Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Application of a Seeded Hybrid Genetic Algorithm for User Interface Design

Nicholas Hardman, John Colombi
David Jacques, Raymond Hill
Air Force Institute of Technology
Wright-Patterson, Ohio 45433 USA
nicholas.hardman@afit.edu

Janet Miller
711th Human Performance Wing
Wright-Patterson, Ohio 45433 USA

***Abstract*— Studies have established that computer user interface (UI) design is a primary contributor to people's experiences with modern technology; however, current UI development remains more art than quantifiable science. In this paper, we study the use of search algorithms to predict optimal display layouts early in system design. This has the potential to greatly reduce the cost and improve the quality of UI development. Specifically, we demonstrate a hybrid genetic algorithm and pattern search optimization process that makes use of human performance modeling to quantify known design principles. We show how this approach can be tailored by capturing contextual factors in order to properly seed and tune the genetic algorithm. Finally, we demonstrate the ability of this process to discover superior layouts as compared to manual qualitative methods.**

Keywords— User interface design, human performance modeling, genetic algorithm, pattern search, human-computer interaction, human-machine interface

I. Introduction

In the modern world, most people have daily interaction with computer-based information sources. The user interfaces (UIs) that moderate those interactions are, too often, less than optimally designed. This paper extends the research on one aspect of UI design that was originally presented by the authors in [1]. In that paper, we presented an approach to display layout design based on human performance modeling. The paper proposed metrics, that, when used with the approach, allowed for a quantitative evaluation of layout effectiveness. The metrics have been positively correlated with actual human subjects experiments in [2] by performing a meta-analysis of aircraft cockpit design projects.

This paper focuses on how to use those same metrics to generate optimal layout predictions early in system design. We do this using a hybrid algorithm that generates new display layouts with minimized control operation times. The hybrid algorithm is comprised of a seeded genetic algorithm in conjunction with a pattern search algorithm.

II. Background

Most modern aircraft use multi-function displays (MFDs), such as the one illustrated in Fig. 1, with bezel buttons around the edges. These are called "soft buttons" as their functions can be tailored to the particular screen being displayed. This approach is now prevalent in other forms of transportation (such as automobile dashboards and boat navigation/radar systems) and public kiosks (such as automatic teller machines and airport check in stations).

III. Design Challenge

Sound tools for UI design, as a part of human-computer interaction (HCI), has been the focus of a large amount of research. Though actual user testing is regarded as the gold standard of evaluation methods, the time and expense of user testing make it prohibitively difficult to be used exclusively. Practitioners generally use qualitative inspection evaluation methods, such as expert surveys or cognitive walkthroughs in early system development [3]. These use accepted design guidelines, drawn from empirical studies, to examine the ordering, layout, and grouping of the proposed display layouts. These guidelines provide useful heuristics, but they are too general in scope to fully guide the mapping of information and functions to specific operational threads [4],[5]. Furthermore, as a whole, they often yield contradicting advice with no established structure for prioritization. Our proposed metrics quantify the established guidelines on menu structure so that they can be used to objectively evaluate displays and provide a objective function for optimization routines. The guidelines related to the design of specific displays (e.g., button order and style, the data representation, and the form of data integration) must still be accomplished.

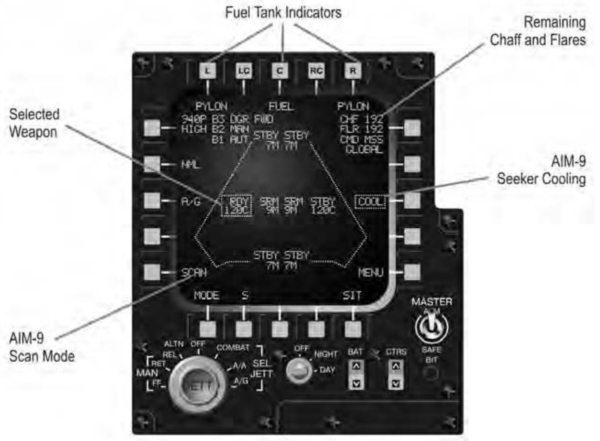

Figure 1. F-15 Programmable armament control system. Example of the use of "soft buttons" on a multi-function display. (U.S. Gov't figure)

