Computational Support for Functionality Selection in Interaction Design

ANTTI OULASVIRTA, ANNA FEIT, and PERTTU LÄHTEENLAHTI, Aalto University
ANDREAS KARRENBAUER, Max Planck Institute for Informatics

Designing interactive technology entails several objectives, one of which is identifying and selecting appropriate functionality. Given candidate functionalities such as "print," "bookmark," and "share," a designer has to choose which functionalities to include and which to leave out. Such choices critically affect the acceptability, productivity, usability, and experience of the design. However, designers may overlook reasonable designs because there is an exponential number of functionality sets and multiple factors to consider. This article is the first to formally define this problem and propose an algorithmic method to support designers to explore alternative functionality sets in early stage design. Based on interviews of professional designers, we mathematically define the task of identifying functionality sets that strike the best balance among four objectives: usefulness, satisfaction, ease of use, and profitability. We develop an integer linear programming solution that can efficiently solve very large instances (set size over 1,300) on a regular computer. Further, we build on techniques of robust optimization to search for diverse and surprising functionality designs. Empirical results from a controlled study and field deployment are encouraging. Most designers rated computationally created sets to be of the comparable or superior quality than their own. Designers reported gaining better understanding of available functionalities and the design space.

CCS Concepts: • **Human-centered computing** → **User centered design**; Systems and tools for interaction design;

Additional Key Words and Phrases: Functionality selection, interaction design, user-centered design, creativity, optimization methods, integer linear programming, computer-supported design, design tools

ACM Reference format:
Antti Oulasvirta, Anna Feit, Perttu Lähteenlahti, and Andreas Karrenbauer. 2017. Computational Support for Functionality Selection in Interaction Design. *ACM Trans. Comput.-Hum. Interact.* 24, 5, Article 34 (October 2017), 30 pages.
https://doi.org/10.1145/3131608

1 INTRODUCTION

This article advances algorithmic methods and tools for *interaction design*—the practice of designing interactive technology for human use (Bucciarelli 1994; Dix 2009; Goodman et al. 2011; Hallnäs and Redström 2006; Löwgren and Stolterman 2004; Preece et al. 2015; Saffer 2010; Winograd 1997). Interaction design is characterized as "the process that is arranged within existing resource constraints to create, shape, and decide all use-oriented qualities (structural, functional, ethical, and

The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant agreement 637991).
Authors' addresses: A. Oulasvirta, A. Feit, and P. Lähteenlahti, Aalto University, PO Box 11000, 00076 Aalto University, Finland; A. Karrenbauer, Max Planck Institute for Informatics, Campus E1 4, 66123 Saarbruecken, Germany.

ACM 1073-0516/2017/10-ART34 $15.00
https://doi.org/10.1145/3131608

esthetic) of a digital artifact for one or many clients" (Löwgren and Stolterman 2004, p. 5). It is about conceptualizing product ideas and designing their behavior from a user's perspective, as opposed to realizing them in code or bringing them to market. The focus placed on people's experiences and behaviors distinguishes the practice from software engineering and product development. Interaction design grew out of several disciplines—such as human-computer interaction (HCI), industrial design, and graphic design—and has now assumed a central role in the development of information and communication technology. "Interaction designer" and "user experience designer" have become a common professions at major companies, where they are responsible for the design of services, applications, and systems across numerous domains, including work, leisure, games, health, manufacturing, transportation, and business. It is important for HCI as a field to develop methods that increase their success.

However, when it comes to computational approaches, it is hard to name a more challenging domain than interaction design. Design in general is characteristically ill-defined (Cross 2006, 2011; Goodman et al. 2011; Löwgren and Stolterman 2004; Schön 1983). Designers' work is concerned with hard-to-formalize conceptual, structure-based, functional, and esthetic aspects of design (Löwgren and Stolterman 2004). Moreover, a designer continuously engages in refining the objectives and constraints of design (Dorst and Cross 2001). The process of design is iterative, selective, and corrective: at times they explore the design space for satisfactory approaches and then switch to in-depth analysis of the problem to identify a hypothetical best design. They also alternate between a constructive and a critical stance. Criticality allows them to assess which aspects of a solution belong together and which are compatible with background data. Their work also deploys a multiplicity of representations of data and ideas. Designers consult various representations of data from user research—such as user profiles, use cases, storyboards, user requirements, and scenarios. These are used also constructively to envision opportunities or to set and refine objectives. To explore their ideas further, they engage in activities like sketching, wireframing, and rapid prototyping. Finally, they work in teams with different roles and dynamics. It is therefore fair to ask if computational methods, which stereotypically insist on preciseness, can have any room in such a domain. Some scholars go as far as claiming that what makes interaction design complex is through-and-through subjective and experiential: "design complexity emerges within activities of designing, experienced through acts of reflection, decision, and judgment. Therefore, the 'eye' of the practicing designer(s) defines [complexity]" (Goodman et al. 2011, p. 3).

The present article shows encouraging results from few years of research on computational support for one core problem in interaction design: the *functionality selection problem*. The selection of functionality and features is a recognized but under-researched subproblem in interaction design. In functionality selection, given a set of candidate functions such as "Print," "Bookmark," and "Share," a designer has to choose which to include in a design and which to leave out. This tasks precedes and is set apart from user interface design, which assumes a set of functionalities as given. Nevertheless, functionality choices affect how useful, easy to use, novel, profitable, satisfying, and marketable the final design will be. However, there is no obvious "rational" solution to the problem. While on the one hand trying to include desirable and useful functions, a designer must, on the other, avoid "feature creep" that compromises usability and increases costs (Thompson and Norton 2011). Designers are known to make functionality choices as part of other efforts, such as when creating user requirements, generating design candidates, exploring the design space, evaluating candidates, or improving them iteratively (Buxton 2010; Cross 2006; Dorst and Cross 2001; Goodman et al. 2011; MacLean et al. 1991; Preece et al. 2015). This is problematic also for designers because the selection of functionality may be overshadowed by other concerns. In one of the only empirical studies reporting how designers solve this problem, designers reported that although they may know how selection should be done, they may "shortcut" it (Goodman et al. 2011).

