environment provides a front-end to the interactive genetic algorithm. We allow the user to configure the IGA behavior through the GUI. Our current environment provides limited functionality and does not address several usability and efficiency issues. For example, we use XUL, a markup language for UIs, as the target language for our UIs because of its flexibility and ease with which widgets can be manipulated [26]. Due to the limited support of XUL rendering with wxPython, our environment implementation language, we dump the IGA output to a file every generation to be viewed by the user through a system capable of rendering XUL. To visualize this file we then use the Mozilla web browser.

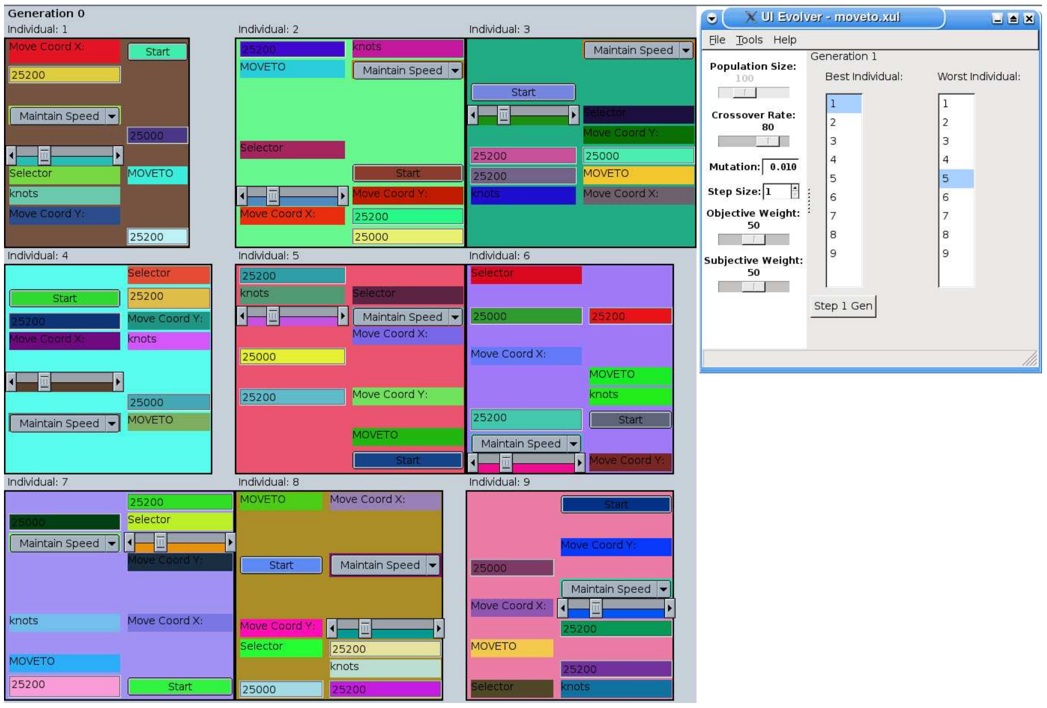
We present three modifications to the existing environment aimed at improving research productivity: (1) integration of XUL output into the main wxPython window; (2) a manager for specifying experiment runs; and (3) a manager for the analysis and visualization of data produced from the many experiment runs. We discuss how such improvements to the environment empower the user and increase research productivity by providing the user with an intuitive tool that reduces tedious tasks. We also look into the effort needed to improve the research environment and assess its worthiness versus alternatively spending this effort on actually conducting research experiments using the existing (less developed) environment. In other words, we discuss two different approaches: the first is to invest some time and resources to better prepare the research tools (and then conduct the experiments), the second is to focus immediately on conducting the experiments and advancing research (by using less elaborated, albeit operational research tools).

The long-term goal is to deploy our environment to user interface designers. However, the environment needs to be prepared for the context on which it will be used for our intended audience. We foresee two audiences, researchers and end-users. The current environment is tailored towards researchers. We discuss how we will go about moving our environment from a *"researcher's tool"* to an *"end user's tool,"* focusing on how we will conduct this transition.

We hope that the discussion presented in this chapter will help other researchers customize their experiment environments and tools developed for the end-users and thus strengthen the work of both the research and end-user communities.

4.1 Overview of Our Research Environment

The environment allows the user to configure the IGA parameters. As shown in figure 4.1, settings that can be modified include the crossover rate, mutation rate, population size, number of individuals to display, the objective and subjective weights of the fitness linear sum, and the frequency of user input. The user can further customize advanced options such as using roulette wheel or tournament selection for the IGA, and if tournament selection is chosen, the ability to specify the tournament size and the probability of choosing the winner of the tournament.



Population Size: 100
Crossover Rate: 80
Mutation: 0.010
Step Size: 1
Objective Weight: 50
Subjective Weight: 50

Figure 4.1 User interface evolution environment.

The IGA was implemented with Python and the GUI with wxPython. Python was chosen because it enabled us to do agile programming through fast prototyping, continuous refactoring, and iterative redesigns.

4.1.1 User Interface Specification

A user defines a UI to be evolved by writing the list of widgets to be evolved in XUL format. XUL, as a subset of XML, is intuitive and straightforward. A button in XUL is defined by “$< button\ label = `I\ am\ the\ label\ for\ this\ button!'/ >$.” The XUL list is loaded into the environment through a file dialog, and evolution can begin. Currently Python has limited support for XUL renderers. Consequently, we write the evolution output to a XUL file, and view the file with the Mozilla web browser, which uses the Gecko rendering engine to render XUL through the browser [26]. We made the web browser refresh every few seconds, to keep the user from having to refresh the web browser window every generation.

4.2 Investing Time on Development Versus Research Experimentation

We have worked on improving the effectiveness of our research environment by implementing three new features: (1) integrating the UI output into wxPython instead of writing it to a XUL file; (2) adding an experiment manager to handle numerous experiment runs and their organization; and (3) adding a data manager to navigate and visualize the large amounts of data generated by running many experiments.

4.2.1 wxPython Integration

As implemented (in a simpler way) in the previous version of our research environment, the dumping of the visualization of individuals to a XUL file presents efficiency and usability issues. First of all, both the Mozilla web browser and the environment window must be started every time in order to be able to view the population status and to provide relevant feedback to the IGA. Once Mozilla has been started, the user must then open the XUL file to which the output was sent. Lastly, the user has to constantly switch back and forth between the UI evolution environment itself (to enter the user input of the best and worst UI displayed) and the Mozilla browser (to see what the UIs at the current generation look like).

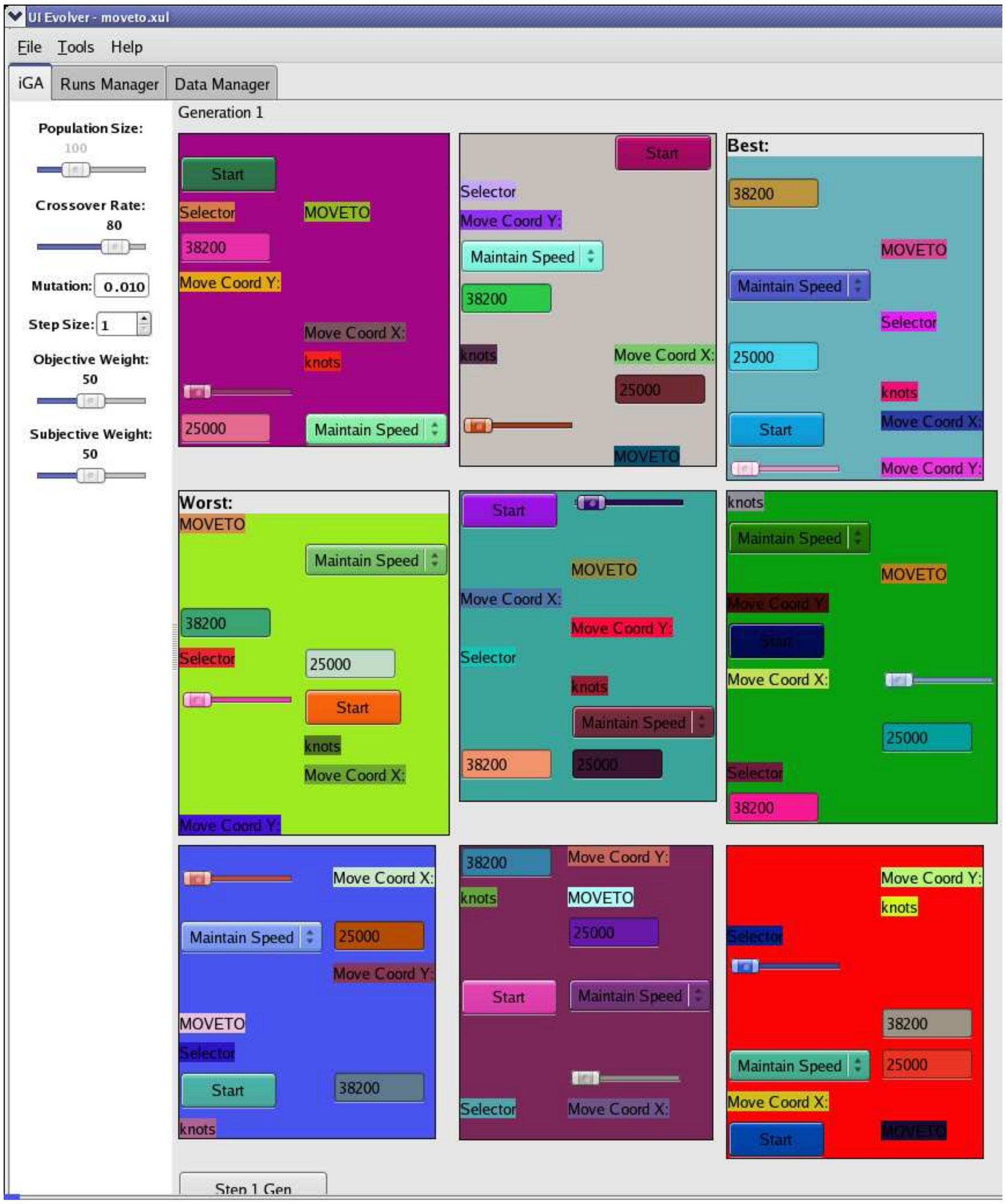
We propose a design solution to the aforementioned problem: to integrate in the main wxPython window the rendering of individuals being evolved instead of writing it to a XUL file. Alternatively, we could implement the entire GUI with XUL. The advantage of the latter approach is the ability to make the system available online and thus have users evolve GUIs through the web. However, the challenge in this is the communication overhead between the XUL widgets and the python IGA backend, which would require extensive processing. Therefore, we have opted for the former solution.