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

IV. Model Construction

A display layout can be modeled as an irreducible, nonabsorbing, time homogenous, Markov chain. A Markov chain is usually modeled as a state diagram. For display layout problems, the *nodes* (or vertices) of such a diagram are represented by v_x and represent a unique combination of information, selectable functions, and submenu options to be included in the system. We will refer to each specific combination as a *page* when the context is clear as it is consistent with website terminology and makes the analysis more readable. The *edges* (or arrows) of the graph represent transitions and are defined as node pairs $e_{a,b} = [v_a \ v_b]$. Specifically, each edge is an explicitly defined input from the human that causes a change of display state. For more on how to identify the necessary model data see [1].

Mathematically, a graph can be expressed in several matrix forms. The most intuitive is an *adjacency matrix* of binary numbers. For each *i,j* element of the matrix equal to one, there exists an edge between node *i* and node *j* [6]. This paper uses the term *HCI-defined G* to refer to a graph *G* that has been built from the definitions stated here and is thus a representative model of a valid interface design layout. By design, it will be a connected asymmetric directed graph with a specification-derived node set. The value of doing the modeling effort in this way is that numerical results can be calculated by numerous mature software packages. For our work we used the Matlab® optimization toolbox and the graph theory toolbox [7]. The graphics were generated by the Matlab® Biograph.

Each edge has a probability of transition $\rho_{a,b}$ that is the likelihood of accessing node v_a from node v_b. For example, Fig. 2 depicts a simple graph of displays labeled with their primary task information. In it, $\rho_{\text{AR,ILS}}$=0.03 < $\rho_{\text{CRUISE,ILS}}$=0.20 which quantifies the fact that it is operationally less likely to transition directly to a precision approach after air refueling than it is from cruise flight. A matrix of these probabilities is called the transition probability matrix, or simply transition matrix. For a layout of *n* pages, the transition matrix is an *n* x *n* matrix of $\rho_{a,b}$ values. For existing systems the transition matrix can be formed empirically. For predictive purposes the transition matrix can be formed as follows: first, decompose the system's intended purpose into task sequences, known as operational threads. Cognitive systems engineers have produced a large amount of research on how to best do this. The recommended methods include task analysis and cognitive work analysis [3]. Secondly, total the count of each specific page change during the execution of all identified tasks. Divide each element by the sum of its row. Now the entry of row *a* column *b* is the expected $\rho_{a,b}$. Transition matrices may also be formed through the use of simulations.

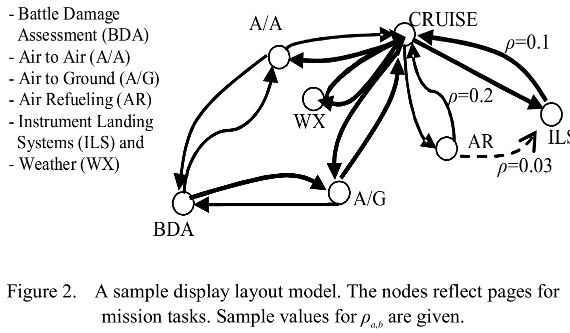

Figure 2. A sample display layout model. The nodes reflect pages for mission tasks. Sample values for $\rho_{a,b}$ are given.

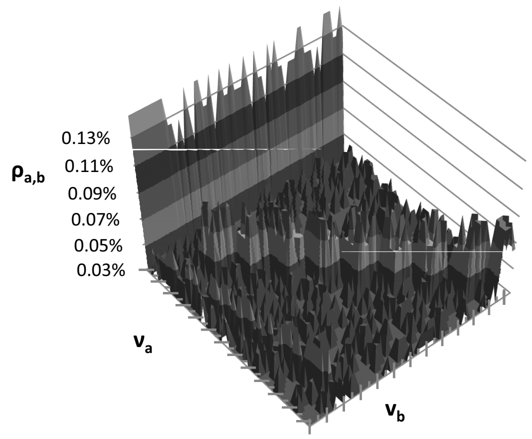

Figure 3. Surface plot of affinity matrix for 63 page example.