Given these issues, our working hypothesis is that computational methods might support designers by providing them alternative perspectives to functionality selection, but this should be implemented in an iterative and controllable manner that can be made compatible with existing practices. We believe that computational methods might be helpful in particular due to the very large number of possibilities that designers might have little time to explore. From a combinatorics perspective, the potential is very clear. If for n functions there are $2^n - 1$ candidate designs, we already have 1,125,899,906,842,623 candidates with only 50 functions, and this is not even a large application. In realistic design projects, sizes of candidate sets may be anything between a ten and a few thousands. Even with multiple constraints in place, such design spaces are too large for manual search.

A major challenge we address here is how to define an optimization task such that it addresses the objectives that designers hold to be important and, yet, fits with their practices. While existing research on product optimization, particularly in product and product-line design and in assortment design, has focused on markets, production, and logistics (see *prior work*), the most important objective in functionality selection is to consider how the design is used and experienced by end users (Goodman et al. 2011). Not only are the objectives different from those in product and assortment optimization, but designers understand their work as being iterative, explorative, and corrective rather than about finding the best possible design. Moreover, characteristic of interaction design is that the objectives can be under-determined and choices subjective and tacit (Cross 2006). This is compounded by the fact that interaction designers must also at times consider business and technical aspects of design. We show how these considerations may be formulated and addressed as an optimization task.

We have developed a novel objective function for functionality optimization based on literature and interviews of professional designers. It addresses several of well-known objectives and considerations: usefulness attributed to the various functionalities, dependencies among them, the ease-of-use with which they can be interacted with, user satisfaction, differences among users, as well as profitability and implementation costs. Ease-of-use, usefulness, and satisfaction form the keystone of "usability" (Nielsen 1994). Usability and usefulness also correlate with technology acceptance (King and He 2006). Profitability and implementation costs, as we here address them, allow expressing not only monetary factors as well as risks but those related to license fees, patents, and so on. More precisely, we formulate the task as choosing a set of functionalities that maximizes "goodness" G. We define G as a linear combination of four objectives: usefulness U, satisfaction S, ease of use E, and profitability P.

$$G = \omega_U U + \omega_S S + \omega_E E + \omega_P P \tag{1}$$

where $\omega_\bullet$ denotes the weight given to an objective. We define each term based on the pre-study and later revisit each term to develop theoretically motivated extensions.

Appreciating the subjectiveness of design choices, along with the presence of uncertainties and missing information, we aimed for an approach that is controllable and robust in the face of ambiguity. Informed by designers, we opted for an explorative approach to optimization as opposed to searching for one "best" option. This approach better acknowledges that "cost" and "benefit" and "usefulness" may be difficult to estimate *a priori*. We assume this approach better fits early stage design, where a designer has not yet converged to a particular solution. In some recent forms of software engineering practices (e.g., agile), functionalities are selected collaboratively using a point-based selection method, and then developed iteratively starting with ones that deliver most user value for the least cost. In Study 2, we investigated how multiple stakeholders of a project can use this as a tool to explore alternative designs.

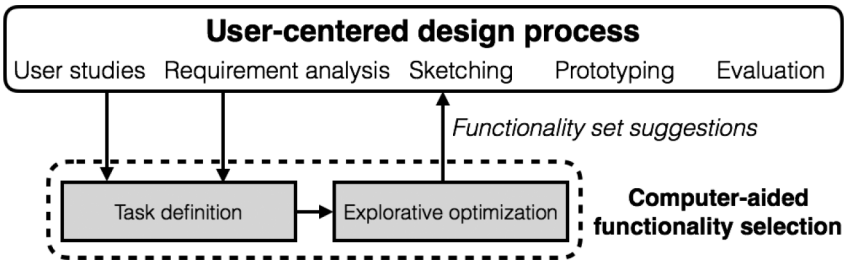

Fig. 1. This article proposes an algorithmic method to support functionality selection in early stage interaction design. A design team lists functionality candidates and fills in a survey to rate the candidates for multiple aspects. This defines an optimization task. An explorative optimization method produces several visualized suggestions on alternative approaches to the problem. The suggestions try to maximally span the space of plausible interpretations of the input data, covering aspects of usability, usefulness, user experience, and business goals.

To support exploration of different designs in the face of uncertainty, we build on Monte-Carlo-based approaches in robust optimization (Beyer and Sendhoff 2007). Instead of just one "optimal design," our approach generates multiple optimization tasks on the basis of expressed uncertainty in input data. The assumption is that the designer who gives input to the optimizer is never completely sure and is also willing to explore the design space beyond the precise input values. The outputs should be useful if the input values given to the optimizer are informative. This solution also allows us to deal with a scenario involving multiple stakeholders with diverging opinions.

The multiple generated optimization tasks are solved one at a time using integer linear programming (ILP). The resulting set of designs is then "mined" to find *diverse designs*. These are presented to the designer visually together with so-called *robust solution* that is the best compromise design among all generated optimization tasks. The idea of presenting several diverse options is a technique called Design Gallery (Marks et al. 1997). The relaxation of input preciseness is critical for practical use. It allows evaluating the objective function without constructing the user interface of the product, generating several varied options for further inquiry quickly, and it allows intuitive control of optimization. This idea has been explored before in interactive design tools in a technique called parallel paths (Terry et al. 2004). However, our approach requires the designer to consider functionality selection in the absence of an interface sketch, an issue we explore empirically in Studies 1 and 2.

Besides the objective function and a designer-centric optimization concept, we present how the problem can be solved efficiently using known ILP techniques in optimization. We develop a fast integer linear solution and present a proof for polynomial time solvability when hard dependencies among functionalities are assumed. Numerical examples show that problems with up to 1,300 functions can be solved to optimality within a second on a regular computer. This set size is in the range of some of the most complex existing computer programs. The contributed formulation enables fast solution to the global optimization task (finding the single best design) and yet makes it feasible to use robust optimization methods for exploring the problem for alternatives as described above. Moreover, the use of exact methods (here: integer programming) allows computing (non-constructive) bounds for obtained solutions. The solver can tell at any time how far the best found design is from the global optimum.

