Fitness Interpolation in Interactive Genetic Algorithms

Juan C. Quiroz, Sushil J. Louis, *Member, IEEE,* Amit Banerjee, and Sergiu M. Dascalu *Member, IEEE,*

***Abstract*—We attack the problem of user fatigue in using an interactive genetic algorithm to evolve two case studies: user interfaces and floorplans. We show that we can reduce human fatigue in interactive genetic algorithms (the number of choices needing to be made by the user), by 1) only asking the user to evaluate a small subset from a large population size, and 2) by asking the user to make the choice once every t generations. We present an analysis of the user actions and results obtained. Finally, we introduce the concept of collaborative interactive genetic algorithms and its effects on the exploration of solutions in the search space.**

***Index Terms*—Interactive genetic algorithm, user fatigue**

I. Introduction

INTERACTIVE genetic algorithms (IGAs) differ from genetic algorithms (GAs) in that the objective fitness evaluation is replaced with user evaluation, thus allowing for the user to guide an explorative evolutionary process when there is no better fitness measure than the one in the human mind [16]. As such, they can incorporate intuition, emotion, and domain knowledge from the user. 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 [13]. 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 [16].

We address the issue of user fatigue by the use of the following techniques: (1) asking the user to evaluate a small subset while maintaining a large population size; (2) asking for user input every t^{th} generation; and (3) introducing collaboration between concurrent individual interactive evolutionary sessions. The first and second techniques are showcased with the case study of evolving a simple layout of buttons and corresponding color schemes for a simple user interface (UI). We present collaborative interactive genetic algorithms with the second case study of floorplan design, as an extension of the simple UI design case study. During collaboration users participating in interactive evolutionary sessions can see each others' designs, and can inject a solution from any of their peers at any time during the evolutionary session.

Juan C. Quiroz, Sushil J. Louis, and Sergiu M. Dascalu are with the Evolutionary Computing Systems Lab, Department of Computer Science & Engineeering University of Nevada, Reno Reno, NV 89557, USA email: {quiroz, sushil, dascalus}@cse.unr.edu.

Amit Banerjee is with the School of Science, Engineering and Technology Pennsylvania State University, Harrisburg Middletown, PA 17057, USA email: aub25@psu.edu.

II. Related Work

Research on IGAs has been done in various research and real-world applications. Below we present a short survey of IGA work relating to mitigating user fatigue in IGAs, from the use of small populations to the use of machine learning algorithms to augment fitness evaluations.

A. 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"* [5]. 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 [16].

Llorá et al. make the user pick the best solution from a small subset of the population displayed [13]. 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 $s2$ [13]. 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 [6], [13]. For an in-depth discussion and applications of support vector machines see the work of Gunn, Burges, and Bennett and Campbell [9], [4], [2].

The work presented in this paper does not attempt to do any user modeling with machine learning techniques, yet it is discussed as future work for this line of work. 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 [10]. We address how to choose a good value for t in section IV. 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 [10] . 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.

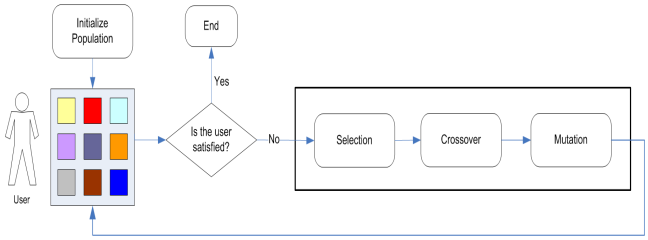

Fig. 1. Interactive Genetic Algorithm

III. Keep the Large Population - Evaluate a Few
IV. IGAs for UI Design

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

Graphical user interface development toolkits and libraries help UI 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 [**?**]. Therefore, UI designers use style guidelines and design principles to design usable and engaging interfaces. Style guidelines and design principles also help to evaluate a generated design. Guidelines define the look and feel of a UI in addition to addressing 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 include Apple's Human Interface Guidelines, Microsoft's User Interface Guidelines, Sun's Java Look and Feel Guidelines, and GNU's GNOME Human Interface Guidelines [**?**], [**?**], [**?**], [**?**].

The use of style guidelines and design principles leads to a couple of issues. The first issue is that *"interpreting the guidelines unambiguously and applying generic principles to a particular design problem is itself a major challenge"* [**?**]. Secondly, guidelines are either too specific or too vague, so they do not always apply to the problem at hand. For example, here is an excerpt from Apple's Human Interface Guidelines: *"use color to enhance the visual impact of your widgets"* [**?**]. This guideline is incomplete and confusing in that the guideline does not tell us which color to use for a given widget and on which context this principle should be applied. Such ambiguous guidelines force UI designers to make subjective decisions and evaluations to fill in omitted details.

A. Related Work on Evolution of UIs

Oliver et al. explored the evolution of the appearance and layout of websites [**?**]. 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 the algorithm replaces 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 of the best nine individuals, allowing us to sample the space of UIs more effectively and to present the user with potentially high fitness individuals.

B. Representation

We encode the UI representation in two chromosomes (figure 2). One chromosome encodes widget layout organization, and the second chromosome encodes widget characteristics (currently being widget color). We organize the widgets on a 10 rows by 2 columns grid. The grid, while allowing for limited layout designs, is simple for our initial experiments. In UI design a sizer usually manages widgets, and a grid sizer allows efficient widget organization in a layout. The grid layout also enforces alignment of widget, 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 **??**.

To preserve the integer representation of the layout chromosome, we use PMX, partial mapped crossover [**?**]. 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.

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.

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.

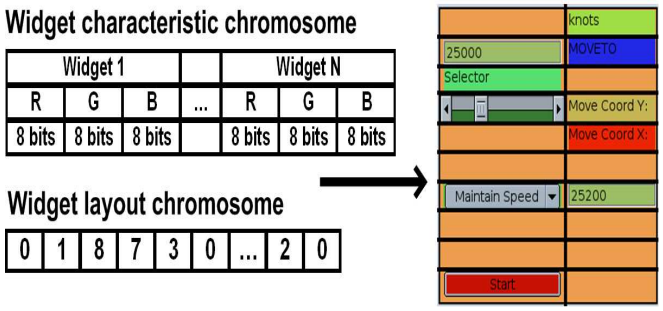

Fig. 2. UI encoding consists of two chromosomes. The widget characteristics chromosome encodes the color of each widget in a bit format. The widget layout chromosome encodes the position of the widgets in the grid. Widgets are identified by integer IDs greater than 0 and empty cells in the grid are identified with 0s.

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 on 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.

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 [**?**], [**?**]. Therefore, we decided to stick to RGB, treating RGB colors as vectors in a 3D color-space.

C. 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.

1) Subjective Evaluation: Our earlier work discusses our decision to choose a subset consisting of best individuals in the population [**?**], [**?**]. 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 individual 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.

$$subjective = b_s + w_s$$

We were fortunate to have our interpolation technique work well with the use of euclidean distance for the widget characteristics chromosome and hamming distance for the layout chromosome. A heuristic such as hamming distance might not work well for problems in other domains and with other representations. Each problem domain would require its own interpolation technique, either in the phenotypic space or in genotypic space with hamming distance, longest common subsequence, among others.

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 section **??**.

D. 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.

V. UI Evolution Results

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?

In the following sections we address these questions by first conducting experiments using a simulated user, and second by conducting experiments with three real users. The simulated user gave us the leverage to conduct the first set of experiments and to test our approach with a hypothetical tireless user.

A. Simulated Users

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.

As expected, we found that using tournament selection with a tournament size of 4 outperformed roulette wheel selection (see figure 3). The figure shows the best individuals in the population. Tournament selection's stronger selection pressure leads to much quicker convergence to better values.

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.

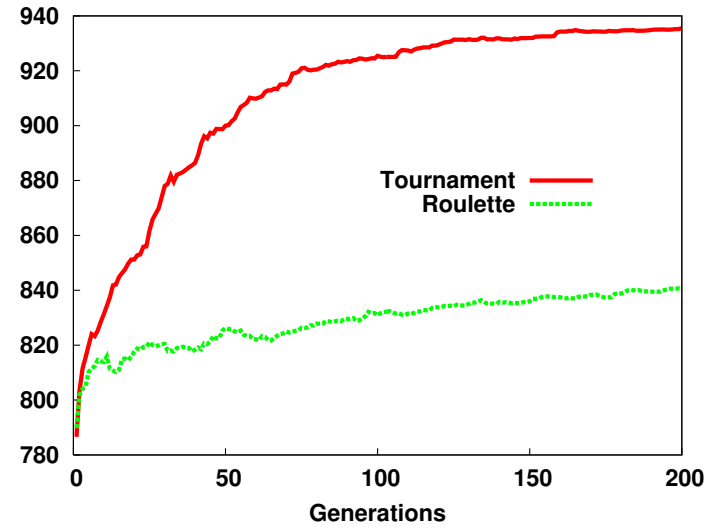

Fig. 3. Tournament selection versus roulette wheel selection. The plot shows the best individuals in the population.