Our display design method involves an edge weighting using a special form of the transition matrix. It is referred to in literature as the affinity matrix and represented by ***P*** (capital rho). To form ***P***, we tally all transitions during operational threads as described above. The affinity matrix is the joint probability mass function of the tallied matrix. By definition, the sum of all entries in ***P*** must equal 1.0. As shown in Fig. 3, in general, the transition matrix will not be symmetric. Values on the diagonal represent the number of selectable functions on a page. These are modeled as self loops as they do not invoke a page transition.

V. Design Analysis

Once the proposed layout has been defined as above we are able to find potential solutions. The most valuable contribution of model-based evaluation is the ability to quantitatively score proposed layouts, even before initial prototyping. Wiener, et al, proposed a sum of the shortest path distances between all nodes in a graph, now called the Wiener Index, which provides a simple goodness estimate for Markov chain analysis [8]. Liu proposed that a similar type of scoring should be used to evaluate the search efficiency of menu layouts [9]. In [1] we proposed a novel metric to do this which we termed the HCI Index. This index is the expected mean control operation time of a proposed display layout. To obtain this we use a weighted shortest path calculation based on the Floyd and Warshall algorithm, as coded by Iglin [7]. The paths are weighted by mean system processing speed and expected human choice reaction time; the latter being modeled using the HCI work of Hick, Hyman, Deninger, and Wickens. The "distance" between any two edges, then, is defined as:

$$d_w(v_0, v_k) = \sum_{i=1}^{k} \left(t_i + \left(0.212 + (0.153)\log_2 \left(d^+(v_{i-1}) + 1 \right) \right) \right) \qquad (1)$$

where:

v_0 and v_k are any two arbitrary nodes,

t_i is a time delay constant associated with the i^{th} edge on the (v_0, v_k) minimum path, and

$d^+(v_{i-1})$ is the number of selectable options of the tail node of the i^{th} edge on the (v_0, v_k) minimum path.

This metric is written as $d_{\text{w\_G}}(v_0, v_k)$ when the associated graph G is not clear from the context.

The weighted distance calculation in (1) can be used to derive a metric for evaluating the entire layout with respect to task transition time, which is also called control operation time. We write this as:

$$D_w(G) = \frac{1}{n^2} \sum_{v_a, v_b \in V(G)} d_w(v_a, v_b) \tag{2}$$

This is useful for interface design, but minimizing (2) does not guarantee that a layout is optimized for its mission, or even any specific task. For that, the metric must include input from its expected operational context. Designers can provide this in the form of the affinity matrix described above. This weights the transitions according to those most likely to be accessed in the projected use of the system. We named this the HCI Index and write it as:

$$D_{w.\rho}(G) = \sum_{v_a, v_b \in V(G)} d_w(v_a, v_b) \cdot \rho_{a,b} \tag{3}$$

In [1] we proved that, given a set of proposed layouts, the one with the smallest value of (3) is guaranteed to have the lowest expected control operation time.

VI. Optimal Design Problem Formulation

We are now ready to define the display layout design as an optimization problem. Other optimization approaches begin with an assumed fixed hierarchical structure and then seek to optimize the node placement in that structure [10]. This optimization effort differs in that no structure is assumed a priori and the optimization performs its selection on the set of all possible edges. Assume that designers have determined a proper node set of an HCI-defined G, an operationally-relevant affinity matrix, and a list of time constraints on time critical pages. The following paragraphs express the design problem in optimization terms.

A. Design Variables

Design variables are the things that can be controlled. In a menu design problem, as we have defined it, these are the edges $e_{a,b} = [v_a\ v_b]$ for all a and b from 1 to n.

B. Objective Function

For this problem, the objective function is the expected mean control operation time. The algorithm seeks to minimize *(3),* the HCI Index. Since the HCI Index is an implicit function, it cannot be explicitly expressed in terms of the design variables, we adapt it for use in optimization routines by coding it as a software routine which receives the design variables as inputs and returns the Index value.