The integration into wxPython, illustrated in figure 4.2, has several advantages. First, the user does not have to keep switching back and forth between Mozilla and the main environment interface. Second, the user selection becomes intuitive and less error prone. A left double click on an individual selects it as the best UI design, and a right double click on an individual selects it as the worst UI design. Another advantage is that productivity improves through having the session run faster. With the XUL output viewed on Mozilla, the user had to switch windows and refresh the browser to see the latest subset for user evaluation. Instead, with wxPython integration the update is almost instant, speeding up the IGA session. Overall, the evolution of individuals and their visualization becomes straightforward.

4.2.2 Experiment Manager

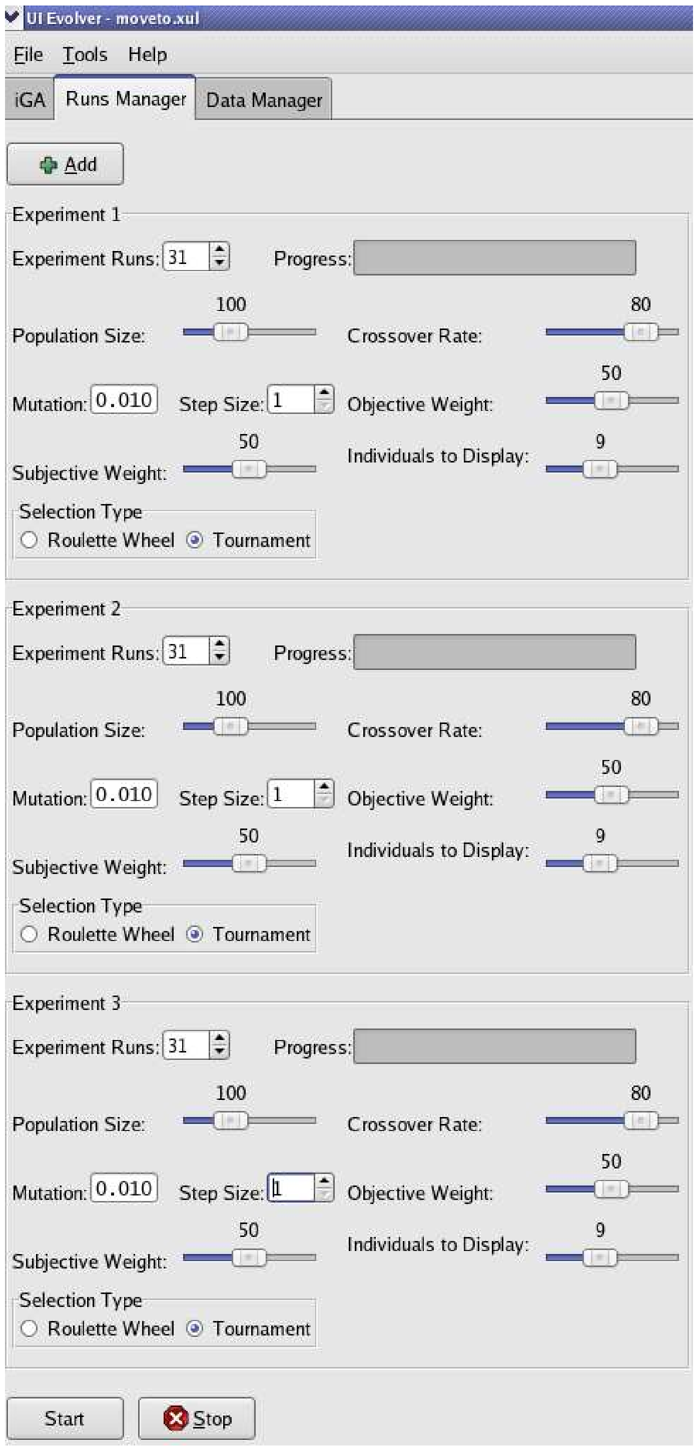
The second improvement was to add a manager of experiment runs. The interface of this experiment runs manager, shown in figure 4.3, allows the user to specify as many experiments as are desired and their configurations. The processing is parallelized, so that the user does not have to manually run the code in multiple machines. Configurations for an experiment include all settings that would usually be set through the main interface, as described previously in section 4.1. An experiment is added by clicking on the *"Add"* button. Clicking the *"Start"* button begins running the experiments, with a progress bar showing the status of each experiment as time goes by.

For the experiments to be discussed in chapter 5, an experiment consisting of 30 runs was done for each of the main results presented, which amounted to a lot of time setting up experiments in separate nodes in a cluster and tedious data management. The experiment runs manager, developed to increase research productivity, abstracts all that away, allowing the user to run as many experiments as necessary, with the ability to customize almost every aspect of the IGA for each experiment, parallelize the processing to have the experiments run as fast as possible, and automate the organization of the vast amounts of data resulting for each experiment.



Population Size: 100
Crossover Rate: 80
Mutation: 0.010
Step Size: 1
Objective Weight: 50
Subjective Weight: 50

Figure 4.2 wxPython integration



Experiment 1
Experiment Runs: 31 Progress:
Population Size: 100 Crossover Rate: 80
Mutation: 0.010 Step Size: 1 Objective Weight: 50
Subjective Weight: 50 Individuals to Display: 9
Selection Type
Roulette Wheel [✓ Tournament]

Experiment 2
Experiment Runs: 31 Progress:
Population Size: 100 Crossover Rate: 80
Mutation: 0.010 Step Size: 1 Objective Weight: 50
Subjective Weight: 50 Individuals to Display: 9
Selection Type
Roulette Wheel [✓ Tournament]

Experiment 3
Experiment Runs: 31 Progress:
Population Size: 100 Crossover Rate: 80
Mutation: 0.010 Step Size: 1 Objective Weight: 50
Subjective Weight: 50 Individuals to Display: 9
Selection Type
Roulette Wheel [✓ Tournament]

Figure 4.3 Experiment manager

4.2.3 Data Manager

The third improvement we have integrated in our environment is closely related to the experiment runs manager. The data manager allows the user to browse and explore the data produced from the many runs of each experiment intuitively. Each experiment and its corresponding runs are organized in a tree construct, with two visualization modes.

First, clicking on an experiment expands its children (the experiment runs) and displays an average plot of the results from the experiment. An example is shown in figure 4.4. Second, clicking on one of the runs from an experiment displays the data in a spreadsheet, as shown in figure 4.5.

Usually, a script is written to parse the data produced by the many IGA experiments. The data then needs to be fed into a plotting program, such as xgraph or gnuplot to view the results. The data manager takes care of retrieving and organizing the data from each experiment, and allows the user to rapidly make sense of the vast amounts of data through the plots. This third environment enhancement also saves a significant amount of time and makes easier the work of the researcher.

4.2.4 Future Productivity Improvements

To further increase research productivity, we would like to incorporate in our environment a more intuitive way to define a UI to be evolved. An option would

