User Interface Design with Combinatorial Optimization

Antti Oulasvirta, Aalto University

Optimization methods have revolutionized almost every field of engineering design, so why not user interface design? The author reviews progress and challenges in model-driven UI optimization, in which an optimizer utilizes predictive models of human perception, behavior, and experience to anticipate users' responses to computer-generated designs.

Optimization methods in user interface (UI) design is a long-standing topic in human–computer interaction (HCI) research. Currently, user-centered design is largely focused on human creativity, sensemaking, empathy, and creation of meaning,[1] but optimization methods have been explored as supplemental ways to help speed up the design cycle and improve design quality (for more information about computational UI design, see the sidebar). Unlike any other design method in HCI, optimization methods offer a greater-than-zero chance of finding an optimal design, and some exact methods even offer mathematical guarantees for solutions.[2] More importantly, if the design task and its assumptions are appropriate, the outcome represents the best achievable design.

Optimization methods do not stop at searching for a single best design; they can also be used to explore the design space for alternative ideas. In addition, optimization could promote a change in design culture by encouraging the explication, scrutiny, and accumulation of design knowledge that has tended to be tacit. However, progress has been slow. Scarcely 15 years ago, keyboard layout was the prime—and virtually only—application of optimization methods that had empirically demonstrated benefits to end users.

Model-based UI optimization refers to the use of combinatorial optimization methods to solve a UI design

APPROACHES TO COMPUTATIONAL USER INTERFACE DESIGN

The concept of using computational methods instead of manual exploration to generate good designs was discovered more or less independently in computer science, cognitive science, and human–computer interaction (HCI). The following are different approaches to computational user interface (UI) design.

TYPEWRITER LAYOUT AS A QUADRATIC ASSIGNMENT PROBLEM

Rainer Burkard and his colleagues worked on the optimization of typewriter layouts as early as 1977,[1] formulating it as a quadratic assignment problem (QAP). This finding was valuable because the problem could now be solved through efficient solvers known for QAP. However, the estimates of a typist's time costs were not based on empirical data, which made the task easier to solve and the outcomes less valuable. Realistic input data for a typist's motor performance was added a decade later.[2]

COGNITIVE MODELING AND OPTIMAL PERFORMANCE ENGINEERING

Stuart Card and his colleagues proposed the first simulations of a user for HCI in 1983 (GOMS).[3] Instead of guesswork or expensive studies, a designer could evaluate an interface by simulating how users perceive, think, and act when completing tasks. Subsequent models (such as ACT-R) extended this modeling to consider factors such as errors and learning. However, the models became difficult to use and extend. To aid practitioners, mathematical simplifications (such as KLM and GLEAN) and interactive modeling environments (like CogTool and Distract-R) were developed, but these were not combined with algorithms that could generate designs.

A decade later, Donald L. Fisher proposed that human factors researchers should not be satisfied with a good solution but seek the optimal solution using appropriate methods.[4] However, because little advice was given on central technical enablers such as optimization, applications were limited to parameter optimizations and simple mappings.

UI DESCRIPTION LANGUAGES

Software engineers have studied UI description languages, formal abstractions of the UI and its properties, operation logic, and relationships to other parts of the system.[5] These can be used to compile UIs in different languages and port them across platforms. However, because they lack a rich notion of the user, they are limited to transformations instead of generation for given user-related objectives.

OPTIMIZATION USING DESIGN HEURISTICS

Design heuristics can be used as objectives and constraints to drive UI optimization. Peter O'Donovan and his colleagues formulated an energy minimization approach to the design of page layouts using heuristics like alignment, visual importance, white space, and balance.[6] This approach improves the visual appeal of optimized layouts. However, heuristics offer no link to end-user-related objectives such as task completion time or aesthetic perception. Resolving conflicts among multiple conflicting heuristics is a recognized issue.

References

1. R.E. Burkard and J. Offermann, "Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme," *Zeitschrift für Operations Research*, vol. 21, no. 4, 1977, pp. B121–B132 (in German).
2. L. Light and P. Anderson, "Designing Better Keyboards via Simulated Annealing," RIT Scholar Works, 1993; scholarworks.rit.edu/cgi/viewcontent.cgi?article=1729&context=article.
3. S.K. Card, A. Newell, and T.P. Moran, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, 1983.
4. D.L. Fisher, "Optimal Performance Engineering: Good, Better, Best," *Human Factors: J. Human Factors and Ergonomics Soc.*, vol. 35, no. 1, 1993, pp. 115–139.
5. J. Guerrero-Garcia et al., "A Theoretical Survey of User Interface Description Languages: Preliminary Results," *Proc. 2009 Latin American Web Congress* (LA-WEB 09), 2009, pp. 36–43.
6. P. O'Donovan, A. Agarwala, and A. Hertzmann, "Learning Layouts for Single-page Graphic Designs," *IEEE Trans. Visualization and Computer Graphics*, vol. 20, no. 8, 2014, pp. 1200–1213.