C. Constraints

Any real world design solution will be subject to constraints. The solution must be a connected graph (no isolated components) with no stubs (a node with no out edges). Also, any real world problem will have a maximum on the number of programmed transitions from any one page. The solution must also meet constraints on the required accessibility of critical functions. These are modeled by assigning limits to the respective node maximum distance. In graph theory, the *eccentricity* of a node v_a is the maximum of the distances between v_a and all other nodes of the graph. Using (1), the *weighted in-eccentricity* is the maximum $d_w(v_b, v_a)$ for any v_b of G:

$$ecc_w^-(v_a) = \max_{v_b \in V(G)} \{d_w(v_b, v_a)\} \tag{4}$$

Simply stated, this is a measure of the greatest separation, in time, that exists between a particular screen and any part of the rest of the graph. The weighted in-eccentricity of critical nodes v_{crit} must be no greater than some maximum access time b_i [s]. Thus the objective function is constrained by:

$$s.t. \quad ecc_w^-((v_{crit})_i) \le b_i \quad i : 1 \text{ to } m$$

where m is the number of constrained nodes.

Since this is what developers truly want to control for critical information, is a better way to identify those constraints than traditional methods.

VII. Problem Characterization

A. Existence, uniqueness, and convexity

The display layout problem is explored in detail in [2]. Here, we assert without proof, that the existence of an optimal solution is guaranteed except in the overly constrained form, there is no guarantee on solution uniqueness, and the problem is non-convex.

B. Computational Complexity

For space reasons we also assert without proof, that the solution space is of size: $2^{n^2} - 2(n-1)$. This means that even modest problems have a very large feasible set. For example, a problem involving n=63 nodes, such as the example for a multi-role aircraft discussed in [1], has more than $10^{1,000}$ feasible designs. This quickly eliminates methods of exhaustive search as viable approaches.

C. Matching to the Proper Tool

In addition to being complex, both the objective function and the constraint functions are nonlinear. That is, neither the property of additivity nor homogeneity are satisfied. Also, the design variables are discrete. This precludes the use of traditional optimality methods and even numerical search methods (e.g., integer programming and sequential linearization methods) which require gradients or gradient estimations [11]. This type of specialized problem can be effectively addressed by heuristic methods such as simulated annealing, genetic algorithms, or tabu search.

We evaluated these optimization techniques and determined that this problem is best accommodated by the attributes of the genetic algorithm (GA). This meta-heuristic is based on concepts of genetic theory. An entire population of potential solutions is characterized by design variables in the same way that chromosomes characterize an organism. The GA allows us to perform global parallel searches in a highly discontinuous solution space and resist premature convergence to a local minimum [12]. Also, we are able to tune the initialization and selection criteria to exploit a priori knowledge of partial solution clustering. The work by Matsui and Yamada confirmed the superior performance of a properly tuned GA for this type of problem [13].

To implement a GA, one begins with a population of complete designs. Then:

1. Test the fitness of each member using the fitness function.
2. Form new generations: Select from competing members for reproduction based on fitness values. Form new designs for the next generation by performing crossover and mutation.
3. Perform migration: choose some from similar groupings to move to a dissimilar grouping.
4. Repeat from Step 1 until the stopping criteria is satisfied [11].

This competition within the population has enabled an aggregation of good solution subsets that were not deterministically obvious. A genetic algorithm must be designed, or tuned, for a particular problem. We have done the following tailored parameterizations.

VIII. Algorithm Tuning

A. *Design Variable Representation*

A proper representation maps the state space of the variables used in the GA population to the solution space of the actual problem. The most intuitive representation in this problem is a nxn length string of binary numbers representing the elements of an adjacency matrix, but there are numerous other graphs representations in use such as: the node list, an edge matrix, and a star representation array [14]. All were examined to see which contributed to improved computational efficiency. Michalewicz and Fogel have identified key qualities of a good representation for GA problems. They state that a representation should:

1. Be a bijective mapping (that is, one to one and onto) to the real world variable set
2. Maintain the link of parents to offspring; usually through a graduated range
3. Be segment-able in that components of the whole are independently useful
4. Always, or at least frequently, produce feasible solutions [11]

These are important for the performance of the algorithm. As it can be seen, the adjacency matrix only possesses the first quality. In fact, none of the traditional mathematical forms provide all the desired qualities.