The result is a tool for exploring alternative designs. An overview is given in Figure 1. A designer or a team initializes an optimizer by enumerating all plausible candidate functionalities. This set can be large and include candidates that depend on each other. For example, in the case

of a note-taking application, we identified 106 candidate functionalities in popular apps on the market. Then all functionalities, or a subset of them, are rated for all objectives (here: using Likert-scales), by at least one stakeholder by using a survey tool. This forms a task instance for the optimizer to solve. The objective function receives parameter values that represent the designer's best estimate of the four objectives (one could talk about a "subjective function"). Designers can base their ratings on multiple sources, such as user research data, project objectives, brand considerations, technical experience, and just "gut feeling." They can express their confidence levels in the survey, which is taken into account in the robust optimization part. A central design goal for us has been to make this as light-weight as possible, to better fit with modern fast-paced and agile practices. According to our experience, filling in a survey for 100 functionalities takes about 1–2 hours of work. The ratings serve as input to the explorative optimization scheme, which produces a desired number of alternative functionality sets, ranging between the extremes of a "minimum design" and a "full feature design." Each design is visually presented as a set of functionalities and their associated objective values. Several distinct alternatives and a robust design are visualized for the designer, who can use this output as a basis for ideating, sketching, critique, and discussion.

A clear benefit of the approach is that once the task is defined, an optimizer can be used to address several problems in design:

—Identifying an optimal functionality set for designer-set objectives (point optimum)
—Exploring alternative, surprising designs (design exploration)
—Finding the best compromise among the needs exhibited in multiple stakeholder groups (conflict resolution)
—Customizing products to different user groups (customization)
—Simplifying a complex product (simplification)
—Identifying opportunities for upgrades to an interface (upgrade design)

In this article, we focus on the three first-mentioned uses.

A prime goal in our work has been to avoid disrupting the practices of interaction design. We see that the prime benefit of optimization for a designer is that it can "separate" functionality selection from other decisions with little effort. In our present implementation, inputs to the optimizer are given by designers by means of a questionnaire or spreadsheet. Any relevant stakeholder can complete the survey if the goal is to find good compromises among differing opinions. Respondents can also report confidence estimates, which are used as input for the robust optimization heuristic. These properties make the approach potentially valuable for early stages of design when no single solution is superior to others. It can not only provide new ideas, but help designers by making them aware of aspects that they might not normally think about. However, the challenge is that, for a designer, it requires a shift in mindset: Instead of evaluating functionalities only in the context of a sketch or scenario, one now deals with functionality sets separately. We present two empirical studies to critically examine this aspect.

In the rest of the article, we first review prior work on product and product line optimization and user interface optimization. We then proceed as follows:

(1) We first formulate an objective function based on an interview study of 10 professional designers.
(2) We develop a fast integer linear programming (ILP) solution, explore theoretical extensions, and give examples of its performance in realistic tasks.
(3) We then propose how the optimizer can be deployed in design work, discussing the generation of input in complex projects, explorative optimization to find diverse design ideas, and visualizations of functionality set suggestions for designers.

(4) Finally, we present results from two qualitative evaluations carried out with professional designers.

Results from the empirical evaluations are largely encouraging but raise topics for future work. In our first study, professional designers were invited to a controlled design study in which they were asked to create functionality sets with and without an optimizer in a realistic task setting. They found the optimizer's suggestions to improve the quality of their designs. Most (but not all) designers ranked its suggestions above their own designs. We report on their perceptions of the tool which, although largely positive, raised some issues. In our second study, we deployed the approach at a large telecommunications company. A product team responsible for a (real) major product comprising more than 100 functions filled in input values for an optimizer. They were later interviewed as part of a workshop including several real stakeholders of the product. In summary, they appreciated the potential to explore design ideas—for example, how to simplify products suffering from feature creep. We believe that these results call for more attention to computational methods in interaction design.

2 PRIOR WORK ON ALGORITHMIC METHODS FOR INTERACTION DESIGN

This work contributes to research on computational support for early stage interaction design, focusing especially on the use of optimization methods. Prior work can be divided into two classes. On the one hand, in HCI, the focus of optimization methods research has been almost exclusively on user interface design and development, assuming functionality choices as given. On the other hand, outside HCI in management sciences and operations research, there is extensive work on product and product line design. Yet, although approaches like the Kano model (Kano et al. 1984) are somewhat known among interaction designers, the approach has been labor-intensive and perhaps misaligned with interaction designers' objectives. There is space for reconsideration of interaction designers as a profession with specific needs for optimization methods.

2.1 Combinatorial Optimization Applications to UI Design

In operations research, Rainer Burkard and colleagues worked on the optimization of typewriter layouts as early as 1977, formulating it as a *quadratic assignment problem* (QAP) (Burkard and Offermann 1977). This made it possible to solve the problem by using efficient solvers that were starting to emerge for the class of QAP. However, because the authors' interest focused on the theoretical problem, they used unrealistic estimates of time costs. This made the task easier to solve and the outcomes less valuable. However, research on typewriter optimization paved the road for applications in HCI. Realistic predictive models were proposed more than a decade later (Light and Anderson 1993; Zhai et al. 2002). Recently a technical improvement using integer programming (ILP) was proposed that allowed solving larger problems (Karrenbauer and Oulasvirta 2014). However, applications of ILP have been mostly limited to keyboards.

2.2 Cognitive Models as Objective Functions

In HCI, Card et al. (1983) proposed the first full-fledged simulation of a user, GOMS, for HCI. Instead of guesswork or expensive studies, a designer should evaluate an interface by simulating how users perceive, think, and act when completing tasks. Subsequent models (e.g., ACT-R) predicted not only task completion time but errors and memory load. As the models grew hard to use, to aid practitioners, mathematical simplifications (e.g., KLM and GLEAN) and interactive modeling environments were developed (e.g., CogTool, John et al. (2004)). However, these were not combined with algorithms that could *generate* designs. In human factors, Donald L. Fisher proposed in 1993 that researchers should not be satisfied with any good solution but seek the optimal solution (Fisher

1993). However, since little advice was given on central technical issues such as optimization tasks and methods, applications were limited to parameter optimizations and simple mappings.