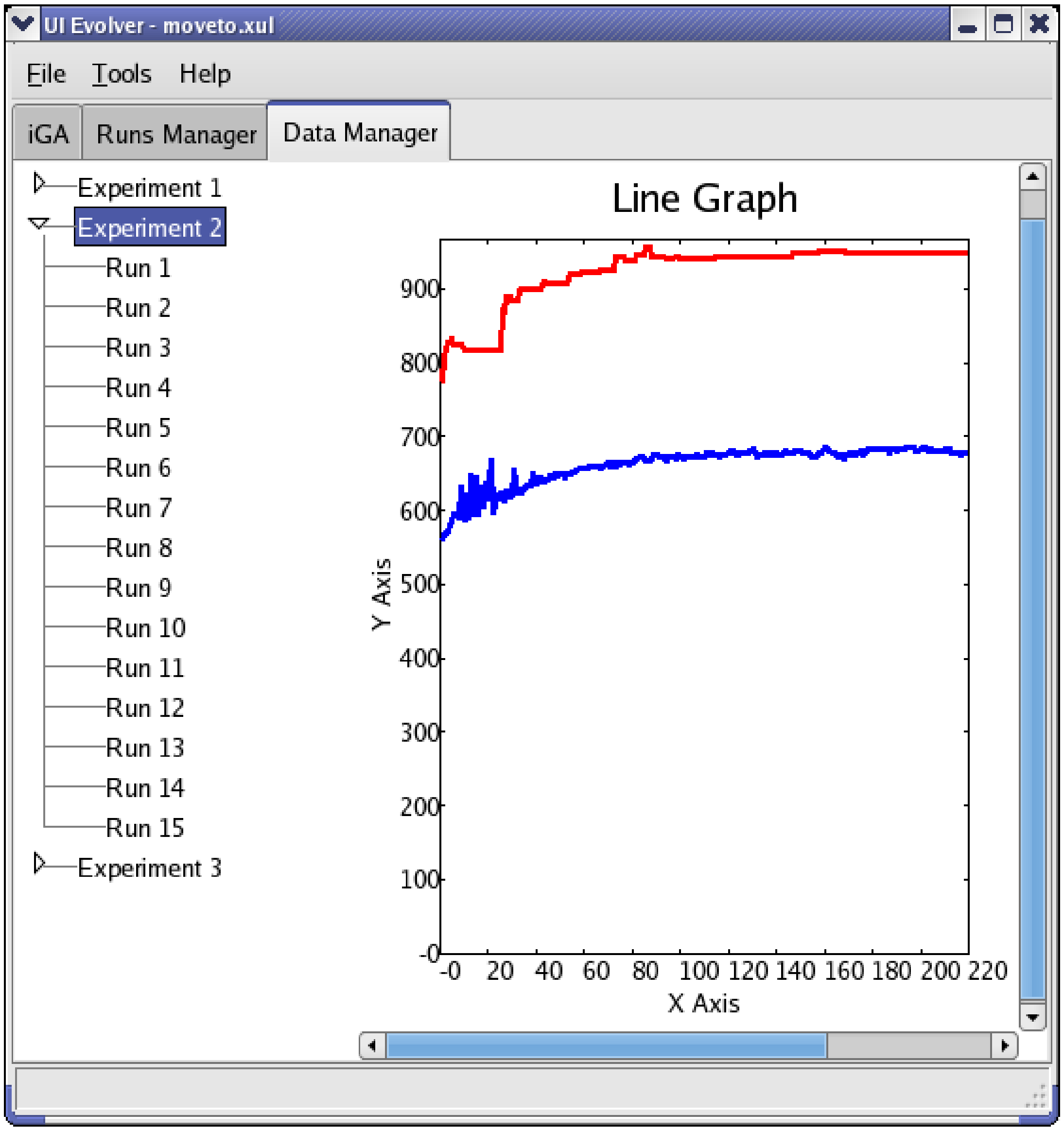
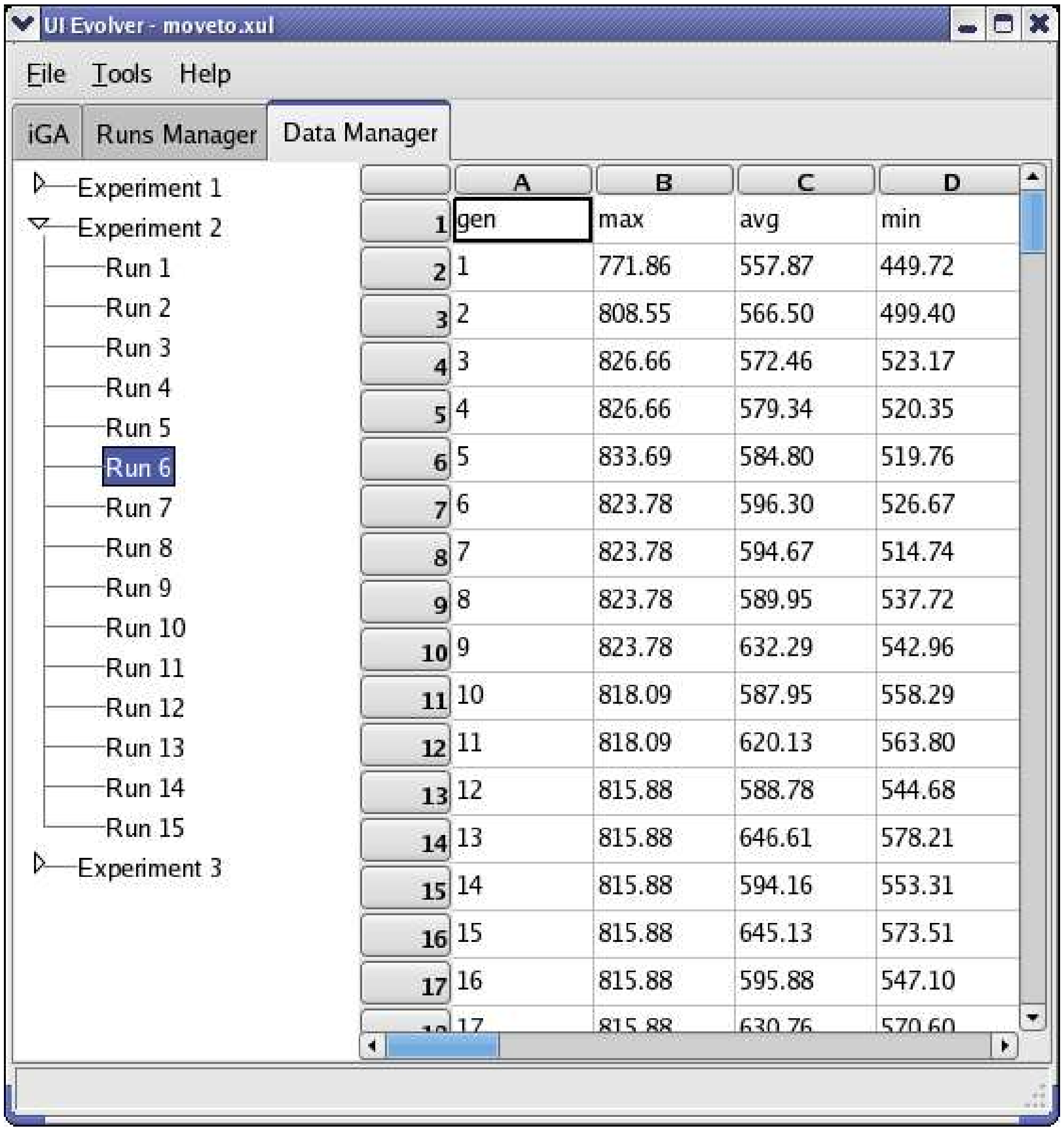


Figure 4.4 The data manager organizes data resulting from experiments into a tree structure. Clicking on an experiment shows a plot of the experiment results.



	A	B	C	D
1	gen	max	avg	min
2	1	771.86	557.87	449.72
3	2	808.55	566.50	499.40
4	3	826.66	572.46	523.17
5	4	826.66	579.34	520.35
6	5	833.69	584.80	519.76
7	6	823.78	596.30	526.67
8	7	823.78	594.67	514.74
9	8	823.78	589.95	537.72
10	9	823.78	632.29	542.96
11	10	818.09	587.95	558.29
12	11	818.09	620.13	563.80
13	12	815.88	588.78	544.68
14	13	815.88	646.61	578.21
15	14	815.88	594.16	553.31
16	15	815.88	645.13	573.51
17	16	815.88	595.88	547.10
18	17	815.88	630.76	570.60

Figure 4.5 The run view of the data manager.

be to have the user define a GUI in a development environment such as wxGlade or NetBeans, and have the representation of it loaded into our environment to be evolved. Another alternative is to have the user define the widgets to be evolved inside the environment, by presenting the user with a list of basic widgets and have the user simply drag and drop widgets into an empty panel. Once the panel was filled with the user desired widgets, then it can be evolved.

We would also like to further abstract the UI specification by allowing the user to specify the type of data that needs to be represented by the UI, and then have our tool evolve both the type of widget used to represent the data and the organization of the widgets.

Code generation of an interface design in the population is also necessary. When a user is satisfied with a design, it would be desirable to generate the code for the selected UI. The user also needs the ability to edit a UI design that is *"good enough"* to the user's preferences, such as moving widgets around, or picking a similar shade of color to the existing color.

4.3 Transitioning from Research Tool to End-User Tool

We would like not only to further improve the productivity of the environment but also to make it available to regular users (non-researchers). In order to conduct user studies and to deploy the system for widespread use, we need to address some usability issues. We foresee having two modes for the environment, an end-user mode and a researcher mode. There are advanced features which a researcher could use, such as the parameters and configuration of the genetic algorithm. However, the end-user (which, in our tool's case is a user interface designer) may not care or understand about such configurations, hence they need to be abstracted.

The researcher mode would allow for configuration of both high level and low level details, giving the researcher the complete control over how the IGA should behave. On the other hand, it does not make sense to present the end-user, a user interface designer, with a cluttered interface and configuration options that are bound to confuse and affect the systems usability and engagement.

The UI designer should be presented with a minimalist interface, with an organization and representation that would be useful for users not familiar with IGAs. This can be accomplished by reducing technical jargon and presenting the user with leverage tools to achieve the desired goals, in this case the exploration of user interface designs. For example, on the context of UI design, it does not make sense to present the user with a slider for *"crossover"* and *"mutation,"* since it does not correlate to the task at hand. A better approach would be to present the user with sliders for *"variety," "creativity,"* or the degree to which the system should *"stick to my choices!"* On the other hand, when designing a tool for researchers we need not shy away from presenting a plethora of configurations.

The end-user tool would basically contain a subset of the functionality presented in the research tool. For example, allowing a user to change the degree of variety in the UIs presented to the user can be done in the background through higher crossover rates and an aggressive selection algorithm. For the advanced user, who wishes to explore how the degree of variety affects the population dynamics, he or she can switch to the advanced mode, and configure low and high level details of the IGA. The sets of features available in researcher mode and, respectively, end-user mode are summarized using use cases in Table 4.3. It can be seen that in our tool's case, with the exception of the *"Extended help"* feature, the end-user mode functionality is a subset of the researcher mode functionality.

For users interested in the use of evolutionary techniques and with a weak programming background, it can be intimidating diving through hundreds of lines of code and customizing a GA to the problem at hand. Through our environment we hope to provide an efficient and usable front-end to both an GAs and IGAs, for the benefit of both the research and end-user communities.

4.4 Related Work

While unique in terms of specific research supported, our software environment can, however, be considered illustrative for two significant challenges faced by scientific researchers. First, how to balance the need for fast research results with the need for better research tools that could improve research productivity in the long run. Second, how to prepare the transition of a tool used for research to a tool accessible by a general category of end users. Nevertheless, because we have started the exploration of literature for reports on the above two topics and found only a few such reports so far, it seems these challenges are yet to be addressed thoroughly [30, 31, 32].

4.5 Summary

We have presented the software environment used for research on evolving user interface designs and described three improvements to the environment aimed at increasing research productivity by automating tedious tasks that had to be conducted previously by the user. Because our research productivity has been significantly increased, we believe that in our environment's case investing effort in developing new features of the research software is beneficial in the long run.

In addition, we have presented a discussion on transitioning the research environment from a researcher's tool to an end-user's tool, and looked into how changes to the current environment could bridge the gap between these two types of tools.

We believe that the two challenges addressed in this chapter, increasing research efficiency via additional tool development and preparing a research tool for transition to an end-user tool, are highly important to researchers and deserve thorough investigation. We believe there is a huge potential for numerous beneficial research and development projects in tackling these challenges.

Table 4.1 Use Cases in End-User and Researcher Modes

	Use Case	End-User Mode	Researcher Mode
1	Define user interface	x	x
2	Load user interface definition		x
3	Customize high level IGA details	x	x
4	Customize low level IGA details		x
5	Start IGA	x	x
6	Stop IGA	x	x
7	Open IGA state	x	x
8	Save IGA state	x	x
9	Save IGA state	x	x
10	Select best and worst UI	x	x
11	Undo evolution step	x	x
12	Redo evolution step	x	x
13	Edit evolved UI	x	x
14	Run batch mode		x
15	Extended help	x	

Chapter 5
User Interface Evolution with a Simulated User

Who from the IGA population do we display for user evaluation? How does our selection of who we display to the user affect the population dynamics and IGA performance? Should we ask for user input every generation? Can we instead ask for user input every 2 generations in order to reduce user fatigue? How does less user input affect the IGA performance? We address these questions by conducting experiments using a simulated user. The simulated user gave us the leverage to conduct the first set of experiments and to test our approach with a hypothetical tireless user. The next chapter will discuss our evolutionary approach with real users.