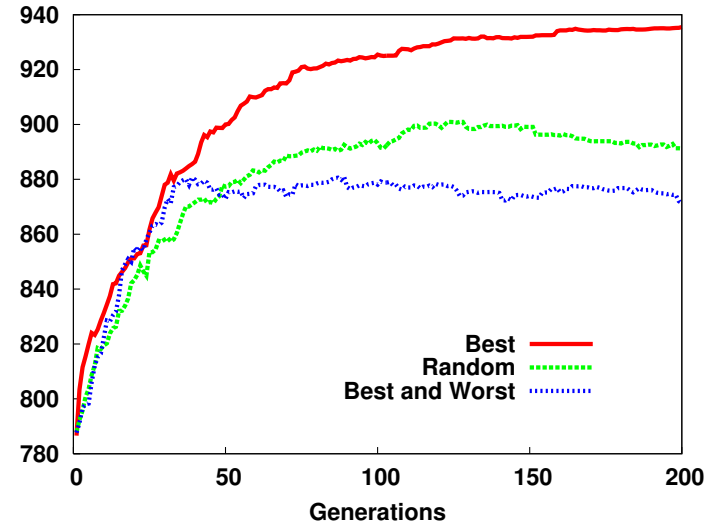

Fig. 4. Subset display method comparison.

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 4. 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 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.

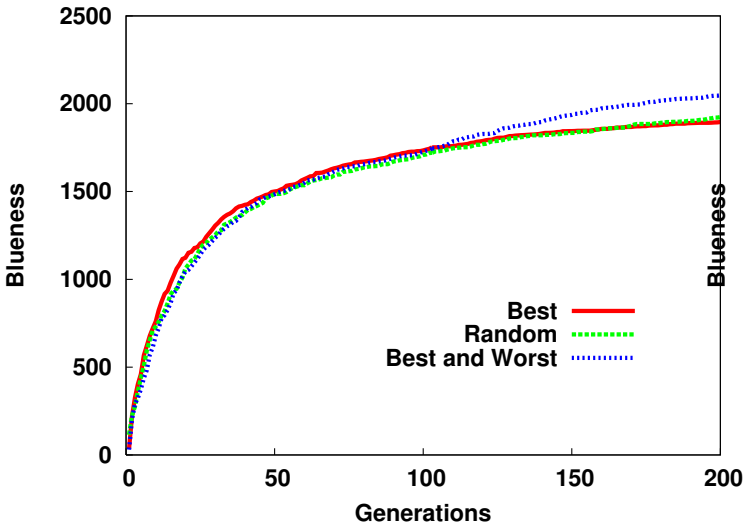

Fig. 5. Subset display method comparison on convergence to blue widgets.

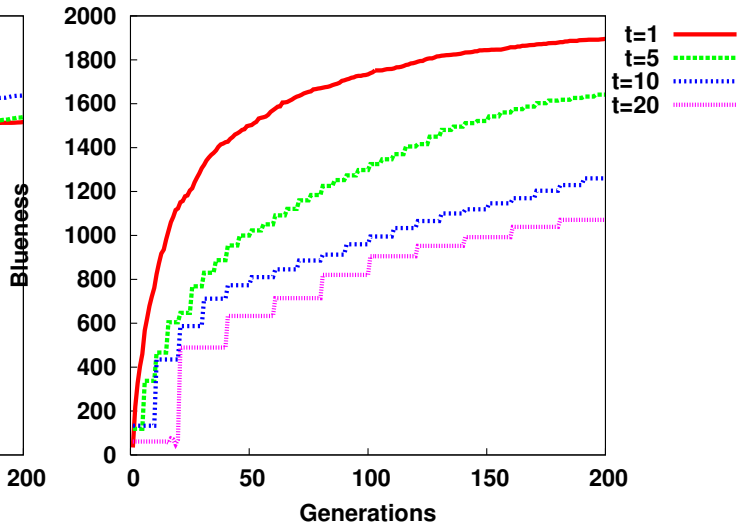

Fig. 7. Effect of varying t on convergence to blue UIs.

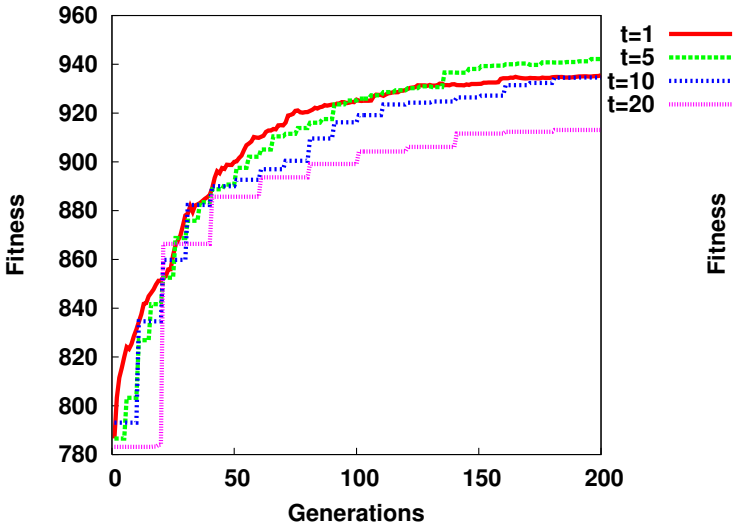

Fig. 6. Effect of varying t on IGA performance.

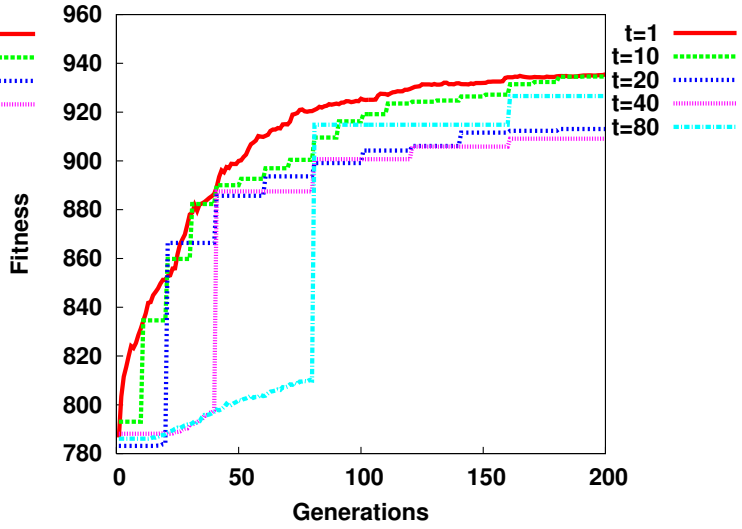

Fig. 8. Degradation on the IGA performance (maximum) for high t values.

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 6 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 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 7 (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 8 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 9 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 increase in fitness.

We also plotted the convergence to blue UIs, which was the user assumed preference. Figure 10 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 11 shows the average blueness of individuals in the population indicating that the average performance correlates well with best performance.

3) User Interfaces Generated: Figures 12 and 13 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

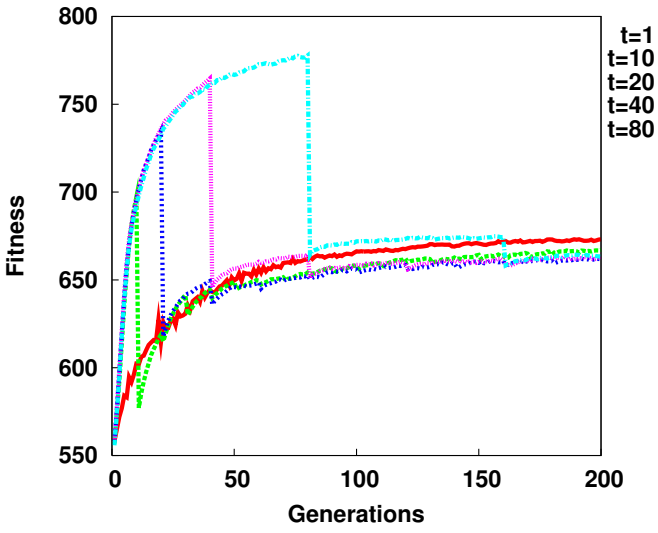

Fig. 9. Degradation on the IGA performance (average) when using hi values.