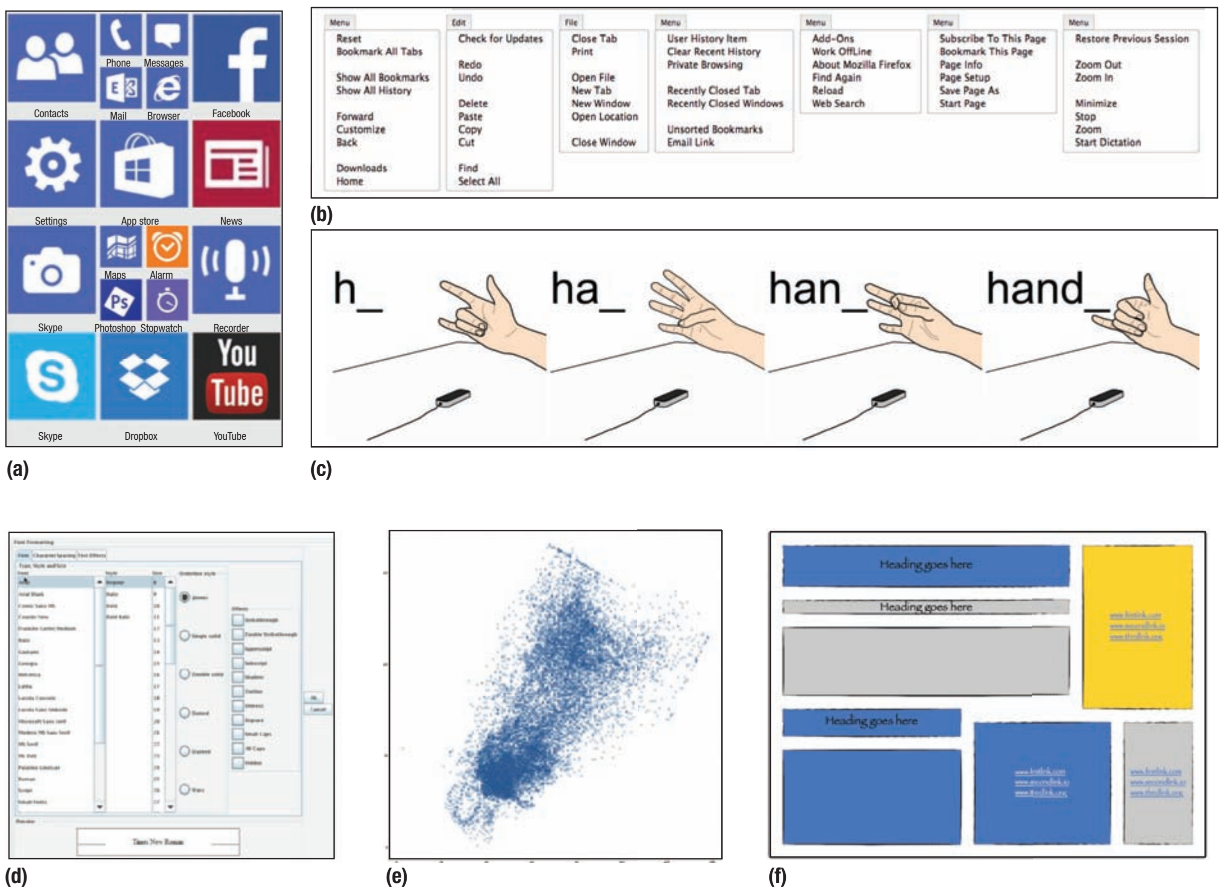

FIGURE 1. Examples of designs generated with recent model-based approaches to user interface (UI) optimization: (a) application menu design, where the positions, colors, and sizes of icons are optimized for visual search, motor performance, color harmony, and minimum clutter; (b) menu system design, where a hierarchy of items is organized into a menu optimized for visual search time and consistency with other menus; (c) gestural input design, where commands are mapped to hand gestures to optimize for performance, individuation of fingers, and learnability; (d) widget GUI design, where widget groups, types, and positions are decided based on users' motor capabilities; (e) scatterplot design, where aspect ratio, marker sizes, and transparency are optimized for maximum performance in tasks like correlation estimation by using perceptual models of brightness, contrast, and unit perception; and (f) web layout design, where the elements of a page are positioned, colored, and sized to optimize visual search time, motor performance, color harmony, and minimum clutter.