5.1 Experimental Setup

We conducted two experiments; the first to investigate which individuals to display for user evaluation and the second to investigate how often we need to ask for user input. All results reported below are averages from 30 independent runs of the IGA.

Instead of using real people we used a simulated human with a preference for the color blue. The simulated human gave us the leverage to have a tireless user do our preliminary tests and experiments. Given a set of UIs displayed for user evaluation, we used a greedy approach to simulate user picking, and the UI with the most blue widgets was chosen as the best, and the UI with the least blue widgets was chosen as the worst.

We chose to test three methods for selecting our $n = 10$ individuals that make up the subset displayed for user evaluation. The first method displayed the best n individuals in the population. The second method displayed both the best $n/2$ and the worst $n/2$ individuals in the population. The last method randomly selected n individuals in the population to be displayed for user evaluation.

For the experiments conducted we used a population size of 100 and we displayed 10 individuals for user evaluation. We compare two selection methods, roulette wheel selection and probabilistic tournament selection. For tournament selection we used a tournament size of 4, with 90% probability of choosing the best individual in the tournament. Four individuals from the population are randomly sampled to form a tournament for parent selection. We used 80% crossover rate and 1% mutation rate.

To test how the frequency of user input affects IGA performance we conducted experiments asking for user input every t^{th} generation. We used t values of $1, 5, 10, 20, 40$, and 80. We keep a reference to the user selected best and worst UIs, so that we can do our interpolation technique even when we use values of t greater than 1.

5.2 Results

As expected, we found that using tournament selection with a tournament size of 4 outperformed roulette wheel selection (see figure 5.1). The figure shows the best individuals in the population. Tournament selection's stronger selection pressure leads to much quicker convergence to better values.

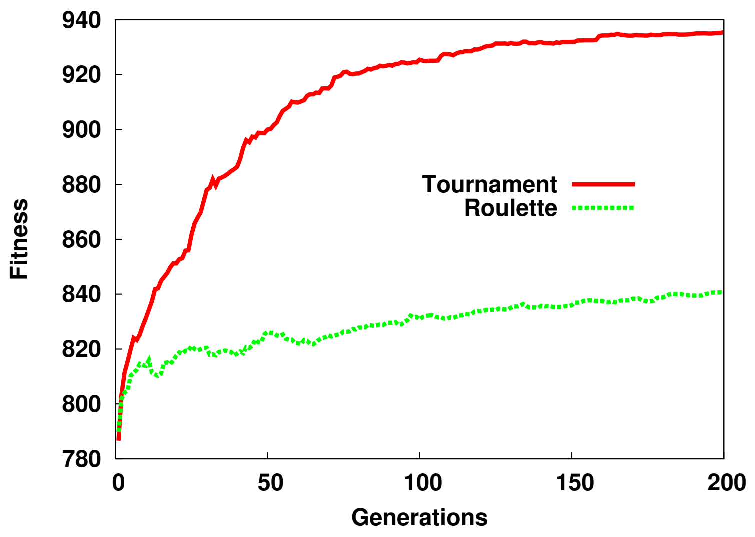


Figure 5.1 Tournament selection versus roulette wheel selection. The plot shows the best individuals in the population.

5.2.1 Subset Display Method

We compared three methods of selecting individuals to be displayed to the user. The three methods are displaying the best n individuals, displaying n random individuals, and displaying the best $n/2$ and the worst $n/2$ individuals in the population. Displaying the best individuals in the population gives the user the opportunity to view individuals that show the greatest potential by both meeting the objective and subjective heuristics most effectively. Displaying random individuals gives the user an unbiased insight into the current state of the population; it can allow the user to see the degree to which the population is converging (by the number of individuals that are similar), but it suffers because it can present bad UI designs to the user.

Lastly, displaying both best and worst individuals allows the user to see what the population is converging to and where it is coming from.

We ran the IGA with each of the three display methods using tournament selection and plotted the fitness of the best individuals in the population as shown in figure 5.2. We can see that displaying the best individuals in the population for user evaluation results in the best IGA performance when compared to displaying random individuals and displaying both the best and the worst individuals in the population. Figure 5.3 also shows that our (simulated) user is able to bias IGA performance effectively by displaying the best individuals in the population for subjective evaluation. Remember, the simulated user preferred blue widgets. Displaying the best and worst individuals in the population results in individuals with blue widgets, but which violate the style guideline metrics that we are trying to enforce through the objective evaluation.

5.2.2 The Power of t

We varied the value of t to explore the effects of user input every t^{th} generation on IGA performance. That is, the user was only asked to make a choice once every t generations and we used that choice for the next t generations to interpolate individuals' fitness. Figure 5.4 compares convergence behavior for $t = 1, 5, 10$, and 20, where we have plotted the average fitness over 30 runs of the best individuals in the population. We were encouraged to see that varying t, for small values of t, has little

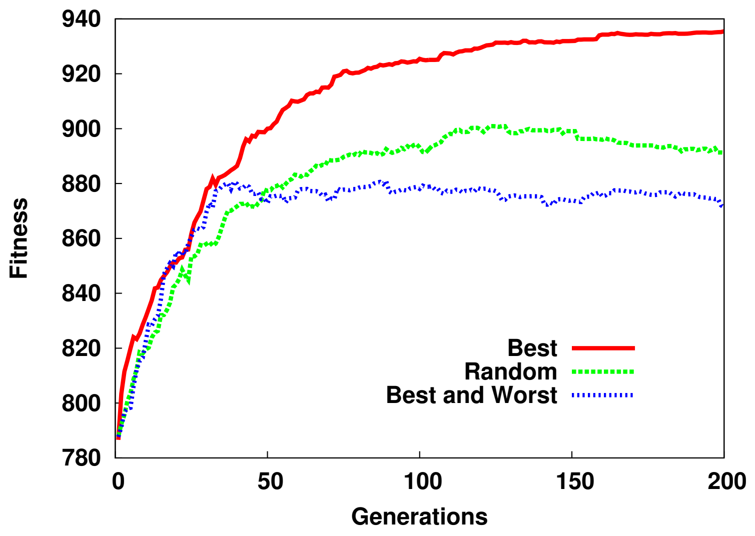


Figure 5.2 Subset display method comparison.

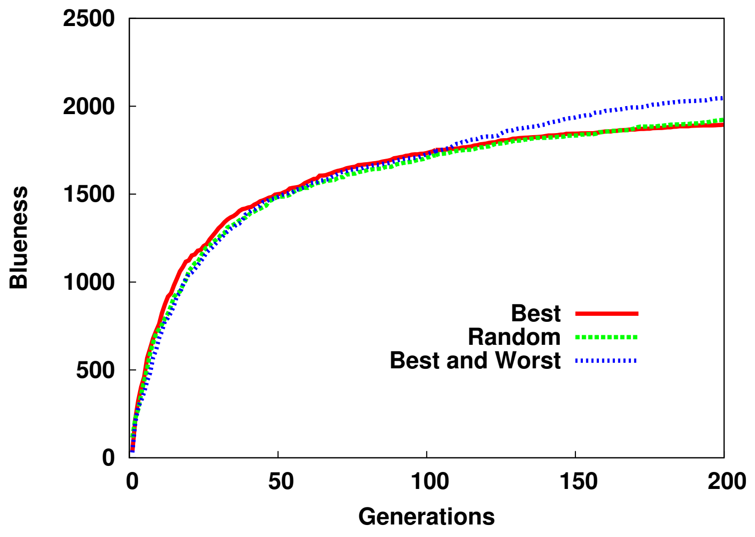


Figure 5.3 Subset display method comparison on convergence to blue widgets.

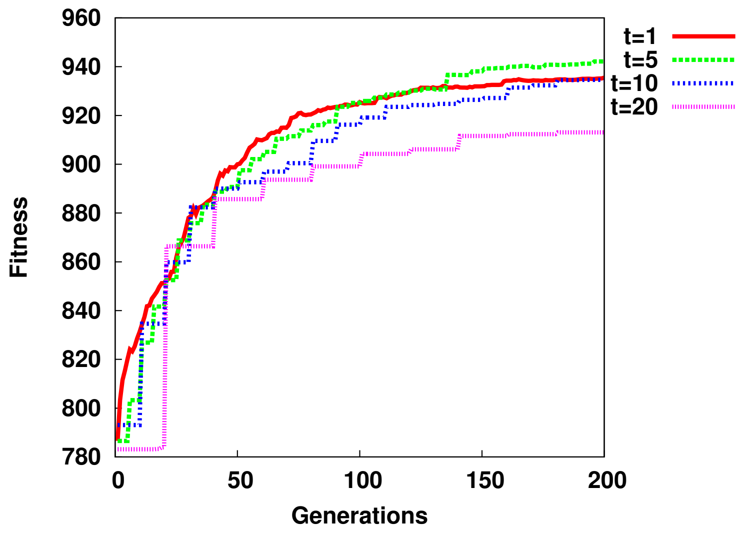


Figure 5.4 Effect of varying t on IGA performance.

effect on the IGA's convergence behavior. Next, to look at the effect of changing t on the subjective fitness, we plotted the convergence to blue widgets in figure 5.5 (again this is average of the best individuals). Note that even a small change in t results in a drop in convergence to blue UIs as shown in the figure. With less frequent user input we get increasingly noisy subjective fitness evaluation.