2.3 Model-Based Interface Generation and Development

In *model-based interface generation and refinement*, models of design, system, device, user, or task are used directly in the objective function (Eisenstein et al. 2001; Gajos and Weld 2004; Mackinlay 1986; Olsen 1992; Oulasvirta 2017; Paterno 2012; Puerta 1997). Four advances can be distinguished in the literature.

The first is the formal definition of a user interface as the object of optimization. Software engineers have proposed formal abstractions of user interfaces (UIDLs) to describe the interface and its properties, operation logic, and relationships to other parts of the system (Eisenstein et al. 2001; Olsen 1992; Oulasvirta 2017; Paterno 2012; Puerta 1997). Such UIDLs can for example be used to compile interfaces in different languages and to port them across platforms. However, since an UIDL contains little information about the user or human factors in interaction, their use has been limited to transformations and retargetings, instead of complete redesigns.

The second advance was the encoding of *design heuristics* in an objective function. For example, Peter O'Donovan and colleagues have formulated an energy minimization approach to the design of page layouts using heuristics like alignment, visual importance, white space, and balance (O'Donovan et al. 2014). The approach improves the visual appeal of optimized layouts. However, such heuristics may have no empirical link with user-related objectives such as task completion time or esthetics. For example, the amount of white space may or may not linearly predict esthetic experience. Moreover, resolving conflicts among multiple conflicting heuristics is a recognized issue.

The third is the direct use of *predictive models and simulations* from cognitive and behavioral sciences in an objective function, extending the work that started in computer science and human factors. Notably, Krzystzof Gajos and colleagues explored model-based approaches in the design of widget layouts, considering models of motor performance in conjunction with design heuristics (Gajos and Weld 2004). This approach has been extended to the design of menu systems, gestural inputs, information visualization, and GUI layouts (Oulasvirta 2017). Techniques for functionality selection could complement this line of research by allowing a way to go back and forth between interface decisions and functionality decisions.

The fourth is the use of *data-driven methods* to learn or calibrate an objective function, either based on a database of designs or from a designer in the loop using a design tool. Using interactive machine learning, users can steer training through changes to training data (e.g., Fails and Olsen Jr (2003)). If applied to design, a designer could guide the outcome of an optimization process by selecting or providing feedback on training examples. In data-driven approaches to interface generation, statistical models of a design space (e.g., element locations in web blogs) are built from a database of crawled examples. Designers can use a layout to query alternative ideas (Kumar et al. 2012). To our knowledge, no data-driven approach yet addresses the functionality selection problem.

Interactive optimization is the idea of a human designer specifying a task and steering the search process. The key motivation for considering how to involve the human designer in the loop is the realization that the standard "fire and forget" paradigm of optimization methods is a poor fit with design practice. Designers constantly refine the problem definition, mix activities like sketching and evaluation and reflection, and draw heavily from tacit knowledge throughout. Approaches to interactive optimization can be divided according to four dimensions: (1) level of detail in specifying the design task, (2) level of control in steering the search process, (3) support for exploration and refinement of designs, (4) level of proactivity in guiding the designer toward good designs (as determined by objective function). Illustrative examples include DesignScape, which supports

rapid exploration of alternative layouts with an energy minimization scheme (O'Donovan et al. 2015), MenuOptimizer, which provides a high level of control for specifying a task and actively visualizes usability-related metrics to guide the designer to choose better designs (Bailly et al. 2013), and Sketchplorer, which tries to automatically recognize the design task and explores the design for diverse designs to serve as the basis of design exploration (Todi et al. 2016).

2.4 Product Design and Optimization

The design of product offerings has received attention in operations research and management science since the 1970s (Shocker and Srinivasan 1974). *Product design* is determination of the best mix of characteristics for a product or service offering. The approaches adopted address consumer decision-making, including preference, attributes, and choice, as well as company-related costs and profit estimates (Albritton and McMullen 2007). *Product-line selection and product positioning* models focus on selecting and pricing product variants for markets (Krishnan and Ulrich 2001). Their focus is on the recognition and realization of market opportunity, with models covering aspects of product definition, product design, process design, and supply-chain design (Jiao et al. 2007). *Assortment design* is the problem of selecting products in retail, displaying them, and replenishing stores (Kök and Fisher 2007). A subset of the products should be chosen and displayed on shelves, in consideration of customer demands, choice, and product availability.

The *Kano model* is a product-design model that takes into account consumer preferences for features of products (Kano et al. 1984). It distinguishes among various types of product qualities, each with differential effects on satisfaction. It outlines several relationships that a quality can have with satisfaction, such as "must have," or "one dimensional," or "attraction" or "reverse." A study combined the Kano model and a robust design approach to optimize design choices (e.g., as to the form of a mobile device) for esthetic perception (Chen and Chuang 2008). In Section 4.4, we explore an asymmetric relationship between absence vs. presence of a functionality.

In software engineering, *the next release problem* is the task of selecting enhancements to software (Bagnall et al. 2001) while taking into account customer preferences, dependencies among functions, and costs. *Software product line engineering* has studied methods for choosing a set of products that share more commonalities than differences (Benavides et al. 2010).

2.5 Summary

The present article is the first to focus on the optimization of functionality sets for early stage interaction design. Our approach builds on ideas from model-based optimization, interactive optimization, product design, the next release problem, and the Kano model, in particular in our aim support exploration of ideas in early-stage interaction design. We take into account the ill-defined nature of designers' problem-solving (Cross 2006) by proceeding from ideas of robust product designs (Chen and Chuang 2008). To specifically target interaction designers, our objective function takes into account two central design objectives important for designers: usefulness and ease of use (see also Gajos and Weld (2004)). The other two factors in our objective function, profitability and satisfaction, have been addressed in product design and assortment design, but to our knowledge not in model-based optimization research in HCI. We also consider dependencies among functionalities, which has been previously modeled for the next release problem, product-line engineering, and product-line selection.