In the pursuit of a suitable chromosomal form, we represent the design variables of the display layout by an n-length vector of real numbers between 1 and the edge count constraint, as defined in the specifications. Each of these values in the sequence represent the number of edges emanating from the corresponding node. The connecting node list is determined by using the affinity matrix as a look up table. For each node, the corresponding row of the affinity matrix is rank ordered by transition probability score. Edges are then assigned up to the stated number. This is not a natural representation, but it is superior to all traditional graph representations in that it possesses all of the desired qualities of a GA population. Furthermore, it uses contextual knowledge to guide the connectivity of the graph toward likely quality designs. The limitation on this representation is that it is dependent on an accurate task analysis to produce quality transition probabilities. To make this process more robust, an additional variable is added to make it an $(n+1)$-length vector. This variable determines the amount of random noise introduced to the rank ordering of the affinity matrix elements. Thus, the algorithm is presented with a sampling from across the entire range of feasible solutions.

Since this is a custom population type, we had to modify the Matlab® creation function, mutation function, and crossover function. In addition, we had to create transformational routines that convert between the vector representation and the adjacency matrix form used by the shortest path algorithm. During this conversion we round the real value outputs of the GA into integer solution values.

B. *Initial Population Creation*

Heuristic methods are not very computationally efficient. As studied by [12], the genetic algorithm will perform better if "seeded" by feasible solutions with wide variation. To achieve this, we initially provided the GA with graphs generated using Prim's algorithm. The algorithm guarantees minimum spanning trees [15]. We found that this method did not produce an initial population with enough variation to prevent premature convergence to a local minimum. We instead provide the genetic algorithm with an initial population of pseudo-randomly generated spanning trees.

In addition, to further counteract premature convergence, two sub-populations are simultaneously evolved with full independence except at periodic migration points. Every 20 generations a mutual migration of 10% of the population occurs. These steps have successfully made the algorithm more resilient to the trap of sub-optimal local minima.

C. *Fitness Function and Constraint Handling*

The fitness function determines the solution space landscape. A good fitness function should yield some sort of correlation to the quality of the solution, that is, a score should be proportional to the distance from the optimal solution. The objective function described in (*3*) was found to do this without modification.

A primary difficulty of implementing a GA is constraint handling. Our problem requires several added procedures to account for all of the constraints discussed above. First, all variables are controlled to be between 1.0 and the edge count constraint. This means that every solution will yield at least a connected graph. Secondly, critical node constraints are evaluated as part of the scoring algorithm. Infeasible solutions are addressed by allowing an adaptive penalty function where the size of the penalty is a function of how frequently the algorithm ventures into the infeasible space. This allows transitions through infeasible space to find other optima; especially early in the search time. The penalty grows linearly as the number of infeasible solutions increase in order to force the population back into the feasible domain.

D. *Selection Procedure*

The selection procedure determines how a member's fitness function score is used to determine both its survival and chance to spawn future generations. We use the roulette technique. This is analogous to assigning members an area on a roulette wheel that is proportional to their fitness score. This works without scaling because our fitness function algorithm returns a bounded score for all feasible solutions. In addition, we implemented an *elitist operator* in the selection process. We set this at 0.1 so the highest 10% of both populations are automatically carried to the next generation. This ensures that the best fitness level is monotonically increasing with each generation.

E. *Variation Operators*

In addition to migration and spawning, other variation operators are used to evolve the populations. We will use both mutation and crossover.

Mutation is the process of making limited random changes to fit members. We implement mutation using an adaptive feasible function. This means that the changes are made with a weighting of the success of those changes in previous generations. This makes the mutation rate self-adaptive.

Crossover is a type of multi-parent reproduction. Children are reproduced by a recombination of parts from fit members. We implement it using a heuristic function that generates children nearest the parent with the highest fitness function, but in a direction opposite from the parent with the lowest fitness function. Offspring that exceed the fitness of all parents are kept. Of those that do not, 1/n of the offspring are kept.

F. *Final Local Search*

The genetic algorithm was allowed to run for n*120 generations or until there has been a <0.1% improvement for 0.1(n*120) generations. These values were determined by observation of genetic algorithms on similar problems [11].