We increased the value of t to 20, 40, and 80 generations to assess the effect on IGA performance. Figure 5.6 shows the fitness plot of the best individuals in the population. We can see the step-like increase of fitness corresponding to the generation when our user makes a selection. Figure 5.7 shows the fitness plot of the average individuals in the population. The sharp decrease in fitness in early generations corresponds to the generation in which the user makes the second picking, since the first user picking is done upon population initialization. We then see a slow

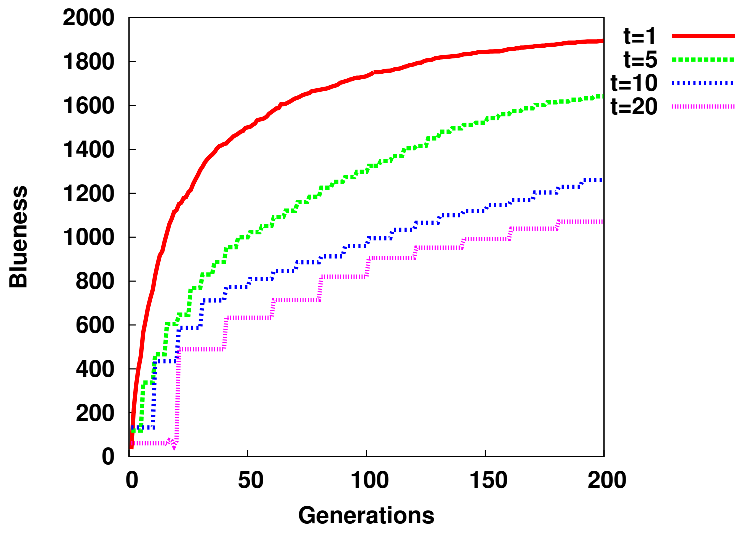


Figure 5.5 Effect of varying t on convergence to blue UIs.

increase in fitness.

We also plotted the convergence to blue UIs, which was the user assumed preference. Figure 5.8 shows the *"blueness"* of the best individuals in the population. From the figure we see that increasing values of t leads to decreasingly blue UIs. Thus, as expected, less user input results in a less effective subjective bias on the population. Finally, figure 5.9 shows the average blueness of individuals in the population indicating that the average performance correlates well with best performance.

5.2.3 User Interfaces Generated

Figures 5.10 and 5.11 show a subset consisting of the 10 best individuals in the population at generations 0 and 200, respectively. In generation 0, widgets start with random positions and random colors. In generation 200, the UIs shown all have blue widgets, which was the user assumed preference. The UIs at generation 200 both

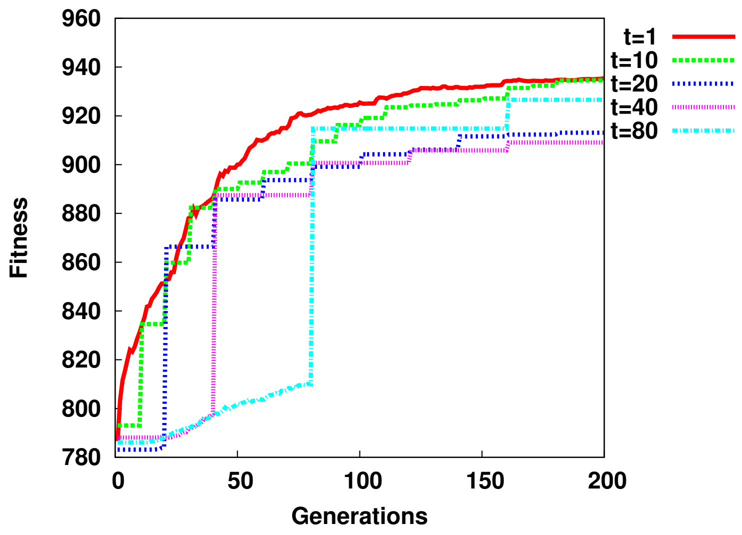


Figure 5.6 Degradation on the IGA performance (maximum) for high t values.

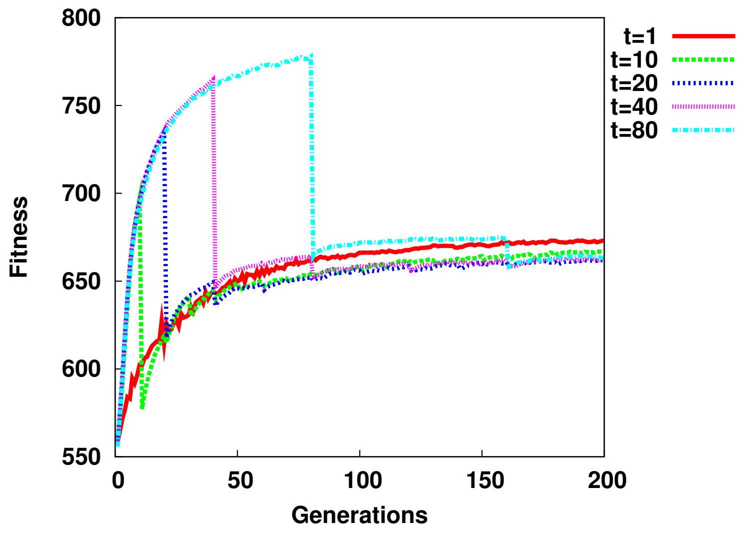


Figure 5.7 Degradation on the IGA performance (average) when using high t values.

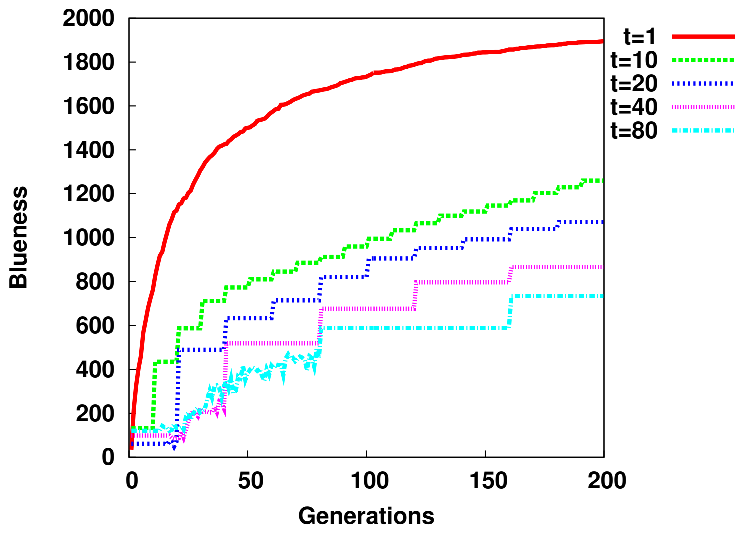


Figure 5.8 Degradation in the convergence (maximum) to blue UIs when using high t values.

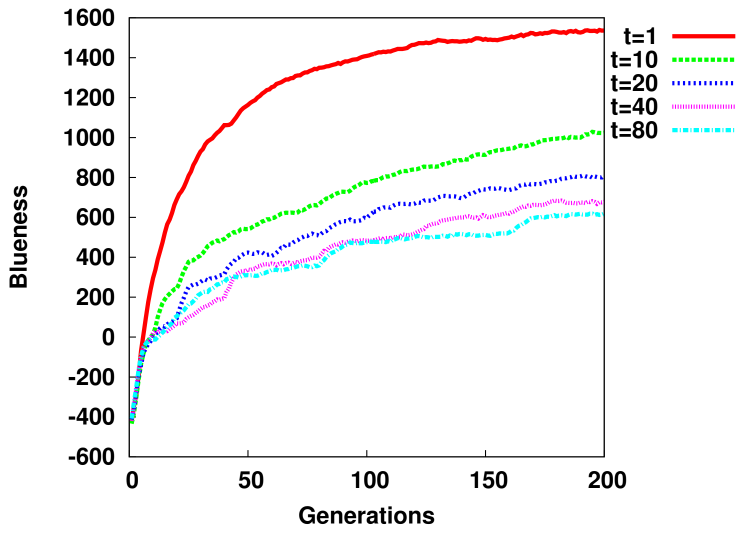


Figure 5.9 Degradation in the convergence (average) to blue UIs when using high t values.

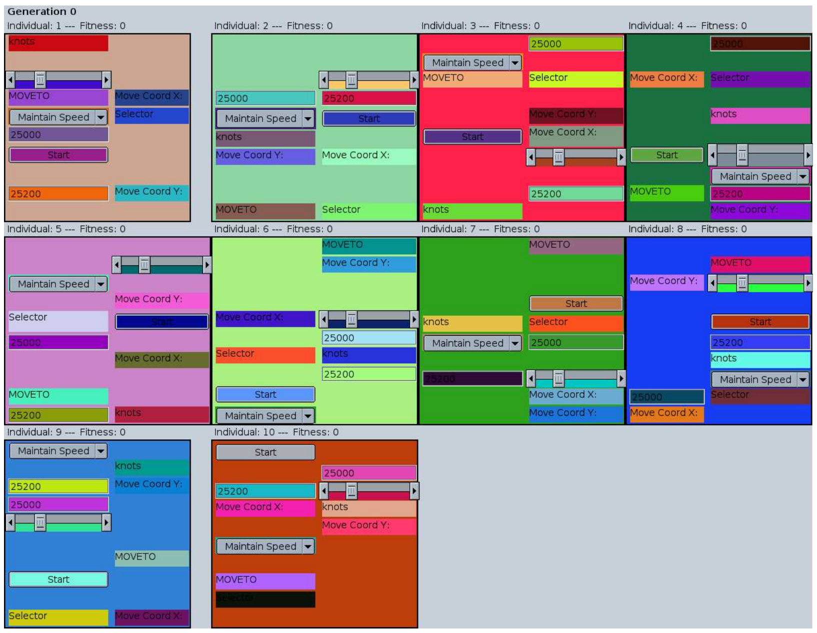

Generation 0
Individual: 1 --- Fitness: 0
Individual: 2 --- Fitness: 0
Individual: 3 --- Fitness: 0
Individual: 4 --- Fitness: 0
Individual: 5 --- Fitness: 0
Individual: 6 --- Fitness: 0
Individual: 7 --- Fitness: 0
Individual: 8 --- Fitness: 0
Individual: 9 --- Fitness: 0
Individual: 10 --- Fitness: 0
knots
MOVETO
Move Coord X:
Move Coord Y:
Maintain Speed
Selector
Start
25000
25200

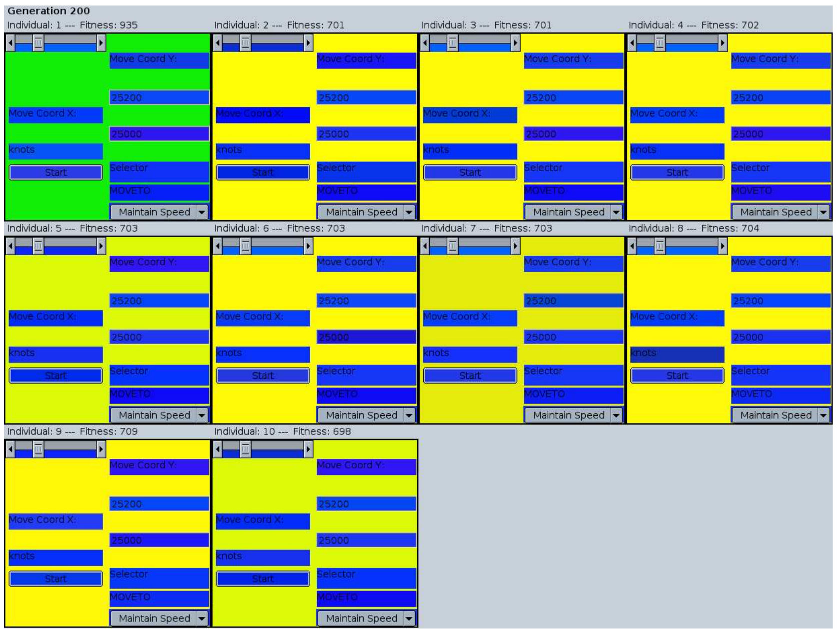
Figure 5.10 Displaying the best 10 individuals for user evaluation at generation 0.

respect the metrics enforced on the objective evaluation: 1) Widgets should all have a similar shade of color, and 2) There should be a high contrast between foreground and background colors.