problem formulated as a search problem by using predictive models of human behavior and experience. The significant benefit of model-based UI optimization over previous approaches is that it can find and propose UI designs that are both visually appealing and verifiably usable. With model-based UI optimization, a webpage design can be automatically generated so that users regard its items and colors as aesthetically pleasing,[3] and a programmer can task an optimizer to design a full menu system with hundreds of commands—complete with hierarchy, groups, and shortcuts—simply by listing its command names.[4]

MODEL-BASED UI OPTIMIZATION

Like a civil engineer whose objective is building a bridge with a certain capacity and tolerance, a designer can create a UI by defining how easily users should find their targets and how they should perceive it aesthetically. Technically, this is based on the formulation of a design task mathematically using predictive models of human behavior as objectives. Figure 1 shows examples of designs generated with recent model-based approaches to UI optimization, including application menu design and menu system design,[4] gestural input design,[5] widget GUI design,[6] scatterplot design[7], and web layout design.[4] Figure 2 shows an overview of model-based UI optimization from a design team's perspective.

UI DESIGN WITH AN OPTIMIZER IN THE LOOP

Tools might encourage or enforce user interfaces that were highly usable, rather than today's stance that tools should be neutral....[8]

A starting point for re-envisioning UI design with an optimizer in the loop is the realization that the standard "fire and forget" paradigm of optimization methods is irreconcilable with design practice. A designer cannot be asked to provide all inputs to an optimizer within a decimal point and come back later for an answer. Designers constantly change the problem definition and the design space,[9] and mix design sketching and evaluation with reflection and sensemaking, drawing from tacit knowledge. How might design tasks be defined for a computer, and how could the optimizer positively encourage the designer to adopt better designs without choking creativity?[8]

To be used at all, optimization requires effort to define or redefine a task and assumptions. A considerable amount of information is needed to design a menu system, for example.[4] A key concern, therefore, is which aspects of such work can be automated and which should be left to the designer.

Figure 3 plots the automation continuum, exposing a tradeoff between the designer's and the optimizer's objectives. The design space, objectives, and constraints that an optimizer works with should be aligned with—or at least not conflict with—the designer's view of them. However, the designer might want to change this data often, especially in early design stages, which takes effort and could break the design flow. Though a design tool could hide some of the complexity of controlling an optimizer by inferring inputs, its outputs could easily become misaligned with the designer's objectives.

MenuOptimizer and SketchPlorer, two initial optimizer concepts, explore the two extremes of this continuum.

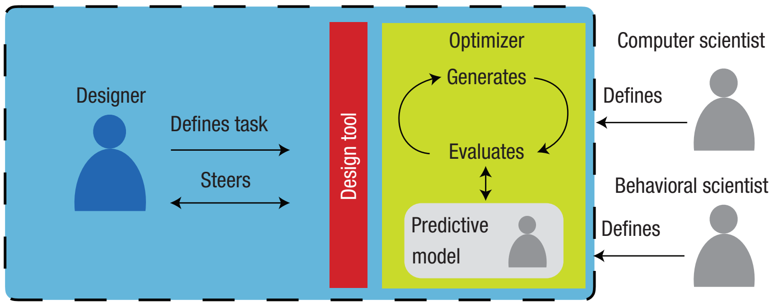

FIGURE 2. Model-based UI optimization from a designer's perspective. Predictive models (mathematical or simulations) are used in an optimizer to evaluate generated designs against designer-supplied objectives. The designer can steer and redefine the tasks intermittently as results stream in. Initialization might require contributions from a computer scientist to define the task and an optimizer, as well as from a behavioral scientist to supply predictive models.