We implemented a two-phase hybrid algorithm in that, once the GA reached a termination criterion, the results are further analyzed by a local search algorithm. There is a great deal of work on how to best combine the explorative strength of the GA with the exploitative power of various local search algorithms [12]. We choose a pattern search algorithm because it is able to search a local "topography" without the need for a

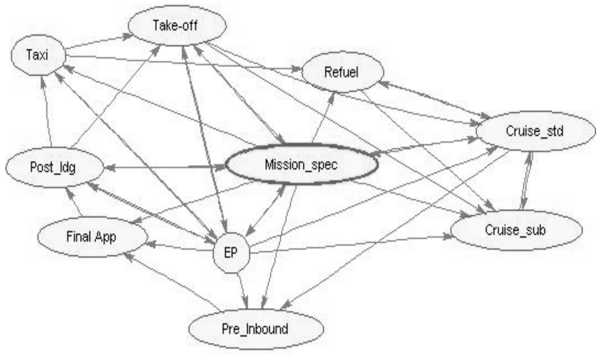

Figure 4. A ten page problem. Manual design result: 673. The GA design layout contained multiple additional edges and scored 618.

derivative or Hessian. With this approach we are able to guarantee at least local optimality.

IX. PERFORMANCE DESIGN METHOD

As an initial test of proposed method, we developed design specifications for two simple node display layout problem; one four and one five pages. We then formed the transition probability matrices in accordance with the described method. For purposes of comparison, we "manually" predicted an optimal display layout by connecting the nodes with the highest affinity. We then analyzed the specification data using our algorithm. Finally, we executed a computer-based exhaustive search to find the global optimum.

In the four page example, the GA found what turned out to be the global optimum in less than three generations; though of course it continued to search until the termination criteria were met. This solution turned out to be better than the one proposed by manual design by using one less edge.

In the five page example, the GA was consistently able to find the global optimum in less than five generations. This solution turned out to be better than the implied optimal by using one less edge. This solution turned out to be better than the one proposed by manual design by using multiple additional edges. Note that even these seemingly simple problems required a computer to run several hours to perform the exhaustive search. The results are the first two sets of data points on Fig. 7.

To study the performance of the process with more complex cases, we first developed a 10 page MFD example. The algorithm was able to find a solution, shown in Fig. 4, that was 9.0% better than the one proposed by manual design.

We then revisited the 63 node problem that was originally presented in [1]. In that paper, three rational yet ad-hoc approaches were used to configure the pages of a fighter aircraft MFD. All three approaches have been used by designers of real world products. Figure 3 is a surface plot of the affinity matrix used in this problem. In [1], we scored all of the approaches with the HCI Index. The best design is shown in Fig. 5. We then analyzed the same specification data using our seeded hybrid GA. It produced a design that achieved an 18.0% better performance prediction. A graph of the resultant solution

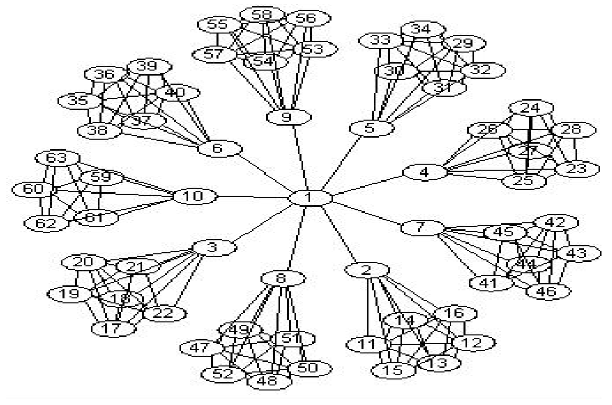

Figure 5. The best ad-hoc layout

is shown in Fig. 6. All of the results are shown in Fig. 7. In addition, two results obtained from actual human subjects tests on displays of 41 and 43 pages are shown.

X. Conclusion

As it has been shown, the seeded hybrid genetic algorithm with human performance modeling is very useful for early user interface design. We have shown the necessary adaptations so that genetic algorithms can solve this problem type and even take advantage of problem-specific information. We can rapidly predict optimum designs directly from design specifications to facilitate rapid display layout development. This is widely applicable to cockpit, control station, public kiosk, and electronic device design.

Acknowledgment/Disclaimer

This work was supported in part by the 711th Human Performance Wing, Human Effectiveness Division.

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the U.S. Air Force.