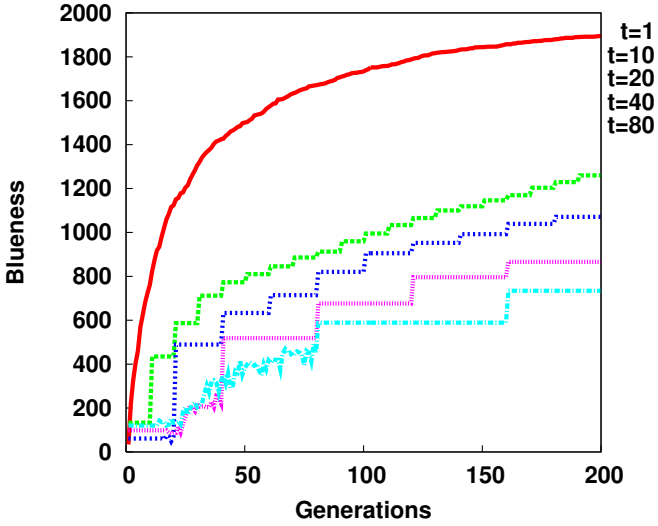

Fig. 10. Degradation in the convergence (maximum) to blue UIs when high t values.

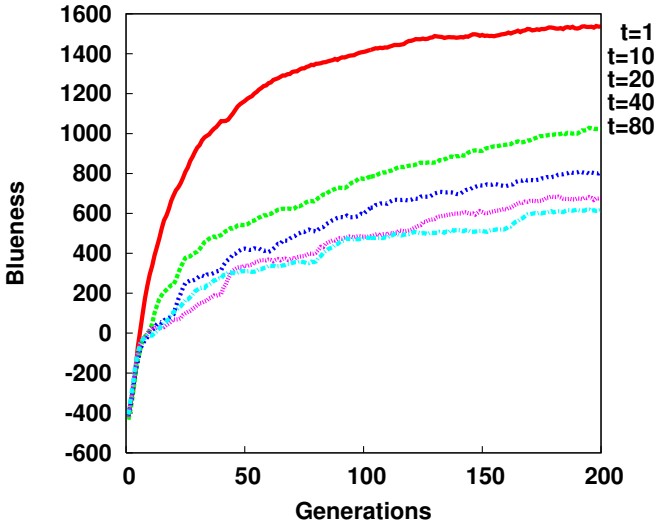

Fig. 11. Degradation in the convergence (average) to blue UIs when using high t values.

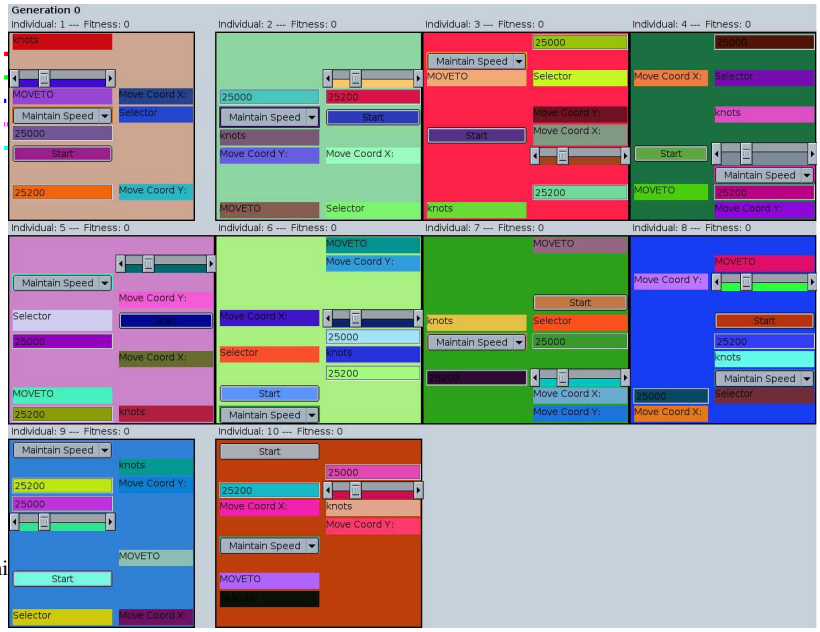

Fig. 12. Displaying the best 10 individuals for user evaluation at generation 0.

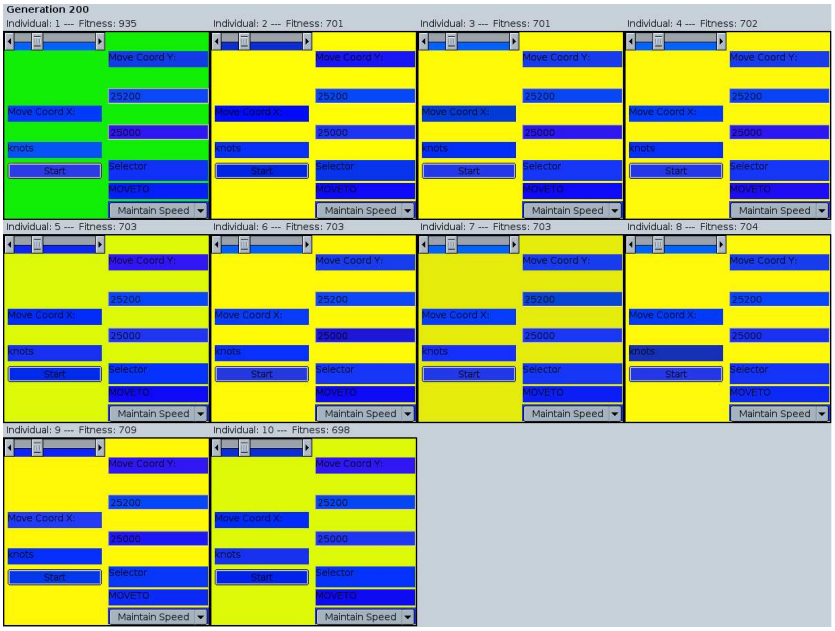

Fig. 13. Displaying the best 10 individuals for user evaluation at generation 200.

assumed preference. The UIs at generation 200 both 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.

B. Real Users

We collected data from 3 users. Our IGA's parameter settings were the following: (1) population size of 100, (2) we displayed 9 individuals for user evaluation, and (3) we used 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).

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 experiment investigated the effects of delayed user input, and explored how varying t affects convergence behavior and performance with real users.

We plotted the fitness convergence for our three study subjects: $user1$, $user2$, and $user3$. Figure 14 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 15 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 16 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 17, 18, and 19 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 [**?**].

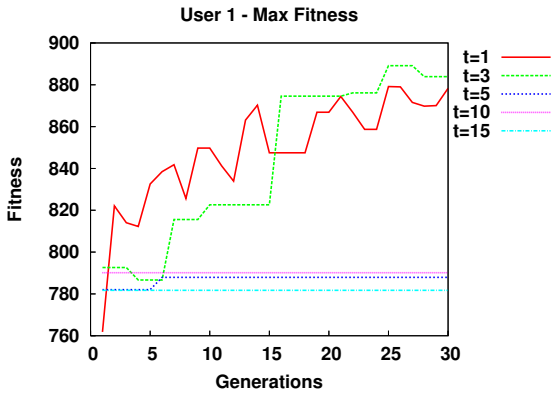

Fig. 14. Fitness performance of user 2. The plot shows the best individuals in the population.

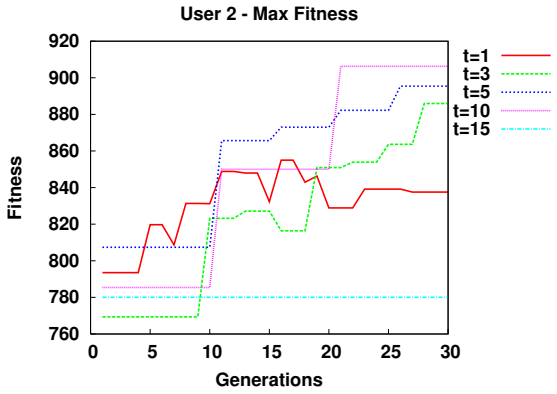

Fig. 15. Fitness performance of user 2. The plot shows the best individuals in the population.