3 PRE-STUDY: DESIGNERS' OBJECTIVES IN FUNCTIONALITY SELECTION

Despite a wealth of empirical studies of designers (Bayazit 2004; Carroll 1997; Cross 2004; Purcell and Gero 1998), no study, to our knowledge, has elicited their priorities in particular in functionality selection. The pre-existing studies offer anecdotal evidence for commonly known objectives in

Table 1. Interaction Designers in the Formative Pre-study

ID	Company type	Responsibilities	Degree	Years
1	Consultancy	Interaction design	B.Eng.	10
2	Consultancy	UI design, usability	M.Sc.	10
3	Design agency	Interaction design	M.Sc.	9
4	Software company	UX design, usability	M.A.	8
5	Consultancy	UX design, concepting	M.Sc.	7
6	Freelancer	Interaction and UX design, web design	M.Sc.	7
7	Consultancy	Interaction design, usability	B.Sc.	6
8	Consultancy	Functionality architecting, user research, usability	M.Sc.	5
9	Design agency	Interaction design, concepting	M.Sc.	5
10	Consultancy	Interaction design, wireframing, prototyping	M.Sc.	3

Note: "UX" refers to user experience and "UI" to "user interface."

this task. Goodman et al. (2011) report a case of a designer who would like to do feature selection by asking clients to sort potential features by business, user, and technical priorities, and another case where user experience-related goals dominated choice among two possibilities. More generally, efficacy (ability to complete tasks with no errors), efficiency (completion of tasks efficiently), learnability, and user satisfaction have been common design objectives since the advent of usability engineering (Nielsen 1994). Later on, the scope has broadened to include esthetic objectives, and recent redirection to user experience has put focus on user needs and creation of meaning (Alben 1996; Preece et al. 2015). The shared interest of interaction designers with service design has foregrounded business and marketing considerations (Holmlid 2009).

However, when it comes to functionality design, it remains unclear which objectives are most important in early-stage design, how these objectives are perceived and might be formulated as part of an objective function, if at all, and what kinds of other requirements are imposed by designers' existing practices. For this end, we conducted an interview study of 10 professional interaction designers. We interviewed them on how they solve the functionality selection problem in everyday design practice. Our goals were to understand (1) the objectives designers set for functions' selection and (2) the approaches they take to solve this problem.

3.1 Method

We interviewed 10 interaction designers with 3–10 years of experience in the profession (see Table 1). The interviews focused on a concrete and realistic design scenario to help them recount in detail how they select functionalities. They were told that there exist prior data such as scenarios and user profiles, but they were free to determine the content of said data. The participants were first asked to come up with a realistic design scenario that might involve roughly 120 functionalities. They were not asked to enumerate these.

We then focused on functionality selection, asking them consider two circumstances: (1) an ideal case, describing how the interviewee would like to solve this problem if there were no practical constraints such as lack of time or resources, and (2) a typical case, describing how they would solve this problem regularly in their work. This allowed us to understand how they might prioritize objectives. The "ideal" scenario was described as follows:

> Let us consider a case of interaction design for a complex consumer product. Your focus is on the task of choosing which functionalities or features to include in the product. You are given the following information: user requirements (120 features), user profiles (5 groups), and use scenarios (10). Your goal is to design a product

> that is simple enough for most users. You are free to define assumptions about the factors but please stay inside the scenario's frame. How would you solve this question in an ideal case?

During the interview, they were always first given the opportunity to respond spontaneously, but after proving their case in more detail, we asked more sensitizing questions about (1) heuristics or practices they might have followed; (2) their use of user requirements, profiles, scenarios, and other data; and (3) the involvement of different stakeholders in the decision-making process. To avoid biasing the designers, we did not give examples of how others "solve" this problem or how we believe it should be solved. All interviews were carried out by phone or VoIP and lasted 30–60 minutes each. The interviews were audio-recorded, indexed, and selectively transcribed for analysis.

3.2 Role in the Design Process

Overall, our data agree with prior understanding of objectives in interaction design. It agrees with accounts describing it as an iterative, exploratory, and corrective process deploying multiple types of representations and knowledge. The participants recounted utilizing well-known design methods, such as user profiles, scenarios, card sorting, prototyping, sketching, wireframing, and user requirements. However, to our surprise, many also stated that functionality selection is often a matter of "gut feeling"; that the prior data collected does not unambiguously solve this problem. One stated that functionality selection is sometimes straightforward because it is dictated by the client.

Although designers showed differences in how these methods were applied, all participants shared the use of scenarios and profiles combined with either sketching or prototyping as the generative method. They used this combination to construct "projections" of plausible futures wherein a design would be used. Seeing a sketch with particular functionalities implemented in a user interface allowed them to examine it against multiple criteria and develop understanding of which functionalities are needed and belong together. The participants engage in this process multiple times with the goal of developing a prioritized understanding of solution alternatives.

Use scenarios and user profiles had unique contributions in this approach. The designers stated that use scenarios allow them to understand the narrative and contextual factors that functionalities demand. User profiles, on the other hand, allow them to prioritize them. Sketching or prototyping was necessary to evaluate whether the functionality can be designed in an adequate way for those profiles and the scenarios. To our surprise, some recounted using sketches also for functionalities: "Sketches are often informal and can be also only partial. I also use short lists, which are aimed for my self, to bullet point the features that should be grouped together." Wireframing, on the other hand, was mostly done assuming a selected set of functionalities. However, wireframes would help them to see how to group functionalities. Card sorting was mentioned explicitly as a method for figuring out functionality sets and dependencies among them.

Designers also told often asking colleagues to get an opinion on this, and they discussed such decisions as part of a team. All of our participants recounted working on multi-discipline teams in which people with different areas of expertise contribute to functionality decisions. However, they had the role of realizing choices in concrete designs.

3.3 Stated Objectives

In our data, functionality selection stands out as a multi-objective challenge intertwined with the generation and evaluation of designs. When assessing alternative designs, the participants told considering well-known objectives such as usability, learnability, and usefulness but also business- and engineering-related goals such as implementability, technical goals, profitability, and attention to development costs. We grouped them as follows:

Usefulness: It is not surprising that all participants stated that it is essential to take into account how useful the selected functionality is to end users. They emphasized the decision to include a functionality requires understanding of what users want. One designer stated "Products for power users can have added complexity and require larger amount of learning, if they are demanded to be used in many ways." When asked to describe methods related to this aim, they recounted using use scenarios, user requirements, and prototypes. These are well in line with methods in user-centered and interaction design.