MenuOptimizer

The full-control MenuOptimizer combines many support functions and controls to assist a developer using Qt Designer—a widely used integrated development environment (IDE).[4] A developer designs a menu as usual, but as command names are listed, MenuOptimizer proposes full menu systems on a Pareto plot, including hierarchical assignment of menu items with their various groupings into menus and shortcuts. While the designer edits the menus, MenuOptimizer also suggests local improvements, such as "Move this item here to improve user performance by 6 percent." To this end, its objective function is based on a menu-selection-performance model and a heuristic for consistency, which tries to keep the design close to those of existing applications. An important goal in the development of the system was to make inputs to the optimizer fully visible but seed it with meaningful defaults to save editing time. The designer can change design objectives with sliders, change assumptions about users and their tasks via interactive bar plots, change scores for consistency by means of a table, and so on.

We tested this optimizer with computer science students who were asked to design a menu system for a software design scenario. The students were able to design menus they found satisfactory with significantly less effort (38 percent fewer clicks) than when they did not use the optimizer. They reported appreciating the optimizer's suggested improvements to the menus. However, they used very few aspects of the functionality for optimizer control, and they complained that the controls were effortful and overwhelming. Objectives, for example, were rarely changed.

SketchPlorer

To better support an exploratory approach to design, we created SketchPlorer, a minimum-effort (or maximum-automation) concept.[3] In contrast to MenuOptimizer, it does not require any additional input from the designer other than indicating which

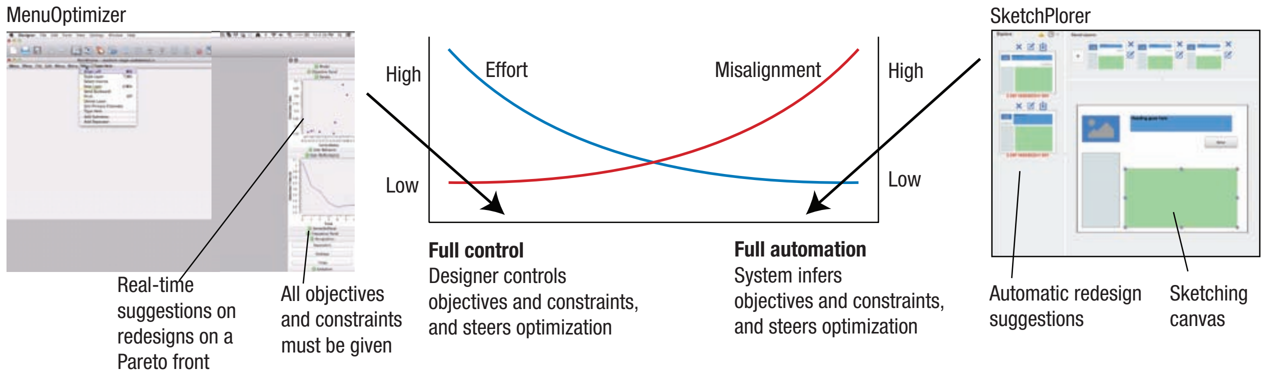
MenuOptimizer
Real-time suggestions on redesigns on a Pareto front
All objectives and constraints must be given
High
Low
Effort
Misalignment
High
Low
Full control
Designer controls objectives and constraints, and steers optimization
Full automation
System infers objectives and constraints, and steers optimization
SketchPlorer
Automatic redesign suggestions
Sketching canvas

FIGURE 3. Automation continuum. Interactive design tools using optimizers must automate some aspects of design work but leave others to the designer. This introduces a tradeoff between the designer's and the optimizer's objectives (center). The full-control concept MenuOptimizer (left) requires objectives and constraints to be specified upon any change in a designer's task. SketchPlorer (right) follows a minimum-effort optimization approach: the computer infers some aspects of the design task and suggests multiple redesigns assuming different design objectives.

elements are more important than others. It infers the design space from the way the designer places elements on a canvas and uses that conclusion to search the design space.

Because the designer's objectives are unknown, the optimizer produces numerous suggestions, some of which might match poorly. However, because these are presented directly on a side panel next to the canvas, they can easily be examined and those that are irrelevant can be ignored. To rapidly generate suggestions, SketchPlorer uses a resource-sensitive optimization strategy that quickly proposes solutions using precomputed designs as starting points, or explores the candidate space more broadly if more time is available.

We tested Sketchplorer with 10 professional designers who were asked to design the wireframe sketch of a blog page. Eight of the designers used the optimizer's redesign suggestions and said they could sketch ideas quickly with this feature. They also indicated that they appreciated the exploration functionality, with one participant stating that he liked the way it at times "turned [his] ideas upside-down."