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 14) and user2 (figure 15). 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

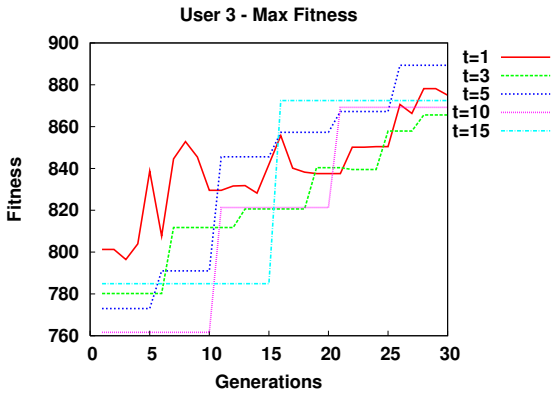

Fig. 16. Fitness performance of user 3. The plot shows the best individuals in the population.

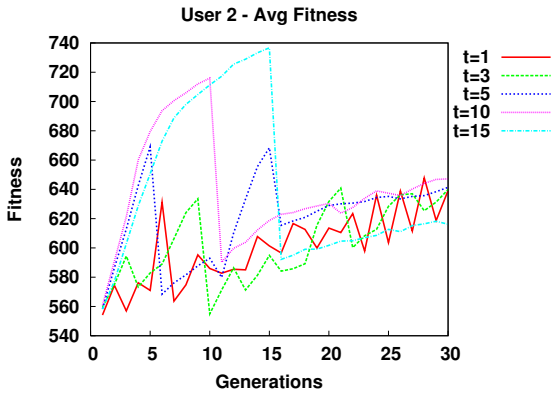

Fig. 18. Fitness performance of user 2. The plot shows the average individuals in the population.

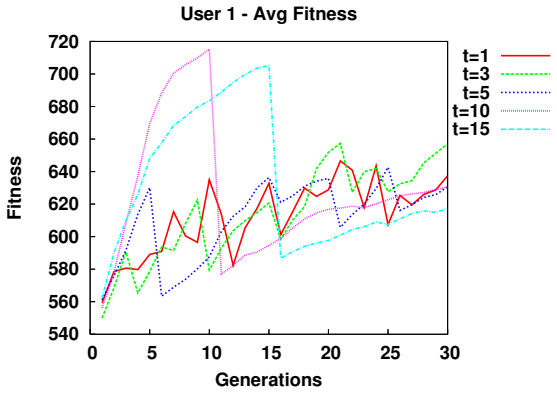

Fig. 17. Fitness performance of user 1. The plot shows the average individuals in the population.

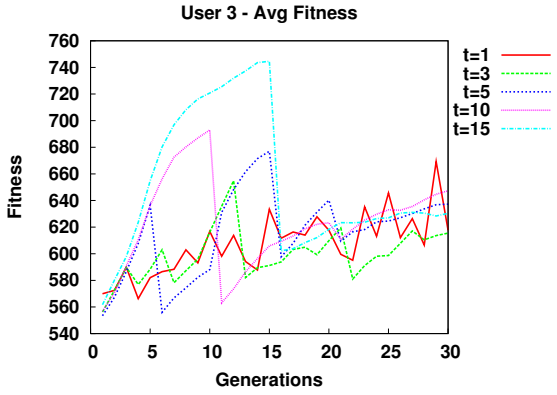

Fig. 19. Fitness performance of user 3. The plot shows the average individuals in the population.

behavior was less common with low values of t since the user has more opportunities to change his/her selection.

Figure 20 shows these results. The plot shows the fitness convergence of the best 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 14 and 15 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.

C. 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. Our impression from the users' feedback and from their behavior during the IGA runs leads to believe that color was the primary decision factor for evaluating the UIs presented. 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.

D. UIs Generated

Figures 21 and 22 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 user3's preferences and which best follow coded guideline metrics.

VI. Doing More With Less - Take From Peers

The computational model of creative design based on collaborative IGAs is shown in figure 23. The figure illustrates

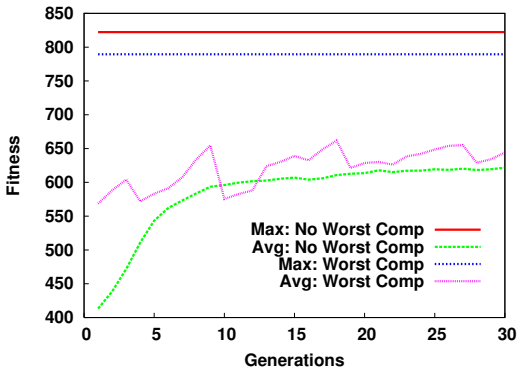

Fig. 20. Fitness performance of population with comparison to user selected worst turned off and comparison to the user selected best turned on.

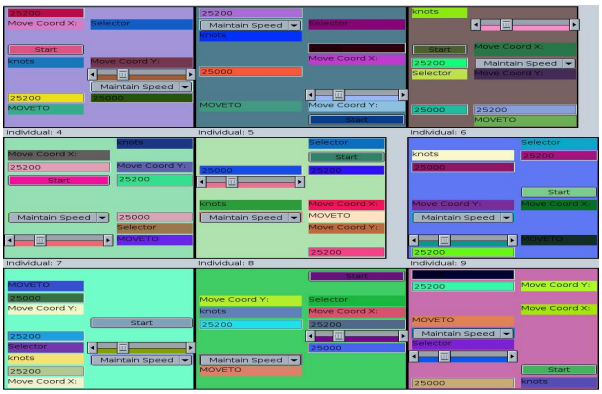

Fig. 21. The best nine individuals in the initial population.

three users collaborating with each other, with each of the peers denoted by the dotted boxes. Each user interacts with a GA by acting as the subjective evaluation. As shown, the evaluation is not purely subjective, instead the evaluation of design solutions consists of the multi-objective optimization of the subjective and objective criteria. We use Pareto optimality to maximize these criteria. The arrows between the GAs of each of the peers represent the communication that takes place between the peers. If a user likes a design solution from one of his/her peers, then the user has the option to inject that solution into his/her population, thus introducing a search bias.

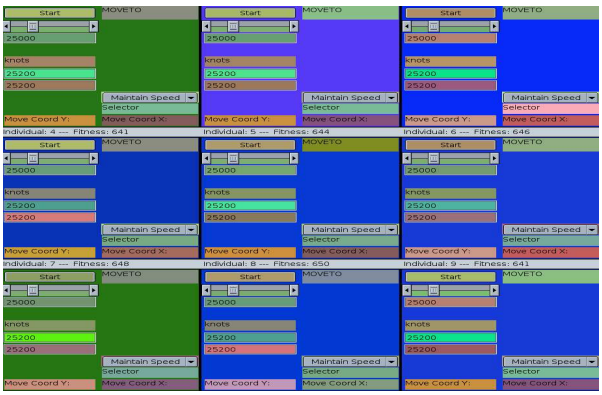

Fig. 22. The best nine individuals at session end.

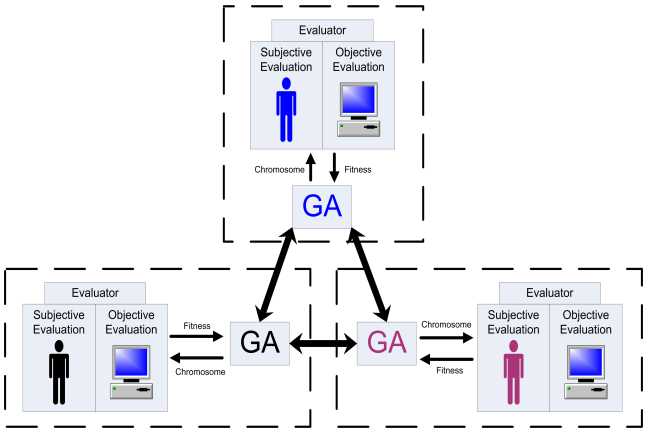

Fig. 23. Computational Model of Creative Design

A. IGAP: Interactive Genetic Algorithm Peer-to-Peer

IGAP is the framework we implemented to test the computational model of creative design. Figure 24 shows the steps involved in IGAP. Each peer acts as an independent node, running as a server which handles incoming requests from peers. On a request, the peer node sends a subset of its best genomes to the requesting peer. The requesting peer, assuming collaboration with more than one peer, constructs a genome pool from all the genomes received from all of its peers. From the genome pool the requesting peer node then selects a random subset to display on the screen of the designer.