The also stressed that to determine usefulness, one has to consider which functionalities are linked to each other and how. This *dependency* can be either one-directional or two-directional. An example of the former is sub-functionality, such as selecting font size, which depends on inputting text while the reverse is not true. An example of the latter is saving and loading a file. Participants said that use scenarios are central for determining such dependencies. Dependencies among functionalities have been modeled previously in software engineering (see prior work), and we extend this to a design support tool particularly for interaction designers.

Usability and user experience: The designers told us that usability and experience factors, such as satisfaction, are critical but harder to determine. However, they felt that it is their "job" to "advocate" or "defend" usability objectives in a project. Handling them relies on expertise, research, and background knowledge. One participant stated that "these types of problems are solved intuitively."

Some participants told evaluating this also by assessing what happens if certain functionality is added or dropped (similar to an optimizer). Recognizability of features was also a recurring consideration. Adding an unrecognizable functionality affects usability and learnability negatively. Some applications end up being complicated by choice when too many functionalities are added. However, two designers recounted that a high level of usability is not always required.

Business value: Most designers in our study worked in consultancy companies and often had clients with business goals and stated constraints to development. All participants described situations wherein they had to make design decisions based on the client's requirements, sometimes going against their own suggestions. One designer even stated "The dirty truth is that the features that bring in the most amount money are in the top priority." Another way business goals surfaced was in how well a functionality might increase users' willingness to spend more money on the product. Some functionality may benefit only a small set of users and hence is prioritized below functionality that users are willing to pay more for. Some characteristics of these higher-business-value features were, according to the participants, novelty and desirability.

Cost: Costs of various types—such as time, development, and person months—were viewed as factors that are influenced most by other professionals working on the project. Time cost was cited as having an especially heavy influence on how many functionalities the final product would eventually end up having, with only one participant feeling that time was rarely a constraint. None of the participants believed that they could accurately evaluate all possible costs of a certain functionality on their own: "Sometimes there is not enough time to evaluate the complexity through user test and so decisions have be made by gut feeling." All but one emphasized collaboration with developers and other project members in estimating budget constraints. In addition, competition related risks could be viewed as potential costs.

3.4 Summary

The designers repeatedly reported on six objectives: usefulness, usability, satisfaction, learnability, business value, and cost. In addition, they considered dependencies among functions. They did not spontaneously report any methods of saying how a particular objective should be weighed. Instead, they perceived these objectives not quantitatively but qualitatively, with every function a

case of its own, and holistically, considering functionalities as part of a whole design, including the user interface. They reported considering alternative functionality choices mostly together with scenarios or sketches, sometimes imagining how the absence or presence of functionality affects the totality.

4 MODELING AND OPTIMIZATION

Our goals in developing an objective function and an optimization approach were (1) to cover the designers' objectives, (2) to allow rapid exploration of interesting design candidates, and (3) make the approach easily controllable. The objective function *USEP* addresses four objectives: usefulness, satisfaction, ease of use, and profitability. As discussed above, these goals are central to designers when considering functionalities, and they are central also to existing views of technology adoption and usability. To decrease load caused by defining input values to the optimizer, we reduced the number of objectives from six to four by combining related objectives. The four subsume the six objectives summarized above as follows. Ease-of-use (E) concerns with intuitiveness, familiarity, and routine, on the positive side, and overheads due to complexity and learning on the negative side. Satisfaction (S) considers positive surprises by presence of functionality on the one hand, and disappointments and other negative experiences on the other. We combine financial considerations into the profitability objective (P). It comprises of business value on the other hand, and implementation costs and risks on the other. In addition, we formalize dependencies that might exist among functionalities.

To allow the designer to explore ideas, we must solve the problem quickly while offering intuitive control such that changes to the task instance have predictable effects. For this end, we formulate the objective as a linear combination of independent terms, making it easy for designers to fill in ratings for functions and set weights in order to explore ideas. We solve the task using integer linear programming. Exact methods like integer programming use a structured (non-random) search approach that guarantees the optimal solution in finite time. For HCI, the benefit is that, unlike random search methods such as simulated annealing or genetic algorithms, mathematical guarantees can be provided for the optimum. What the optimizer produces is the best solution to the given functionality selection task. Rapid solution allows us to solve not only one task but multiple to explore the space for the designer.

4.1 Objective Function for Functionality Selection

We elaborate on the components in Equation (1). In the first, linear variant of the objective function we use linear relationships—except for dependency score, which necessitates a quadratic term. As we show later in the empirical evaluations, this approach is acceptable for designers. However, to improve the accuracy of the model, we revisit the linearity assumptions in a later section.

The *usefulness* U of a selection of functionalities is modeled as follows. Let u_v denote the individual usefulness of functionality v. However, its actual usefulness does not only depend on itself but also on the presence of related functionalities. For example, *Print Setup* is not very useful without *Print*. Thus, we also introduce dependencies between the presence and absence of features, i.e., let u_{vw} denote the usefulness of functionality v, provided that w is also selected, and let $u_{v\bar{w}}$ denote the usefulness of v in the absence of w. For the ease of notation, we define $u_{vv} := u_v$ and $u_{v\bar{v}} := 0$. The total usefulness for a selection of functionalities $\mathcal{X} \subseteq V$ is given by

$$U = \sum_{v \in \mathcal{X}} \left[\sum_{w \in \mathcal{X}} u_{vw} + \sum_{w \notin \mathcal{X}} u_{v\bar{w}} \right]. \tag{2}$$

Note that, in general, the dependencies are not symmetric, e.g., *Print* still makes sense without *Print Setup*. However, the scores can be symmetrized as follows:

$$\begin{aligned} U &= \sum_{v\in X}\sum_{w\in X}(u_{vw} - u_{v\bar{w}}) + \sum_{v\in X}\sum_{w\in V} u_{v\bar{w}} \\ &= \sum_{v\in X}\sum_{w\in X}\bar{u}_{vw}, \end{aligned}$$

where $\bar{u}_{vw}$ denotes the effective usefulness of the pair of functionalities v, w and is defined for distinct features as $\bar{u}_{vw} = \frac{1}{2}(u_{vw} + u_{wv} - u_{v\bar{w}} - u_{w\bar{v}})$, whereas $\bar{u}_{vv} = u_v + \sum_{w\in V} u_{v\bar{w}}$.