5.3 Summary

The results presented in this chapter were published in [33, 34]. We have explored methods by which to reduce user fatigue in IGAs by 1) displaying a subset from the population that yields the best IGA performance, 2) asking the user to select the best and worst UIs from the subset displayed for user evaluation, and 3) assessing the effects of limiting user input by having the user pick every t generations.

We first compared selection methods in order to find the one that yielded the best


Generation 200

Individual: 1 --- Fitness: 935
Individual: 2 --- Fitness: 701
Individual: 3 --- Fitness: 701
Individual: 4 --- Fitness: 702
Individual: 5 --- Fitness: 703
Individual: 6 --- Fitness: 703
Individual: 7 --- Fitness: 703
Individual: 8 --- Fitness: 704
Individual: 9 --- Fitness: 709
Individual: 10 --- Fitness: 698

Figure 5.11 Displaying the best 10 individuals for user evaluation at generation 200.

IGA performance. We found tournament selection's high selective pressure to yield better and more robust IGA results. Next we compared three subset display methods, where each method resulted in a different composition of the subset displayed for user evaluation. Our results indicate that displaying the best individuals for user evaluation results in better and faster convergence of the population, when compared to displaying a random subset and a subset consisting of the best and the worst individuals in the population. We also found that we can get reasonable convergence to well laid out blue (our preferred color) interfaces for small values of t. High values of t can reduce human fatigue considerably, but at the cost of increased noise in the subjective fitness landscape.

Chapter 6
User Interface Evolution with Real Users

We have assessed how the use of a simulated user affects IGA performance. While the simulated user allowed us to extensively test the IGA and to conduct the first set of experiments, the simulated user is far from an accurate model of a user. The simulated user always chooses the UI with the most blue widgets as the best and the least blue widgets as the worst. A user is likely to change his/her mind often, and since there is no way to enforce consistency from the user responses, the selection of who is the best can be anything from a good color combination to a good widget layout. In this chapter we discuss the experiments conducted with actual users, and how both our interpolation technique and asking for user input every t^{th} generation effectively reduces user fatigue.

6.1 Experimental Setup

Three users participated in five IGA sessions, each session lasting 30 generations. For these five sessions, we asked the user to make a selection every t generations, with t values of 1, 3, 5, 10, and 15, allowing the user to bias the evolution of the UIs 30, 10, 6, 3, and 2 times respectively. We instructed the users to choose the UI they liked the best and the UI they liked the least, based on whatever criteria they desired. We keep a reference to the user selected best and worst UIs, so that we can do our interpolation technique even when we use values of t greater than 1.

Our IGA's parameter settings for the experiments conducted with real users are as follows: (1) population size of 100, (2) 9 individuals displayed for user evaluation, and (3) probabilistic tournament selection with a tournament size of 4, 90 percent probability of choosing the tournament best individual (otherwise we choose a random individual from the tournament losers). For the experiments conducted with the simulated user, discussed in chapter 5, we displayed 10 individuals for user evaluation. Here we display 9 individuals because this allows us to aesthetically display 3 individuals per row, instead of having two rows of 4 individuals with a third row with only two individuals.

6.2 Results

We plotted the fitness convergence for our three study subjects: $user1$, $user2$, and $user3$. Figure 6.1 shows user1's session fitness convergence of the best individuals in the population for $t = 1, 3, 5, 10$, and 15. We can see step like increases for $t = 1$ and $t = 3$ as the user varies their selection of the UI they like the best. Sharp increases in fitness reflect the user choosing an individual that also conforms to the objective metrics. Note that in our IGA, the population will constantly evolve towards UIs that reflect the objective design metrics, hence the fitness increases over time. Through the generations the user sees individuals that increasingly reflect conformance to the objective metrics, yet which resemble individuals the user liked. The fitness increase shows the successful fusion of computable objective metrics and user subjective input guiding the evolution of the UIs. Lastly, notice that for a value of $t = 3$ user1 is able to achieve a higher fitness than with $t = 1$. We did not expect this behavior since our previous results with a simulated user showed that giving the simulated user complete control over the UI evolution by allowing them to participate in every generation resulted in the highest fitness performance. Also, we noticed that the maximum fitness for values of $t = 5, 10$, and 15 remain constant. We attribute this behavior to user1 not changing their selection of the best UI during the entire session. With low values of t a user has more opportunities to change the selection of the best UIs, (30 chances with $t = 1$ and 10 chances with $t = 3$).

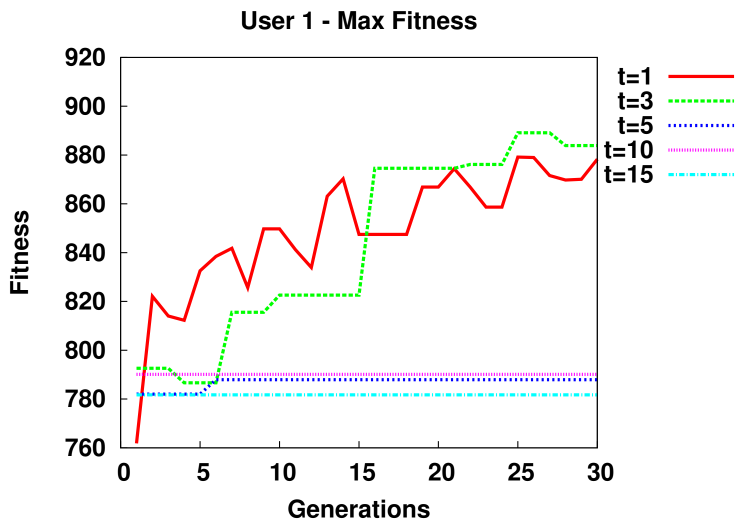


Figure 6.1 Fitness performance of user 2. The plot shows the best individuals in the population.

Figure 6.2 shows user2's session fitness convergence of the best individuals for the same values of t. We see that for user2, $t = 10$ achieved the highest fitness, and for $t = 15$ user2 did not change their selection of the best UI during the entire session. User2 was also able to successfully bias the evolution of the UIs by fusing objective and subjective criteria.

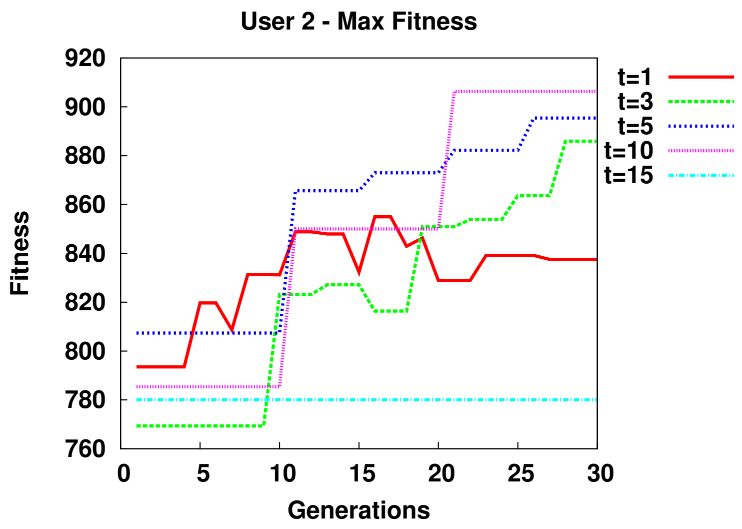


Figure 6.2 Fitness performance of user 2. The plot shows the best individuals in the population.

Figure 6.3 shows the fitness plot for user3. User3 presents interesting results, since his/her varied selections of the best UI helped in finding high fitness values for all t. Notice that for all three users using a value of $t = 1$ did not result in the highest fitness convergence. Figures 6.4, 6.5, and 6.6 show the fitness plot of the population average for the three users. The steep drops in average fitness performance correspond to the time steps where the user makes a selection of the best and worst UIs. These average fitness performance results are similar to our previous results with a simulated user.

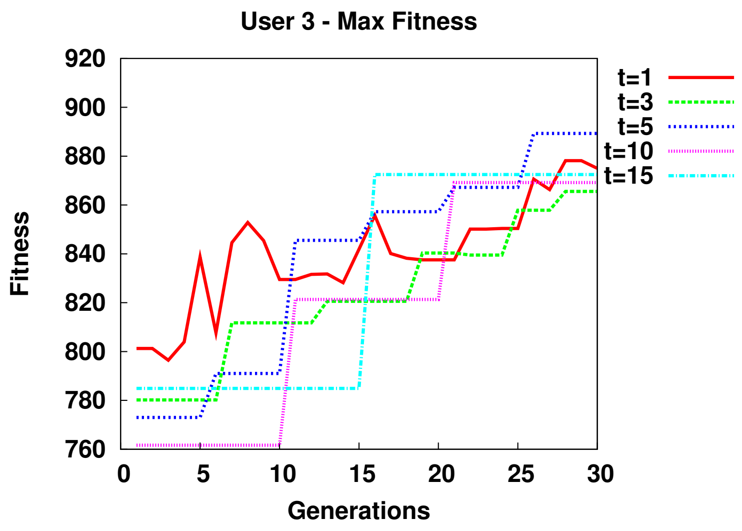


Figure 6.3 Fitness performance of user 3. The plot shows the best individuals in the population.