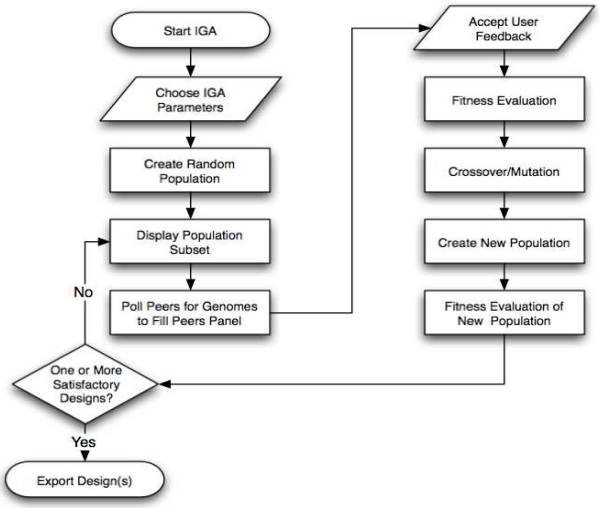

Fig. 24. IGAP Process

B. Multi-objective Optimization

We use the Non-dominated Sorted multi-objective Genetic Algorithm (NSGA-II) to evolve floorplans [7]. The NSGA-II creates fronts of non-dominated individuals, where within a front none of the individuals are any worse than any other individual across all optimization criteria and all individuals within a front are said to have the same rank. We select parents by using the crowded distance tournament. We pick

two individuals to participate in the tournament, and we select the individual with the higher rank to be part of the mating pool. In case the two individuals have the same rank, and consequently belong to the same front, then the crowded distance of both individuals is computed, and we select the individual with the highest crowded distance to be part of the mating pool. This translates to the individual being in a less crowded region of the front and hence, the crowded distance selection favors the most diverse individuals within a front.

We use NSGA-II with a two-criterion multi-objective function: objective fitness and subjective fitness. Previously, we had used a five-criterion multi-objective function, where two of the five criteria measured objective guidelines, while the remaining three criteria measured subjective preferences. However, studies have shown that the performance of the NSGA-II degrades when using more than three criteria during multi-objective optimization [17].

C. A Special Case of Case-Injected Genetic Algorithms

A case-injected genetic algorithm (CIGAR) works differently than a typical GA. A GA randomly initializes its starting population so that it can proceed from an unbiased sample of the search space. The methodology behind CIGARs is that it makes less sense to start a problem solving search attempt from scratch when previous search attempts (on similar problems) may have yielded useful information about the search space [14]. Instead, periodically injecting a GA's population with relevant solutions or partial solutions to similar previously solved problems can provide information (a search bias) that reduces the time taken to find a quality solution. This approach borrows ideas from case-based reasoning (CBR) in which old problem and solution information, stored as cases in a case-base, helps solve a new problem [12]. The collaborative IGA computational model is a special case of CIGARs, where the designer during the interactive evolutionary session determines when and how many individuals to inject into the population, instead of being done in an algorithmic fashion [14]. Furthermore, in CIGARs the side effect occurs in one direction, with the individuals injected from the case base affecting the performance of the running GA. In our model, when a designer chooses to inject a solution from one of his/her peers, the introduced bias will not only become apparent in the designer's own population, but the other peers will also be able to view this change as well, since designers can always see a subset of each others' designs.

D. Collaborative Methodology

Collaborative evolution is implemented with a peer to peer network. We treat each user participating in evolution as a node, handling incoming requests from other nodes (peers) and requesting information from peers. By using a peer to peer network, control is decentralized and each node is free to chose who to connect to and if necessary who to exclude from its set of peers.

The interface during an individual evolutionary session is shown in figure 25, while the interface during collaborative 26. During collaborative evolution, a subset of peer-evolved designs is displayed to the right of the user's population. We limit the number of peer individuals to nine, organized in a 3x3 grid, similar to how we present the user's own population, in order to be consistent. For more than one peer, we cannot display all the individuals belonging to the subset of each peer, since we only display nine. We do make sure that the user selected best individuals from each peer are displayed on the peers subset. We save the user selected best from generation to generation, and we always make it part of the subset displayed the next time the IGA requires user input. We select the rest of the individuals that make up the peers subset by taking a random subset from a collective pool of all individuals that make up peers' subsets. By selecting a random subset, we believe that over many generations, all of the participants will get approximately the same amount of their designs displayed on the screens of collaborators.

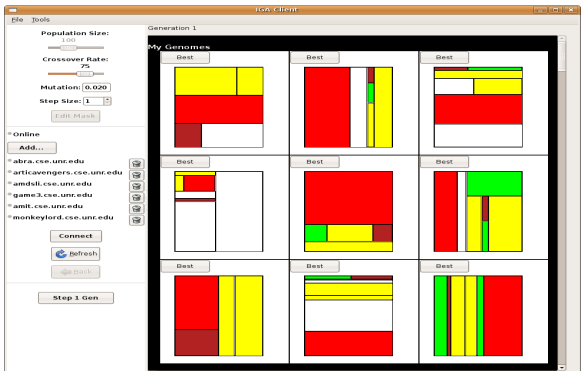

Fig. 25. Screen shot of individual floorplan designing.

The benefit of viewing the best individuals from peers is limited, unless the user is able to take promising individuals from peers and mold them to their liking. We support this by allowing the user to inject individuals from the subset of peers into the user's own population. The user can select an individual from a peer to be added to the user's own gene pool by clicking on the "Add to Genome" button. The user can also select a best individual from the subset of individuals from peers, in which case the user selected best is automatically injected into the population, and used for fitness interpolation. We require the user to select a best individual, but it does not have to be from the user's own population - the user selected best can come from peers.

The injected individuals replace the bottom 10% of the population as done in [14]. If the number of injected individuals is less than 10% of the population, then we insert numerous copies of the injected individuals, until the total sum of the injected individuals is 10%. In case-injected GAs (CIGARs) typically a case base is kept of solutions to previously solved problems, and based on problem similarity, individuals similar to the best individuals in the current population are periodically injected, replacing the worst individuals [14]. In our algorithm, the designer plays the role of determining how many, when, and which individuals to inject at any step during the collaborative evolutionary process. If the injected individuals make a positive contribution to the overall population, then

they will continue to reproduce and live on, while injected individuals which do not improve the population performance will eventually die off. Hence, the user is not penalized for injecting subpar individuals.

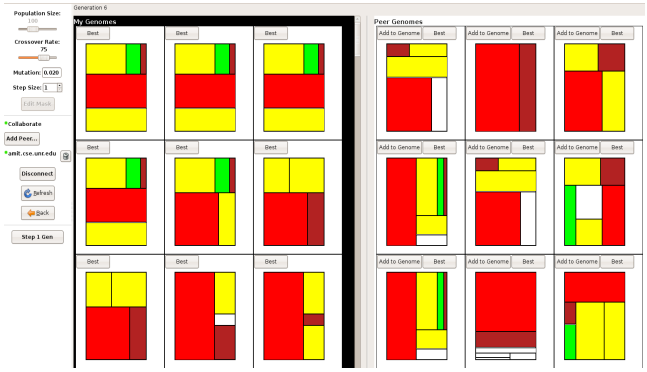

Fig. 26. Screen shot of collaborative floorplan designing.

E. Fitness Biasing

We use fitness biasing to ensure that injected individuals survive long enough to leave a mark on the host population. We use the concept of bloodline to do fitness biasing. Injected individuals are considered to be full blood, while those individuals already in the population are treated as individuals with no blood. The bloodline consists of a number between 0 (no blood) and 1 (full blood).Thus injected individuals will all be non-dominated (in the topmost front) and will not die off immediately. When a full-blooded individual crosses over with a no-blooded individual, then the offspring will inherit a bloodline value equal to a weighted sum of the bloodline of the parents, where the weight values depend on the percentage of the genetic material inherited from each parent. This is shown in equation 1, where $p1$ is the percent of genetic material inherited from the first parent, $p1_{blood}$ is the bloodline value of the first parent, $p2$ is the percent of genetic material inherited from the second parent ($p2 = 1 - p1$), and $p2_{blood}$ is the bloodline value of the second parent.

$$child\ blood = p1 * p1_{blood} + p2 * p2_{blood} \quad (1)$$

VII. IGAs for Floorplanning

We use floorplanning as the case study to test the model of creative design. The rooms in the floorplans are color coded as red (living area), yellow (bedrooms), green (eating areas - kitchen and/or dining rooms), firebrick (bathrooms), and white (empty spaces).