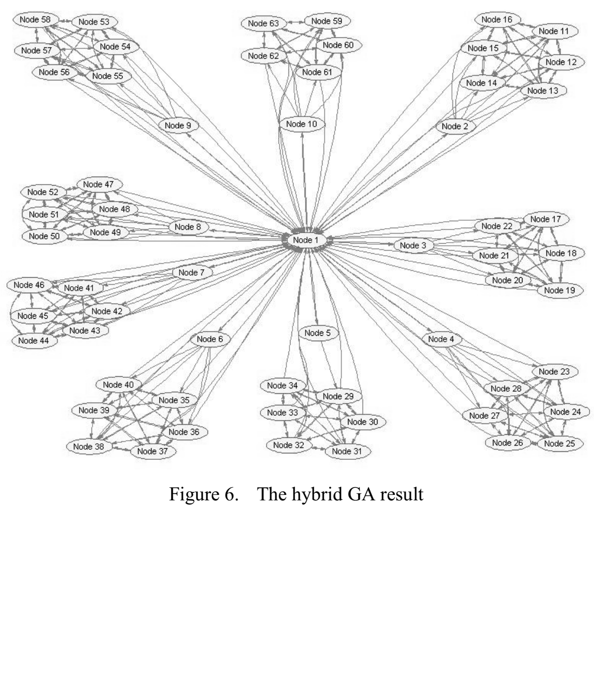

Figure 6. The hybrid GA result

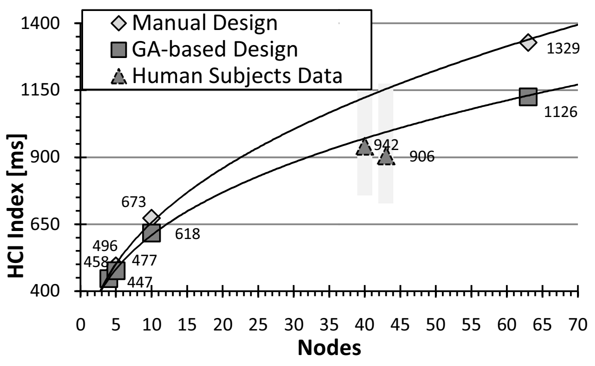

Figure 7. Comparison of manual vs. hybrid-GA design

References

[1] N. Hardman, J. Colombi, D. Jacques, and R. Hill. 2008. Improved user interface design through graph-theoretic modeling. *Proceedings of IEEE International Conference on Distributed Human-Machine Systems*.

[2] N. Hardman, J. Colombi, D. Jacques, R. Hill, and J. Miller. 2009. The validation of user interface design by Markov chain modeling and seeded genetic algorthms. IEEE SMC Part A Transactions (submitted).

[3] A. Bisantz, C. Burns. Applications of Cognitive Work Analysis, CRC Press. 2009

[4] D. Kieras. Model-Based Evaluation. Chap. 60 of Human-computer Interaction Handbook. Taylor & Francis Group. Ed.s A. Sears & J. Jacko. 2004.

[5] B. Shneiderman and C. Plaisant, "Designing the user interface," 4th ed. Boston: Addison-Wesley, 2005.

[6] D. B. West, "Introduction to graph theory," 2nd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2001.

[7] S. Iglin. (2007, Aug) Graph theory toolbox posted in Matlab Central.. Available: http://www.mathworks.com/ matlabcentral/ .

[8] M. Puterman. Markov Decision Processes: Discrete Stochastic dynamic Programming. Wiley & Sons, Inc. 1994.

[9] B. Liu. Applying Models of Visual Search to menu design. Int'l Journal of Human-Computer Studies. 56, 307-330. 2002.

[10] G. Francis. Aircraft multifunction display and control systems: A new quantitative human factors design model for organizing functions and display contents. USAARL Report No. 97-18. 1997.

[11] Z. Michalewixz & D. Fogel. How to Solve it: Modern Heuristics. Springer. 1998.

[12] J. Payne and M. Eppstein. A Hybrid Genetic Algorithm with Pattern Search for Finding Heavy Atoms in Protein Crystals. *Genetic and Evolutionary Computation Conference.* pp. 169-178. (2005).

[13] S. Matsui & S. Yamada. "Optimizing Hierarchical Menus by Genetic Algorithm and Simulated Annealing. *Genetic and Evolutionary Computation Conference*. 2008.

[14] R. Ahuha, T. Magnanti, J. Orlin. Network Flows: Theory, Algorithms, and Applicaitons. Prentice Hall, New Jersey, 1993.

[15] W. Kocay & D. Kreher Graphs, Algorithms, and Optimization. Chapman & Hall, CRC Press. 2000.