While not a conclusive result, our experience with SketchPlorer shows the benefits of inferring a designer's task and relying on his or her ability to quickly recognize good and bad designs from a larger set of visualized options, and to steer optimization with choices from such sets.

EVALUATING DESIGNS USING MODELS OF HUMAN BEHAVIOR AND EXPERIENCE

The adoption of optimization methods critically depends on the formulation of a design problem as a search problem and a definition of a "good" design. In this regard, UI optimization is an application of engineering optimization for discrete design problems,[10] where a task consists of a finite set of candidate designs, an objective function, and constraints. Two challenges stand out.

First, defining a search space (or design space) means defining the design variables that make up the design space. A design task is defined as the task of finding the best combination of design decisions. For example, keyboard design is defined as the task of assigning letters to button slots to maximize typing speed. Given *n* letters and *n* keyslots, the goal is to minimize the average time cost of selecting letter *l* after *k*, weighted by the probability of that letter transition in a given language. In the simplest version of the keyboard layout design problem, the search space is on the order of 4×10^{26}. A menu hierarchy, on the other hand, can be organized in about $(2n)!$ ways. For 50 items, the size of the search space is $100! \approx 10^{158}$. Constraints can be added to define feasible solutions and decrease the search space, such as "elements cannot overlap" or "elements must have a maximum size."

A decade ago, this field was limited to assignment problems (such as the quadratic assignment problem discussed in the sidebar). These problems can now be solved to a size of 10^{38} via known, efficient exact methods like branch and bound.[2] Beyond keyboards, recent work has looked at expanding assignment formulations to other design tasks involving placing elements on a surface, such as menus[4] and GUIs.[3,6] The assignment problem has also turned out to be a viable approach to some gesture design problems where hand postures or transitions are mapped to commands.[5]

Beyond assignment tasks, our group is researching scheduling problems (for example, in notification systems) and packing problems (for example, in designs of pervasive displays for public spaces). We are working to formulate the problem of selecting the functionality for an application in early stage design as an integer programming problem. Given a set of *N* possible functions that could be supported, we want to determine which subset strikes the best balance among usability, usefulness, learnability, and commercial considerations. Many other design tasks remain to be defined in a similar way and are linked to known problems in computer science.

Second, defining a meaningful objective function remains a significant challenge. With few exceptions, these functions consist of multiple subobjectives. Figure 4 illustrates an example of single- and multiple-objective results using the SketchPlorer optimizer.[3] Note that there are no surface-level heuristics used here; instead, we used simulations and mathematical models of how users might perceive, experience, and react to a layout.

It is preferable to use models of human behavior and experience that directly predict design objectives (for example, usability or aesthetics) or correlate with them instead of heuristics. A model is basically a function that maps a design in the design space to an objective value relating to end users. The goal is for the model to predict user performance and experience. It should also permit rapid execution in code.

Typical model types are regression models like Search-Decision-Pointing (SDP)[4] and stochastic models like the Kieras–Hornof visual search model,[3] but there are also neural simulations, such as saliency maps. The challenge in this sort of modeling is that it should predict real behavioral tendencies for any design in the design space. A model must therefore reflect some relatively universal psychological tendencies. As HCI research has proposed only a few general laws of this type,[1] we need to turn to behavioral science, social science, neuroscience, and biomechanics.

Our approach is to relax the modeling requirements: instead of trying to build a comprehensive user model or a set of design heuristics, each case recruits models from a pool of basic models based on which objectives are most important. The studies mentioned above draw from mathematical- and simulation-based models of human visual perception, attention, memory, learning, motor control, biomechanics, and choice.

However, in cases where models are incomplete, invalid, or unavailable, objective functions can also express if-then rules such as design heuristics (for example, "The state of the system must be visible to users"). However, using multiple heuristics introduces the problem of weighing multiple independent subobjectives in an objective function against each other. This can be done empirically, algorithmically using tuning methods, or manually.

After the objective function is defined, familiar combinatorial optimization methods such as branch-and-bound[2] or black-box methods (such as ant colony optimization[4]) can search this set much more efficiently than a human designer can. The choice depends on the designer's needs. Exact methods offer mathematical guarantees for solutions but require analysis of the objective function for revealing simplifications and links to known tasks in the optimization literature. Black-box methods, in contrast, can attack any design problem but require empirical tuning of the parameters and offer no guarantees. Among the black-box methods used for interface design are simulated annealing, genetic algorithms, and ant colony optimization.