A. Floorplan Representation

For evolving floorplans we have used a binary tree representation, coded as a nested list. At every node of the tree, the parameters specify how the rectangular panel at that level is subdivided (either left/right or top/bottom) and the percentage of panel area at that level contained in either the left or the top subdivision. Figure 27 shows how the rectangular panel is subdivided into rooms and spaces. A room is represented by the array [0, 1] and a space by [0, 0]. An arbitrary array [0, 0.75] represents division in top/bottom configuration with top sub-panel containing 75% of the parent panel. Another list [1, 0.80] represents division in left/right configuration with left sub-panel containing 80% of the parent panel. Even though the representation is quite intuitive, it only allows us to represent rectangular shapes. To represent more complicated (possibly organic) shapes, a more complex representation is needed. The plans are decoded according to the guidelines in [15] - depending on the number of rooms (the number of [0, 1] arrays in the encoding) and their relative sizes, the guidelines have explicit instructions pertaining to room labels (living, bed, kitchen etc.). For example if a particular plan has two rooms, the bigger room is labeled as the Living-Bed-Kitchen (studio configuration) and the smaller room is labeled as the Restroom. For plans with more than three rooms or more, the bedrooms are separated from the living room.

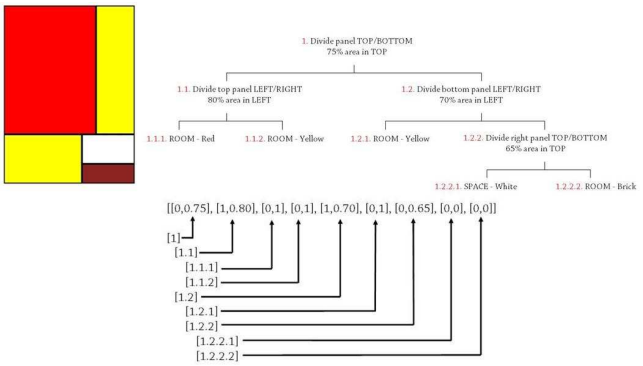

Fig. 27. The binary tree representation of a floorplan is encoded as a nested list.

The binary tree representation for floorplans necessitates the need for a specialized tree-crossover operator. The nested list is parsed as a binary tree and two such parent trees are crossed at randomly chosen nodes, such that entire sub-trees following those nodes are swapped. The tree representation is used in genetic programming [11] and hence, our crossover operator maps to the crossover operator used in genetic programming. Depending on the probability of mutation, the mutation operator works on the two parameters of the nodes (or leaves) differently. It performs a binary swap on the first parameter thereby changing the subdivision configuration. Depending on the value of the second parameter, the operator either performs a binary swap (if the value is either 0 or 1), thereby changing a room to a space and vice versa, or if the second parameter is a real number between 0 and 1, the operator replaces it by another random real number in the same interval, thereby altering the dimensions of the room (or the space).

B. Fitness Evaluation

1) Objective Evaluation: Plans are compared room-wise to ascertain if they meet the minimum dimension and area criteria in [15]. The guidelines for a two-room single-storey house plan call for the bigger room to be at least $300ft^2$ and have a minimum dimension of $20^{'}10^{"}$. The number of

rules that a particular plan needs to adhere to increases with the number of rooms. The objective measure assigned to a two-room plan that satisfies the minimum area and minimum dimension requirement is [0.0, 0.0]. If the area of the bigger room of a certain other plan is $area$ such that $area < 300ft^2$, and if the minimum dimension of the bigger room is $minlen$ such that $minlen < 20^{'}10^{"}$, then the objective minimization measure is given by equation 2.

$$obj = [\frac{300 - area}{300}, \frac{20^{'}10^{"} - minlen}{20^{'}10^{"}}] \quad (2)$$

2) Subjective Evaluation: We also compare the plans to the user-selected best on three criteria. The three criteria are: (1) number of rooms, (2) room adjacencies, and (3) total built area of the plan. The first criteria compares the number of rooms in the user-selected best plan to the particular plan in question. If the user-selected best plan has $bnumrm$ rooms, a plan with $numrm$ number of rooms is assigned a measure of $|bnumrm - numrm|$, if $1 < numrm < 8$, then it is assigned a measure of:

$$1 + max(bnumrm - 2, 8 - bnumrm) \quad (3)$$

In order to compare plans for similarity (or dissimilarity) in room adjacencies, we compare adjacency similarity in certain pairs of rooms, such as living-bedroom adjacency, restroom-bedroom adjacency, kitchen-dining area adjacency, and restroom-kitchen adjacency. This information is stored in a four-bit string (1 for adjacency and 0 for no-adjacency). The Euclidean distance between the bit string for the user-selected best plan and the bit string for the particular plan provides the second subjective measure. For plans that do not have separate kitchens or dining areas or bedrooms, the default adjacency measure is always unity. The third criteria measures the similarity between the plan in question and user-selected best plan in terms of total built area, i.e. area occupied by all rooms. If the user-selected best plan has a total built area of $besttotarea$, and the plan in question has a total built area of $totarea$, then the penalty associated with the third criteria is given by, $(besttotarea - totarea)/besttotarea$ if $totarea < besttotarea$. This is also treated as a minimum subjective requirement, hence there is no default penalty (plans with more built area than the user-selected best get a penalty measure of 0.0). These three measures, added together together and normalized, constitute the subjective penalty function of NSGA-II.

VIII. Floorplanning Results

A. Experimental Setup

We had 20 participants in our study, eight females and 12 males. Out of the 20 participants, 11 were from engineering and math, two were undeclared, and seven were from social sciences. Participants were assigned to groups of four based on schedule availability. We picked groups of four, so that using a 3x3 display grid, allowed for three floorplans from each peer to be displayed on the screen of every other participant.

Participants first were allowed to get familiar with the IGA. They were instructed in how to guide the process, both individually and collaboratively. The participants were told the set of requirements which they would have to follow after the tutorial. This was done so that participants could develop an intuition and a sense for how the system worked. The participants were not told that they were using an evolutionary system, they were simply told that after selecting the best floorplan, the screen would refresh, displaying a new set of floorplans would be similar to what they previously selected as the best.

The set of floorplan requirements given to the participants were: (1) Create a floorplan for a 2 bedroom, 1 bathroom apartment, (2) the bathrooms should be close to the bedrooms, and (3) the bathrooms should be far from the kitchen and dining room areas. During the tutorial phase participants were given the requirements, so that they could practice guiding the IGA to floorplans that meet the given requirements. We also meant for the tutorial to remove any bias with regards to unfamiliarity with the system and with IGAs. Participants were allowed to run for as many generations as they wished. Once the participants had found a floorplan that met all requirements and that they also liked, then they would make a final selection of the best floorplan, and then quit the program.

After the tutorial session, the participants then were instructed to create a floorplan that met all of the requirements individually. Following this, the participants created a floorplan with collaboration. During the collaborative run, the participants were allowed to inject as many designs from their peers as they wished, but their final floorplan selection had to come from their own collection of floorplans.

In each group, every participant picked one final floorplan from the individual run as the best and one final floorplan from the collaborative run as the best. Each participant in the group then graded the two best floorplans selected from each of his/her peers, so that each participant evaluated six floorplans - three collaborative and three individual floorplans. The floorplans were evaluated by the participants using the following criteria: (1) Appealing - unappealing, (2) average - revolutionary, (3) commonplace - original, (4) conventional - unconventional, (5) dull - exciting, (6) fresh - routine, (7) novel - predictable, (8) unique - ordinary (9) usual - unusual, and (10) meets all requirements - does not meet requirements.

Each of these criteria was scored using a seven-point Likert scale. The criteria was subset derived from the Creative Product Semantic Scale [3], [1]. The criteria were randomized for each of the floorplans, to make sure the participants were alert and to make sure they read the criteria before providing a score. The participants were given as much time as necessary to complete the evaluation.

B. Results

Assuming the exploration of a large design solution space, we ask whether collaboration amongst peers is sufficient to allow for the potential to produce creative content in designing without explicitly expanding the design solution space by adding one or more variables. Our hypothesis was that collaboration would be sufficient to produce creative content, and that

designs evolved collaboratively would consistently rank higher in the evaluation criteria than those created individually.