Note that we have not restricted the range for the numerical values here and explicitly allow negative values, and we even permit $-\infty$ for cases where the annoyance of having a functionality that does not make sense without a missing functionality would spoil the whole design.

The *Satisfaction S* is defined as the sum of satisfaction scores for functionalities included in a design:

$$S = \sum_{v\in X} s_v. \tag{3}$$

This score refers to the subjective experience of the functionality, as opposed to usefulness. Note that while we here do not consider pairwise interactions here, like for usefulness, mathematically it makes no difference they were used. All other objectives (Satisfaction, Ease-of-use, and Profitability) can be made to deal with pairwise interactions, however this means that users will have to input more information. In this work, we chose to limit pairwise interactions to usefulness to avoid excessive load when defining tasks.

The *ease-of-use E* can be defined in two ways. First, it can be defined as the sum of complexity of those functionalities *not* included in the design:

$$E = \sum_{v\notin X} r_v, \tag{4}$$

where r_v is the estimated reduction of the complexity when discarding v. This allows computation regardless of the scale of input values (negative values accepted). Alternatively, and more intuitively, E can be defined as the aggregated ease-of-use over all functionalities included in the design:

$$E = \sum_{v\in X} e_v \tag{5}$$

where e_v denotes the ease-of-use of a single functionality.

The *Profitability* p_v of a functionality v is defined in terms of its business value $v_v \in \mathbb{R}$ and costs $c_v \in \mathbb{R}$: $p_v = v_v - c_v$:

$$P = \sum_{v\in X} p_v = \sum_{v\in X}(v_v - c_v) \tag{6}$$

Note that if $c_v > v_v$, profitability is negative. Sometimes including functionalities that are negative in profitability is justifiable in light of the other parts of the objectives. Our profit model is simplistic when compared to existing models of product design (see Prior Work). This is justified, however, given that designers rarely engage in detailed profit calculations but view implementation costs and returns of investments only at a rough level.

Expanded, Equation (1) becomes

$$G = \omega_U \sum_{v\in X}\sum_{w\in X}\bar{u}_{vw} + \omega_S \sum_{v\in X} s_v + \omega_E \sum_{v\notin X} r_v + \omega_P \sum_{v\in X}(v_v - c_v). \tag{7}$$

We treat the weights ω as tuning factors that are controllable by the designer.

4.2 Overview of Optimization Approach

We use *integer programming* to solve the functionality selection problem. Integer programming is a general tool for modeling and solving discrete optimization problems. We refer the interested reader to Wolsey (1998) for an introduction to the theory of integer programming. For practitioners, it suffices to know that there are efficient exact general-purpose solvers based on branch-and-bound methods. That is, it is enough to set up a suitable integer program to obtain an effective solution for a particular problem.

To this end, we introduce binary variables, say $y_v = 1$ if feature $v \in V$ is selected and $y_v = 0$ otherwise. In the following, we will discuss different levels of complexity for modeling objective and constraints for our problem.

4.3 Basic Formulation

In the basic formulation, we consider a restriction of the objective function to linear terms. To this end, we distinguish two cases here.

Without Dependencies. If there are no dependencies between the presence and absence of functionalities or if we do not care about them, it happens that the effective usefulness of pairs of distinct functionalities v, w vanishes, i.e., $\bar{u}_{vw} = 0$. Since in this case there are no further constraints, we may compute a single score for each functionality comprising all properties. Hence, an optimum solution contains a functionality if and only if its corresponding score is positive. Thus, the problem can be solved by a trivial enumeration in linear time in this case.

With Strict Directional Dependencies. The problem becomes more relevant to designers, albeit challenging, if we additionally allow strict directional dependencies, i.e., if the dependency of a functionality v on a functionality w is so strong that having v but not w would yield an objective value that is worse than the trivial solution of selecting all functionalities.

The canonical integer programming model with directional dependencies has constraints of the form

$$y_w \leq y_v \qquad \text{for all features } w \text{ that depend on feature } v, \text{ i.e.,}$$

$y_v = 0$ implies $y_w = 0$ for all dependent features w. Note that these constraints are not effective if $y_v = 1$ and thus model indeed unidirectional dependencies.

Let $b : V \to \mathbb{R}$ denote the aggregated value of each functionality.[1] Note that a value might be negative, if its development is more expensive than its payoff. However, we might still chose a feature with negative value to satisfy dependencies to obtain a larger total payoff.

The complete Integer Linear Program (ILP) is thus

$$\begin{aligned} \max\ & \textstyle\sum_{v \in V} b_v y_v & & \\ \text{s.t. } & y_w - y_v \leq 0 & & \forall v, w \in V \text{ s.t. } w \text{ depends on } v \\ & y_v \in \{0, 1\} & & \forall v \in V. \end{aligned} \tag{8}$$

The following Lemma shows that this ILP can be solved efficiently, i.e., in polynomial time by exploiting its structural properties. This means that an ILP-solver would return an optimum solution already in the root node of the branching tree, i.e., without actual starting the branching process. This property is central to our use of this approach in explorative optimization later on.

[1] In the following text, we use a common notation in the optimization literature where a function that assigns a value to each element of a finite set can also be described by a vector where the ith coordinate belongs to the ith element in an arbitrary but fixed ordering of the finite set.