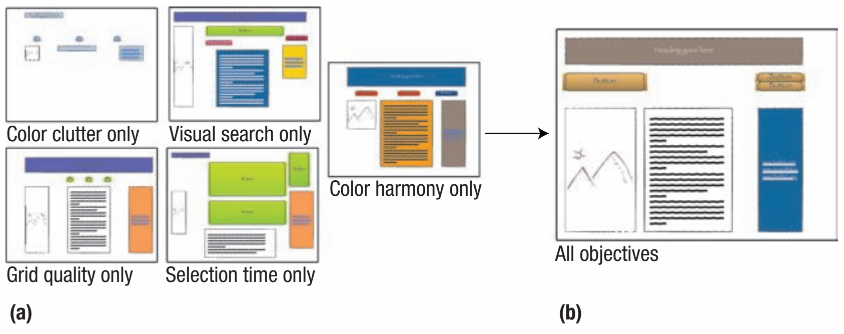

FIGURE 4. Results from (a) single- and (b) multiple-objective optimization in webpage design using the SketchPlorer optimizer. A meaningful webpage layout can only be designed when all objectives are considered together.

BEYOND KEYBOARDS

Perhaps the most significant obstacle to progress in this space has been modeling, not algorithms. Thanks to an increasing pool of models, we can now optimize some of the most common types of UIs.

Menu design is a classic topic in HCI. It was mathematically formulated in the 1980s, but is now an instance of the well-known assignment problem wherein commands are assigned to a vertical array to minimize selection time and maximize familiarity. Extending this to menu systems demands hierarchical assignment and consideration of what makes a hierarchy consistent and how users navigate such hierarchies.[4] On the other hand, interactive graphical layouts must also consider the size and horizontal position of elements, along with their grouping and types. Determining how to do this has extended the domain to widget layouts and webpages.[3,6] Our ongoing work looks at information visualizations, addressing them via perceptual optimization (see Figure 1). Scatterplot design has been formulated as the task of choosing the design parameters (marker size, aspect ratio, and so on) to maximize user performance in tasks such as correlation estimation.[7]

Optimization can also provide solutions to some difficult problems emerging in interface technology. Consider in-air gestures, for example. Although the human hand has numerous degrees of freedom for motion, it is not yet clear how to map these motions to input in a way that is ergonomic, usable, and learnable. We recently developed an optimizer for gesture sets, defining a gesture as a transition from one posture to another and minimizing the movement time, accuracy, and learning costs involved (see Figure 1).[5]

Key questions to explore are whether optimized designs are actually usable, and if so, how they compare with those of human-designed interfaces. Empirical evidence is only emerging and centers on two topics: keyboards and GUIs. The two-thumb keyboard KALQ showed a 34 percent improvement over baseline (Qwerty) after training.[11] The K5 multilingual keyboard (see Figure 1) yielded a 24 percent improvement over Qwerty on a touchscreen device.[12] An optimized mapping of *n* grams to a piano keyboard allowed a trained typist to type at rates comparable to the highest rates achieved with the Qwerty keyboard.[13] For graphical interfaces, a design differentiated by SUPPLE for a motor-impaired user (see Figure 1) showed a 26 percent improvement in speed and a 73 percent decrease in errors from a manufacturer's default.[6] In the realm of application menus, an optimized design was superior in selection time and perceived color harmony to the Windows Phone default design.[3] Although it performed worse in terms of clutter perception, this was predicted by the models underlying the optimized design. This example illustrates a key benefit of model-based approaches: they make empirically verifiable predictions that might steer the improvement of models.

[OPTIMIZATION WILL ENABLE NOVICES TO CREATE GOOD DESIGNS WITHOUT A PROFESSIONAL DESIGNER IN THE LOOP.]

REDESIGNING DESIGN

Perhaps the most startling proposition made here is that essential aspects of design—which has been considered a very nuanced, tacit, and dynamic human activity[1,9]—might be expressed formally and attacked algorithmically, even with a designer in the loop. Algorithms that design UIs can be intuitive and easy to use in design tools, encouraging the adoption of better designs and making designers aware of the involved tradeoffs.