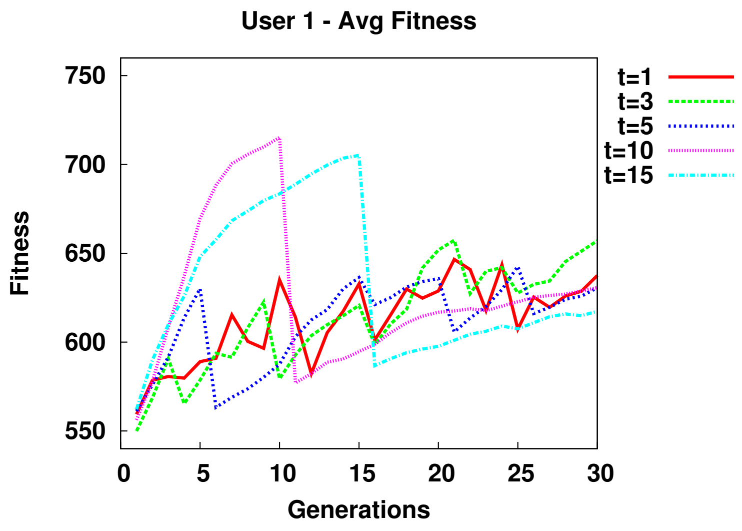


Figure 6.4 Fitness performance of user 1. The plot shows the average individuals in the population.

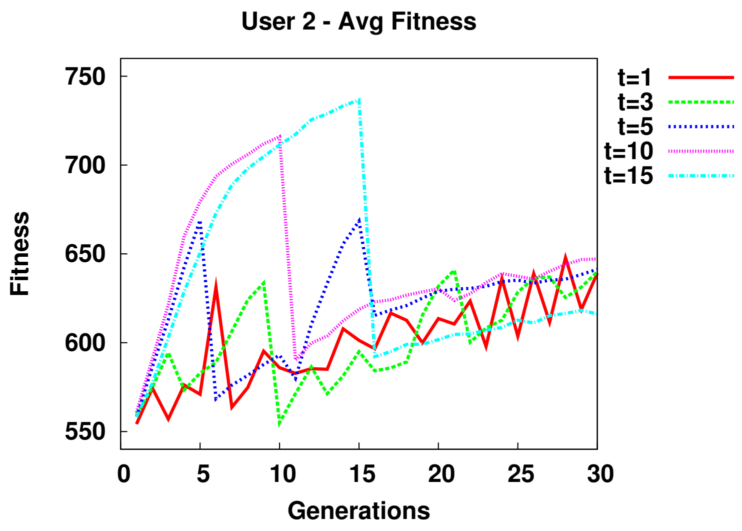


Figure 6.5 Fitness performance of user 2. The plot shows the average individuals in the population.

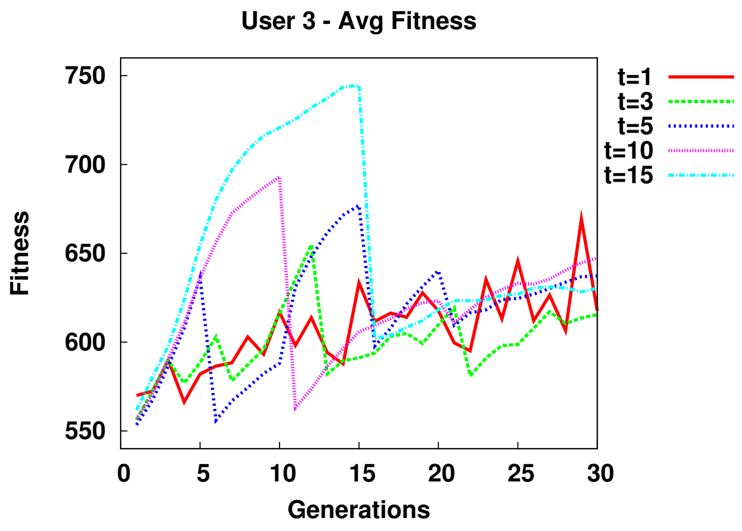


Figure 6.6 Fitness performance of user 3. The plot shows the average individuals in the population.

Why do we see a drop in average fitness performance associated with the time steps on which the user provides input? Initially we expected to see a drop in average performance associated with the user changing the selection of the best and worst UIs. However, we can see a drop in performance even when the user does not change his/her selection of the best UI through the entire session as was seen with user1 (Figure 6.1) and user2 (Figure 6.2). Why do we see a constant best individual with high values of t and not with low values of t? We conducted an experiment to test two hypothesis: (1) the drop in average performance associated with user input is due to the user changing the selection of the worst UI while the best UI remains constant, and (2) the constant best individual results in a flat maximum fitness and is common with high values of t because of the reduced user intervention.

The drop in fitness performance associated with user input can be a result of a user changing the least-preferred UI while not changing the best-preferred UI. We conducted another session run with a user, where the user was instructed to pick a UI as the best at the beginning of the session and to continually pick that UI throughout the rest of the run. We had the user do this on two sessions, where in one of the sessions we turned off the comparison to the user selected worst UI. Finally, we used $t = 3$ - asking for user input every 3 generations, since none of the users picked the same UI as the best for $t = 1$ and $t = 3$. Thus, we wanted to confirm the conjecture that such behavior was less common with low values of t since the user has more opportunities to change his/her selection.

Figure 6.7 shows these results. The plot shows the fitness convergence of the best

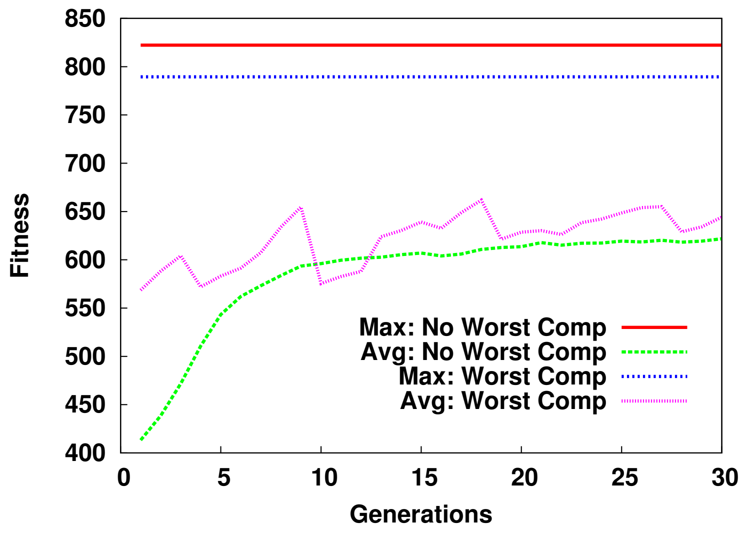


Figure 6.7 Fitness performance of comparing to user selected worst and without the comparison.

and average individuals in the population with and without comparison to the UI the user liked the least. We can see that having the user pick the same individual as the best UI at every time step results in a constant maximum fitness as we saw in Figures 6.1 and 6.2 for user1 and user2. Notice that comparing individuals in the population to the UI the user likes the least results in steep drops in fitness performance associated with the time step (every 3 generations) in which the user makes a selection. We also see from the plot that removing the comparison to the user selected worst individual results in a monotonic increase in fitness performance. This supports our hypothesis that the comparison to the UI the user likes the least accounts for the sharp fitness drops, even when the user selected best UI remains constant. It also supports the conjecture that with low values of t the user has more opportunities to change the selection of the best UI.

6.2.1 User Experience

Doing all 5 IGA runs (for values of $t = 1, 3, 5, 10$, and 15) took about 30 minutes to complete, with the session using a value of $t = 1$ (user input every generation for a maximum of 30 generations), taking over half the time (20 minutes) to complete. We found that using a value of $t = 1$ results in slow changes from generation to generation, forcing the user to pay more attention to detail and making the session more strenuous. One of the users commented that using high values of t usually converged to likable UI colors, without having to spend a lot of time making a selection every generation. Even though 30 generations is not a big number, having to make a selection every generation still results in user fatigue. Higher values of t seem to significantly reduce user fatigue and lessen the time spent on each session.

6.2.2 User Interfaces Generated

Figures 6.8 and 6.9 show a subset consisting of the 9 best individuals in the population at generations 0 and 30 respectively for user3. The figures were taken during the session using $t = 15$. In generation 0, widgets start with random positions and random colors. In generation 30, we can see the best UIs which reflect both the

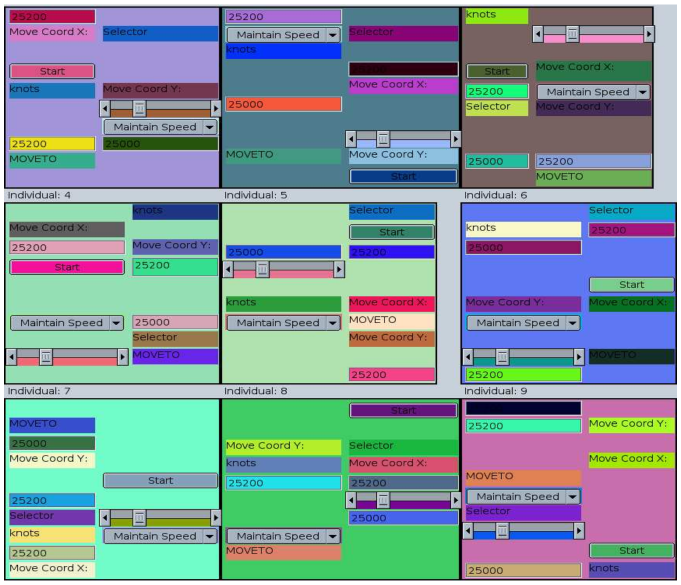


Figure 6.8 The best nine individuals in the initial population.

user3's preferences and which best follow coded guideline metrics.

6.3 Summary

We investigated three methods to reduce user fatigue in IGAs by 1) displaying a subset of the best nine individuals from the population, 2) asking the user to select the best and worst UIs from the subset displayed for user evaluation, and 3) assessing the effects of limiting user input by having the user pick every t generations.