The compiled evaluation results for all groups are shown in table **??**. The table shows all evaluation criteria in the first column. The second column, "Desired Value", specifies the desired range in the seven-point Likert scale that would support our hypothesis. For the first criterion, "Appealing - Unappealing", a value of 1 would represent that a given floorplan was "appealing" while a value of 7 would represent that a given floorplan was "unappealing". For the second criterion, "Average - Revolutionary", a value of 1 would represent that a given floorplan was "average" while a value of 7 would represent that a given floorplan was "revolutionary", and so on.

The third and fourth columns in table **??** show individual and collaborative averages. The individual average provides the average score received by each floorplan evolved individually by each of the 20 participants. The collaborative average provides the average score received by each floorplan evolved collaboratively by each of the 20 participants. The fifth and sixth columns show the corresponding standard deviations. The last column shows the corresponding p-value for each criterion.

By looking at the p-value, we can say that the floorplans created collaboratively ranked slightly higher in the "average-revolutionary" and the "commonplace-original" criteria. For the other criteria, the differences in the averages are not statistically significant. However, even though the floorplans created collaboratively were considered to be more "revolutionary" and "original" than those created individually, the average scores are near the median of the seven-point Likert. Ideally, we would have liked to see these values farther apart, with the values for "revolutionary" and "original" being closer to 7 rather than to 4. Finally, we can see that the participants, either individually or with collaboration were able to effectively bias the floorplan designs subjectively to designs which met most of the requirements, as is shown by the low average scores obtained in the requirements criterion.

From the results obtained we can deduce that perhaps collaboration was not sufficient to make a clear distiction in the creative value between the individual and collaborative floorplans. There are some issues that came up during evaluation that can shed some light on these results. The first is that the participants were not told explicitly which of the contrasting adjectives in the evaluation criteria were positive and which were negative. For example, some participants expressed that while some of the floorplans were "unappealing", because they would not have liked to live in such an apartment, they found the floorplan "appealing" because of its innovative, and at times, bizarre room layouts. Hence, there was some ambiguity in how to evaluate the resulting flooroplan designs. Another issue is the applicability of some of the criteria to floorplan design. For a student, a floorplan for an apartment might not be something that would instill a feeling of "exciting", even if the floorplan had a creative layout. A domain expert, such as an architect, might have a more refined appreciation of the quality of the designs, which might have yielded radically different results. Finally, the participants had one try at creating a floorplan individually and one try at creating a floorplan collaboratively. Asking the participants to evolve more than one floorplan individually and collaboratively might have also yielded a more significant difference in the results.

From our observations and feedback from the participants, we found that (while not explicitly shown by the numbers) the participants found evolution of the floorplan designs during the collaborative session to be easier. During individual evolution, the floorplans tended to converge, as expected, to floorplans which were high fitness but which differed slightly in terms of room dimensions and room layout. On the other hand, during collaborative evolution, the participants were exposed to diverse high fitness individuals which belonged to the peers. Some of the participants also used the ability to inject numerous design solutions from peers as a mechanism to manage diversity in their own populations. We gave the participants no limit to the number of generations (number of picks) before they had to pick their final floorplan selection. While the system was designed to support exploration, many participants looked through the entire population of floorplans to find a design which met the requirements, and stopped after one to two generations, instead of taking advantage of the evolutionary process in order to try and breed new interesting floorplan designs. Other participants restricted themselves to picking from the subset of the best nine floorplans, and picked as many as 10 generations, until they were certain that their population had converged. We found that many participants concentrated in finding a floorplan that met all requirements and subsequently stopped, even if the floorplan looked uninhabitable. We believe this might be due to the system's lack of affordance, which might have encouraged users to continue exploring, and to lack of motivation, since participants had no real incentive to continue exploring and find a better looking floorplan. In average all participants evolved the floorplan designs in less than 10 minutes.

IX. Future Work

The current case study was chosen in part because it had a convenient digital representation, thus it was well suited for our initial tests. However, we intend to apply our computational model to other case studies, which will require different and more complex representations, in order to further test the validity of the model.

In the work presented, we had the group members evaluate each other's designs. We are interested in further evaluation of the resulting designs by domain experts. In the case of floorplanning, a group of architects could evaluate the resulting designs.

Finally, we will implement the addition of design variables to the computational model of design, so that the system conforms to the definition of creative design followed by computational models described in the literature [8]. This can be implemented in IGAP by allowing individual designers to start their corresponding evolutionary processes with a subset of all design variables being evolved, with the rest fixed. For example, assuming two individuals are collaborating, and the four variables are being evolved for design exploration, then

each of the users could start having two variables fixed and two being evolved. In this scenario, case injection through collaboration would expand the initial search space, from exploring designs with two evolving variables to exploring the space of designs with four evolving variables.