The wider implications of this development can only be speculated. Optimization will enable novices to create good designs without a professional designer in the loop. In the MenuOptimizer study, for example, computer science students were able to design satisfactory menu systems by simply typing in command names and choosing a design from the optimizer. However, designers also stand to benefit from the ability to improve low-level interface design. This might not only improve quality but also free up resources to focus on the "uncomputable" aspects of design—creative problems involving incomplete or contradictory knowledge, a large number of stakeholders who differ in their opinions, and severe constraints.[1]

Perhaps most importantly, for the first time in the history of UI design, researchers can talk about optimization as an engineering discipline, discussing the optimality of a design or its sensitivity to assumptions. What does it mean for this field when a design team can prove that their design is 5 percent better than a competitor's? What if optimizers can show that there is no feasible solution to a problem? In this scenario, computational and engineering sciences will play a much greater role in UI design than ever before.

Many challenges arise. How well can we capture design problems formally? We can now address some classic topics such as assignment and packing, but what are some other common problems in computer science? How far can we push the envelope in modeling human behavior? Although the work described here has been successful with basic sensorymotor models and is expanding to cognitive aspects, there is still a long way to go to address the physical, social, and cultural factors of interaction that are key to emerging topics like AI, the Internet of Things, and social media. Finally, which aspects of design should be automated and which should be left to the designer? Carelessly designed tools will lead to deskilling and avoidance of responsibility, thereby undermining the potential benefits of optimization. If such problems are solved, the wider implications of this approach are intriguing.

ABOUT THE AUTHOR

ANTTI OULASVIRTA is an associate professor at Aalto University. His research interests include modeling and computational design of human–computer interaction. He received a PhD in cognitive science from the University of Helsinki. Contact him at antti.oulasvirta@aalto.fi.

ACKNOWLEDGMENTS

This research was funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement no. 637991) and the Academy of Finland project COMPUTED.

REFERENCES

1. Y. Rogers, H. Sharp, and J. Preece, *Interaction Design: Beyond Human–Computer Interaction*, Wiley, 2011.
2. A. Karrenbauer and A. Oulasvirta, "Improvements to Keyboard Optimization with Integer Programming," *Proc. 27th Ann. ACM Symp. User Interface Software and Technology* (UIST 14), 2014, pp. 621–626.
3. K. Todi, D. Weir, and A. Oulasvirta, "Sketchplore: Sketch and Explore with a Layout Optimizer," *Proc. Designing Interactive Systems* (DIS 16), 2016, pp. 543–555.
4. G. Bailly et al., "MenuOptimizer: Interactive Optimization of Menu Systems," *Proc. 26th Ann. ACM Symp. User Interface Software and Technology* (UIST 13), 2013, pp. 331–342.
5. S. Sridhar et al., "Investigating the Dexterity of Multi-Finger Input for Mid-Air Text Entry," *Proc. 33rd Ann. ACM Conf. Human Factors in Computing Systems* (CHI 15), 2015, pp. 3643–3652.
6. K.Z. Gajos, J.O. Wobbrock, and D.S. Weld, "Improving the Performance of Motor-impaired Users with Automatically Generated, Ability-based Interfaces," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI 08), 2008, pp. 1257–1266.
7. L. Micallef et al., "Towards Perceptual Optimization of the Visual Design of Scatterplots," to be published in *Proc. IEEE Pacific Visualization Symp.* (PacificVis 17), 2017.
8. B. Myers, S.E. Hudson, and R. Pausch, "Past, Present, and Future of User Interface Software Tools," *ACM Trans. Computer-Human Interaction*, vol. 7, no. 1, 2000, pp. 3–28.
9. K. Dorst and N. Cross, "Creativity in the Design Process: Co-evolution of Problem–Solution," *Design Studies*, vol. 22, no. 5, 2001, pp. 425–437.
10. S.S. Rao, *Engineering Optimization: Theory and Practice*, John Wiley & Sons, 2009.
11. A. Oulasvirta et al., "Improving Two-Thumb Text Entry on Touchscreen Devices," *Proc. SIGCHI Conf. Human Factors in Computing Systems* (CHI 13), 2013, pp. 2765–2774.
12. X. Bi, B.A. Smith, and S. Zhai, "Multilingual Touchscreen Keyboard Design and Optimization," *Human–Computer Interaction*, vol. 27, no. 4, 2012, pp. 352–382.
13. A.M. Feit and A. Oulasvirta, "PianoText: Redesigning the Piano Keyboard for Text Entry," *Proc. 2014 Companion Publication on Designing Interactive Systems* (DIS Companion 14), 2014, pp. 129–132.