We had three users evolve UIs with our tool to explore how often to ask for user input. High values of t can reduce human fatigue and reduce the time spent by the user on a session. Through our interpolation technique we were able to reduce the number of selections to two every generation. We conducted an additional experiment to test the hypothesis that our comparison to the user selected worst results in steep

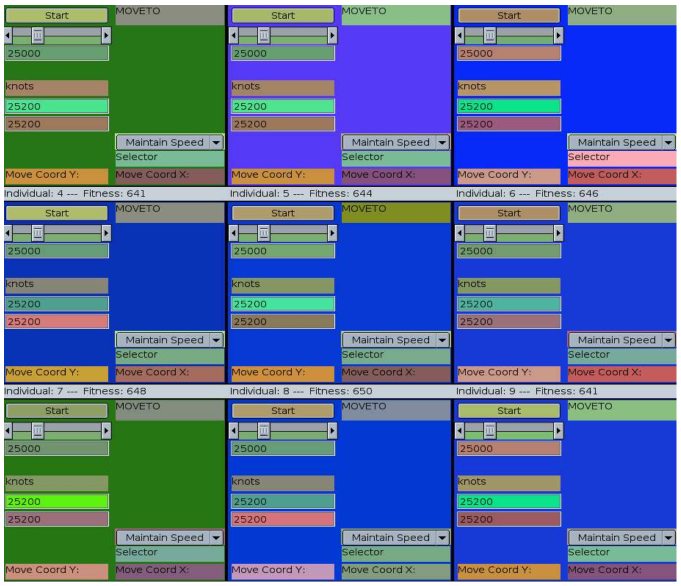

Start
MOVETO
25000
knots
25200
25200
Maintain Speed
Selector
Move Coord Y:
Move Coord X:
Individual: 4 --- Fitness: 641
Individual: 5 --- Fitness: 644
Individual: 6 --- Fitness: 646
Individual: 7 --- Fitness: 648
Individual: 8 --- Fitness: 650
Individual: 9 --- Fitness: 641

Figure 6.9 The best nine individuals at session end.

drops in average fitness performance on the time steps where the user provides input. Sessions conducted with the comparison to the user selected worst turned off show a smooth, monotonic, increasing average fitness. On the other hand, turning on the comparison to the user selected worst results in steady increases coupled with sharp drops in average fitness when the user provides feedback. Picking the same the UI through the entire session results in a constant maximum fitness, yet it is the user's changing selection of the worst UI that leads to the sharp drops in average fitness performance. In the future we would like to explore whether asking for the selection of both the best and the worst is necessary, and to conduct experiments with users picking only the best UI or only the worst UI from a small subset displayed for evaluation.

Chapter 7
Conclusions and Future Work

We presented an IGA that combines both computable metrics, taken from style guidelines, and human subjective input to guide the evolution of UIs. The design process is driven by both formalized style guidelines and by a human sense of aesthetics, making our approach a suitable and promising application that can change the way UIs are designed and maintained.

Within this context, we investigated three methods to reduce user fatigue in IGAs by 1) displaying a subset of the best nine individuals from the population, 2) asking the user to select the best and worst UIs from the subset displayed for user evaluation, and 3) assessing the effects of limiting user input by having the user pick every t generations.

We first compared selection methods in order to find the one that yielded the best IGA performance. We found tournament selection's high selective pressure to yield better and more robust IGA results. Next we compared three subset display methods, where each method resulted in a different composition of the subset displayed for user evaluation. Our results indicate that displaying the best individuals for user evaluation results in better and faster convergence of the population, when compared to displaying a random subset and a subset consisting of the best and the worst individuals in the population. We also found that we can get reasonable convergence to well laid out blue (our preferred color) interfaces for small values of t. High values of t can reduce human fatigue considerably, but at the cost of increased noise in the subjective fitness landscape. Through our interpolation technique we were able to reduce the number of selections to two every generation.

We had three users evolve UIs with our tool to assess how the IGA behavior differed from the simulated user and to explore how often to ask for user input. We found that high values of t can reduce human fatigue and reduce the time spent by the user on a session, while lower values of t increase the mental demand on the user due to the greater need to pay attention to detail.

We believe that the work presented in this paper lays a good foundation for future research and development. We would like to conduct further experiments by varying the value of t over the course of a single run, exploring how asking for user input often early in an IGA session (when there is a high degree of diversity in the population), and asking for less user input in later generations (when the population approaches convergence) affects performance. Further, we would like to expand widget encoding to support coupling between widgets and high level spatial relationships with other widgets and the parent panel.

We plan to incorporate more heuristics from the various style guidelines into the objective evaluation component. Further user studies need to be conducted, with a larger sample, to evaluate the utility of the tool and the effectiveness with which users can guide and bias the evolution of UIs. Finally, we intend to investigate using the longest common subsequence metric as our similarity measure for the layout chromosome.

Last, we plan to further explore representations and genetic operators that can yield higher fitness individuals in less generations and with higher confidence intervals. A representation which yields smooth gradients during crossover and mutation is also needed. A color model that better correlates to how we define similarity among colors can help.

In terms of future work for our research environment, we intend to initiate a quantitative assessment of the increase in productivity brought by the improvements integrated in our environment (we will collect long term data for this purpose). We also intend to develop an application programming interface (API) and generalize our IGA tool such that it could be used by the AI research community as a front-end to genetic algorithms. Lastly, we would like to elaborate a set of guidelines for researchers to be followed when preparing and distributing tools that can be used by the end-user community.

The long-term goal of this evolutionary approach to UI design is to streamline and help reduce the complexity associated with the generation and the fine-tuning of UIs. We believe that the research reported here shows the viability of an interactive evolutionary approach to UI design.

"So it goes." - Kurt Vonnegut, Jr.

References

1. Apple: Apple human interface design guidelines: Introduction to apple human interface guidelines (2006)
2. Microsoft Corporation: Windows xp - guidelines for applications (2006)
3. Sun Microsystems: Java look and feel design guidelines (2001)
4. GNOME: Gnome human interface guidelines 2.0 (2004)
5. Ngo, D.C.L., Teo, L.S., Byrne, J.G.: Modelling interface aesthetics. Inf. Sci. **152** (2003) 25–46
6. Thimbleby, H.: User interface design with matrix algebra. ACM Trans. Comput.-Hum. Interact. **11** (2004) 181–236
7. Llorà, X., Sastry, K., Goldberg, D.E., Gupta, A., Lakshmi, L.: Combating user fatigue in igas: partial ordering, support vector machines, and synthetic fitness. In: GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, New York, NY, USA, ACM Press (2005) 1363–1370
8. Takagi, H.: Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. Proceedings of the IEEE **89** (2001) 1275–1296 Invited Paper.
9. Preece, J., Rogers, Y., Sharp, H.: Interaction Design: Beyond Human Computer Interaction. Wiley (2002)
10. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
11. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
12. Goldberg, D., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. **1** (1991) 69–93
13. Blickle, T., Thiele, L.: A comparison of selection schemes used in genetic algorithms. Technical Report 11, Gloriastrasse 35, 8092 Zurich, Switzerland (1995)
14. Sokolov, A., Whitley, D.: Unbiased tournament selection. In: GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, New York, NY, USA, ACM Press (2005) 1131–1138

15. Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F.: Computer graphics: principles and practice (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1990)

16. Kim, W.C., Foley, J.D.: Providing high-level control and expert assistance in the user interface presentation design. In: CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM Press (1993) 430–437

17. Ivory, M.Y., Hearst, M.A.: The state of the art in automating usability evaluation of user interfaces. ACM Computing Surveys **33** (2001) 470–516

18. Oliver, A., Monmarché, N., Venturini, G.: Interactive design of web sites with a genetic algorithm. In: Proceedings of the IADIS International Conference WWW/Internet, Lisbon, Portugal (2002) 355–362

19. Monmarché, N., Nocent, G., Slimane, M., Venturini, G., Santini, P.: Imagine: a tool for generating html style sheets with an interactive genetic algorithm based on genes frequencies. In: Proceedings of the International Conference on Systems, Man, and Cybernetics, IEEE Computer Society (1999) 640–645

20. Cho, S.B.: Towards creative evolutionary systems with interactive genetic algorithm. Applied Intelligence **16** (2002) 129–138

21. Deb, K., Kalyanmoy, D.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons, Inc., New York, NY, USA (2001)

22. Gunn, S.: Support vector machines for classification and regression (1998)

23. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Discov. **2** (1998) 121–167

24. Bennett, K.P., Campbell, C.: Support vector machines: hype or hallelujah? SIGKDD Explor. Newsl. **2** (2000) 1–13

25. Kamalian, R., Zhang, Y., Takagi, H., Agogino, A.: Reduced human fatigue interactive evolutionary computation for micromachine design. In: Proceedings of the 2005 International Conference on Machine Learning and Cybernetics. Volume 9., IEEE Computer Society (2005) 5666–5671

26. XULPlanet: Xulplanet.com (2006)

27. Douglas, S.A., Kirkpatrick, A.E.: Model and representation: the effect of visual feedback on human performance in a color picker interface. ACM Trans. Graph. **18** (1999) 96–127

28. Wu, Y., Takatsuka, M.: Three dimensional colour pickers. In: APVis '05: proceedings of the 2005 Asia-Pacific symposium on Information visualisation, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2005) 107–114

29. ECSL: Lagoon (2006)

30. Ryan, C., Tewey, B., Newman, S., Turner, T., Jaeger, R.J.: Estimating research productivity and quality in assistive technology: a bibliometric analysis spanning four decades. IEEE Transactions on [see also IEEE Trans. on Rehabilitation Engineering] Neural Systems and Rehabilitation Engineering **12** (2004) 422–429

31. Using information technology to leverage research productivity. In: Engineering and Technology Management, 1996. IEMC 96. Proceedings., International Conference on, Vancouver, BC (1996)

32. Yao, J.T., Yao, Y.Y.: Web-based information retrieval support systems: building research tools for scientists in the new information age. In: Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on. (2003) 570–573

33. Quiroz, J.C., Dascalu, S.M., Louis, S.J.: Human guided evolution of xul user interfaces. In: CHI '07: CHI '07 extended abstracts on Human factors in computing systems, New York, NY, USA, ACM Press (2007)

34. Quiroz, J.C., Louis, S.J., Dascalu, S.M.: Interactive evolution of xul user interfaces. In: GECCO '07: Proceedings of the 2007 conference on Genetic and evolutionary computation, New York, NY, USA, ACM Press (2007)