X. Purely Subjective Onemax: Subset Method Comparison

A. Varying Step Size

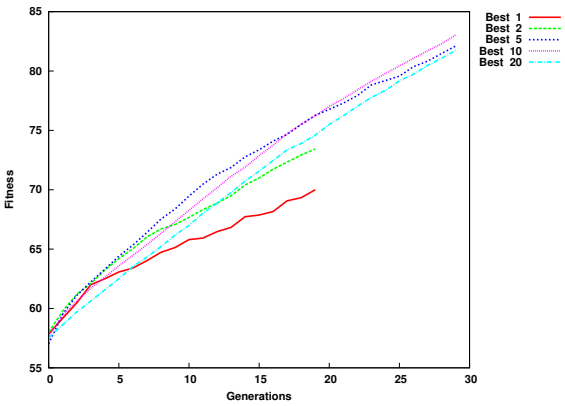

Fig. 28. Best

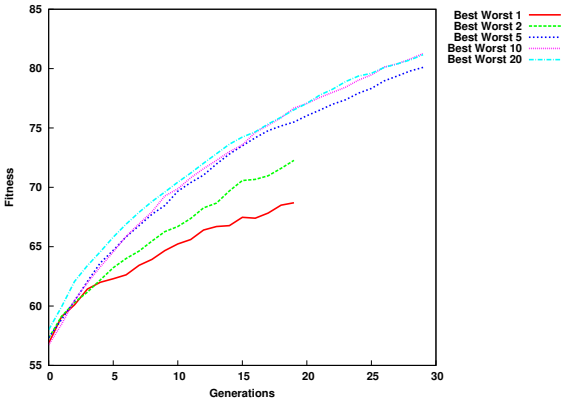

Fig. 29. Best and worst

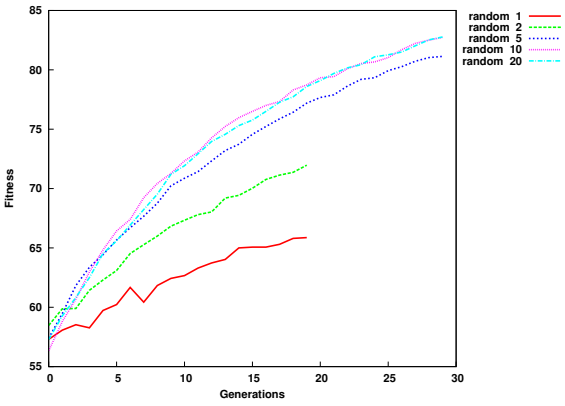

Fig. 30. Random

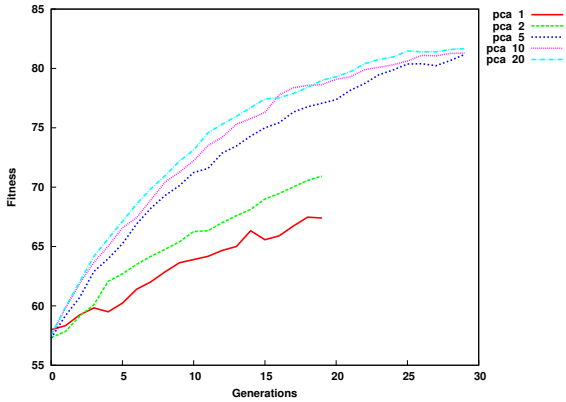

Fig. 31. PCA with k-means

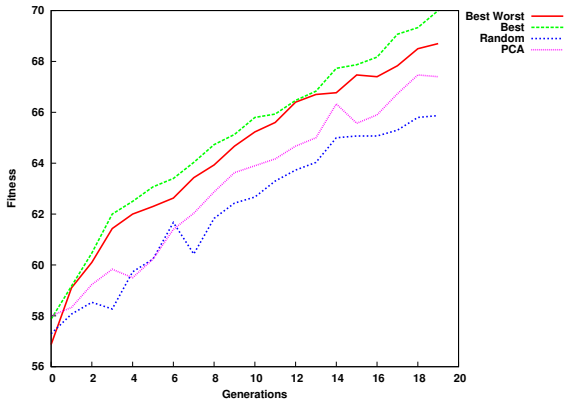

Fig. 32. Step 1

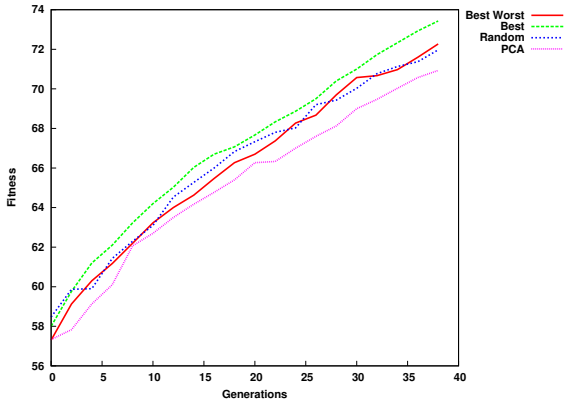

Fig. 33. Step 2

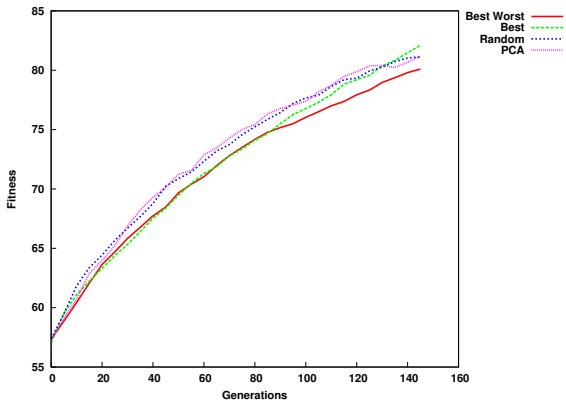

Fig. 34. Step 5

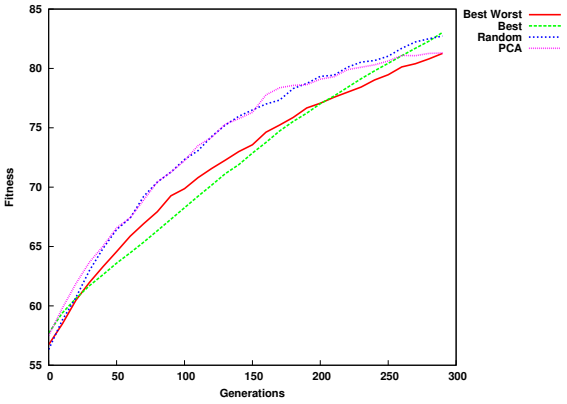

Fig. 35. Step 10

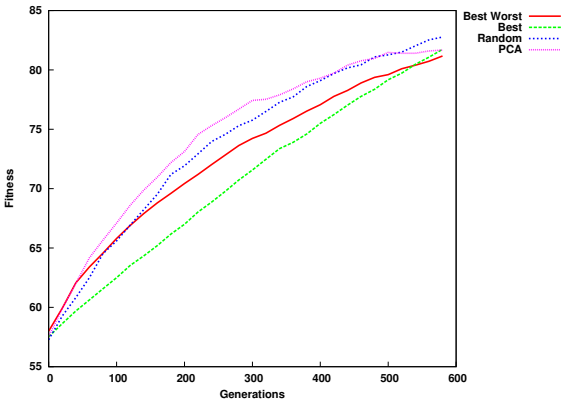

Fig. 36. Step 20

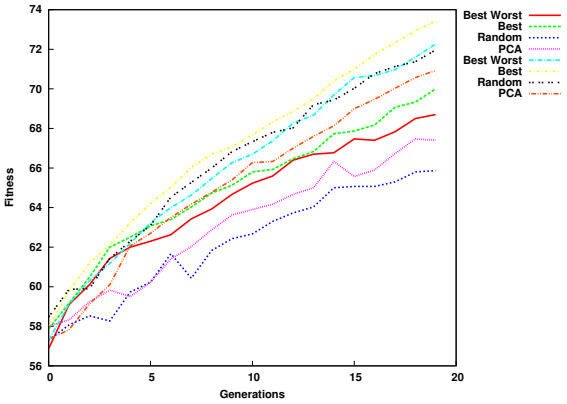

Fig. 37. Step 1 vs 2

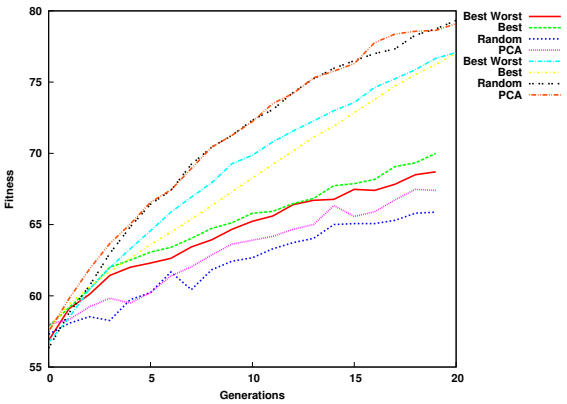

Fig. 38. Step 1 vs 10

B. Varying Subset Method

XI. Conclusion

We have presented a computational model of creative design based on collaborative interactive genetic algorithms. We showed the potential of the model to produce creative content by analyzing design space exploration. Our results showed that those floorplans created collaboratively scored slightly higher in terms of "revolutionary" and "unconventional" criteria than floorplans created individually. We expected collaboration in our computational model to provide enough potential to produce creative designs. However, from our results we can conclude that in the majority of the criteria, floorplans created individually scored similarly to those created collaboratively. Thus, there is a need to combine collaboration in our model with an explicit expansion of the design solution space by adding one or more variables.

XII. Acknowledgments

We thank the study participants for their time. This work was supported in part by contract number N00014-0301-0104 from the Office of Naval Research and the National Science Foundation under Grant no. 0447416.

References

[1] J. Andrews and D. C. Smith. In search of the marketing imagination: Factors affecting the creativity of marketing programs for mature products. *Journal of Marketing Research*, 33:174–187, May 1996.

[2] K. P. Bennett and C. Campbell. Support vector machines: hype or hallelujah. *SIGKDD Explor. Newsl*, 2:1–13, 2000.

[3] S. Besemer and K. O'Quin. Analyzing creative products: Refinement and test of a judging instrument. *Journal of Creative Behavior*, 20:115–126, 1986.

[4] Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, June 1998.

[5] Cho. Towards creative evolutionary systems with interactive genetic algorithm. *Applied Intelligence*, 16:129–138, Mar. 2002.

[6] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley and Sons, 2001.

[7] K. Deb, K. Deb, A. Pratap, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6:182–197, 2002.

[8] J. S. Gero. Computational models of creative designing based on situated cognition. pages 3–10, Loughborough, UK, 2002. ACM.

[9] S. R. Gunn. Support vector machines for classification and regression. *ISIS technical report*, 14, 1998.

[10] R. Kamalian, Y. Zhang, H. Takagi, and A. Agogino. Reduced human fatigue interactive evolutionary computation for micromachine design. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 9, pages 5666–5671 Vol. 9, 2005.

[11] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Dec. 1992.

[12] D. B. Leake. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press, Aug. 1996.

[13] X. Llorà, K. Sastry, D. E. Goldberg, A. Gupta, and L. Lakshmi. Combating user fatigue in igas: partial ordering, support vector machines, and synthetic fitness. pages 1363–1370, Washington DC, USA, 2005. ACM.

[14] S. Louis and C. Miles. Playing to learn: case-injected genetic algorithms for learning to play computer games. *Evolutionary Computation, IEEE Transactions on*, 9:669–681, 2005.

[15] E. Neufert, P. Neufert, B. Baiche, and N. Walliman. *Architects' Data*. Wiley-Blackwell, 3 edition, Aug. 2002.

[16] H. Takagi. Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89:1275–1296, 2001.

[17] S. Tiwari, P. Koch, G. Fadel, and K. Deb. Amga: an archive-based micro genetic algorithm for multi-objective optimization. pages 729–736, Atlanta, GA, USA, 2008. ACM.