Lemma 4.1. *The basic ILP* (8) *is equivalent to its LP relaxation* $\max\{b^T y : A^T y \leq 0,\ 0 \leq y \leq \mathbb{1}\}$ *where A is the node-arc-incidence matrix of the directed graph describing the dependencies. The LP relaxation is equivalent to a min-cost flow problem and can be solved in* $O(nm \log n + n^2 \log^2 n)$ *time.*

Proof. Define a directed graph with a node for each functionality and an arc from functionality v to functionality w if the latter depends on the former. Consider its node-arc-incidence matrix $A \in \{-1, 0, 1\}^{n \times m}$, which has a row for each node and a column for each arc, i.e., a row for each of the n functionalities and a column for each of the m dependencies. This matrix A has exactly one 1 and one -1 in each column, i.e., -1 in the row for v and 1 in the row for feature w if w depends on v, so that $A^T y \leq 0$ yields the difference constraints in (8). Since node-arc-incidence matrices are totally unimodular, i.e., all determinants of square submatrices are in $\{-1, 0, 1\}$, and this property is maintained by taking the transpose and attaching a positive and a negative identity matrix, the polytope defining the LP relaxation is integer, i.e., all vertices are integer vectors and thus the optimum solution is attained at an integer point without explicitly enforcing this property, which proves the first claim. Furthermore, the dual of the LP relaxation, which equivalently could be solved instead, is

$$\min\{\mathbb{1}^T z : Ax + z \geq b,\ x, z \geq 0\} = \min\{\mathbb{1}^T z : Ax + z - z' = b,\ x, z, z' \geq 0\}.$$

Summing up these equations, we obtain

$$\mathbb{1}^T b = \mathbb{1}^T A + \mathbb{1}^T z - \mathbb{1}^T z' = \mathbb{1}^T z - \mathbb{1}^T z'.$$

If we multiply this implied constraint by -1 and add it to the dual LP, we obtain a constraint matrix that is again a node-arc-incidence matrix, i.e., a matrix with exactly one 1 and -1 in each column, where the additional constraint corresponds to an extra node and the variables z, z' represent the flow over arcs from and to the extra node, respectively, connecting it to all other nodes with a forward and backward arc. Hence, problem (8) is in fact a min-cost flow problem and can be solved in $O(nm \log n + n^2 \log^2 n)$ with Orlin's algorithm (Orlin 1988, 1993). □

4.4 Theoretical Developments

While the previous formulation is in line with the findings of the pre-study, we also explored formulations that would be better in line with practical expectations and literature in HCI. This increases the options practitioners to use the system for varied purposes.

We first revisit two assumptions of the previous formulation in light of theoretical work. First, we model pair-wise dependencies among functionalities. This is necessary in projects where dependencies are strong. We show how to model pair-wise dependencies for the usefulness objective; however, the same logic can be followed for any other objective.

We use quadratic terms in the objective function to model *pair-wise dependencies* such as in usefulness. That is, we still accumulate the linear contributions in a score b_v for each functionality v, but we also add

$$\sum_{v,w \in V} \bar{u}_{vw} y_v y_w \tag{9}$$

to the objective function. This then yields a quadratic integer program, which can be solved by modern MIP solvers. Pair-wise dependencies have been earlier modeled for example in the next release problem (see Prior Work).

Higher order dependencies can be modeled by submodular functions provided that they exhibit a nature of *diminishing returns*, e.g., if two functionalities have some overlap in their usefulness such that having both is not more useful than the sum of their individual usefulness in the absence of the other, i.e., $u_{vw} + u_{wv} \leq u_{v\bar{w}} + u_{w\bar{v}}$ for all functionalities v, w in the quadratic function in Equation (9). More formally speaking, diminishing returns are defined as follows:

Definition 4.2. A set function $f : V \to \mathbb{R}$ is *submodular* if

$$\forall \mathcal{X} \subseteq V,\ v, w \in V \setminus \mathcal{X}, v \neq w :\ f(\mathcal{X} \cup \{v\}) - f(\mathcal{X}) \geq f(\mathcal{X} \cup \{v, w\}) - f(\mathcal{X} \cup \{w\}).$$

A set function f is *supermodular* if $-f$ is submodular, i.e., the $\geq$ flips to a $\leq$ in the inequality above. A set function that is both submodular and supermodular is called *modular*.

Note that a modular function can be represented by a linear function in y. Moreover, linear terms w.r.t. binary decision variables in the objective function can be captured by modular functions and thus combined with submodular or supermodular parts, respectively.

Minimizing submodular functions and maximizing supermodular functions without further constraints is solvable in strongly polynomial time (Iwata et al. 2001; Schrijver 2000). Maximizing submodular functions is NP-hard but can be approximated within a factor of $\frac{1}{2}$ in linear time for non-negative submodular functions (Buchbinder et al. 2015).

Our second extension captures the fact that ease-of-use is compromised by including more and more functionalities. A designer seeking to find simple designs may want to impose a stricter non-additive relationship between ease-of-use and the number of functionalities in a design. Indeed, ease-of-use would increase linearly only in the case that the interface was organized randomly. We also note that some interface techniques allow constant time for a small number of functionalities; for example, hotkeys can make access time virtually constant, but users typically learn only few hotkeys. While we use the reciprocal function in the following, we recognize that the exact shape of this function depends on many factors, including the skills of the designer and the interface type. The reciprocal formulation is given as

$$E = \frac{1}{|\mathcal{X}| + 1}. \tag{10}$$

We show that this is a *supermodular function*. Note that this alternative does not consider designer-given ratings E. The benefit is that defining an input task is faster. The shortcoming is that we lose information about differences among functionalities. To this end, instead of $|\mathcal{X}|$, one might take a reciprocal of the sum of complexity associated with selected functionality.

Lemma 4.3. *The reciprocal ease-of-use* $E = \frac{1}{1+|\mathcal{X}|}$ *is supermodular.*

Proof. Let $\mathcal{X} \in V$ and distinct $v, w \in V \setminus \mathcal{X}$.

$$\frac{1}{1 + |\mathcal{X} \cup \{v\}|} - \frac{1}{1 + |\mathcal{X}|} = \frac{-1}{(1 + |\mathcal{X}|)(2 + |\mathcal{X}|)}$$
$$\leq \frac{-1}{(3 + |\mathcal{X}|)(2 + |\mathcal{X}|)} = \frac{1}{1 + |\mathcal{X} \cup \{v, w\}|} - \frac{1}{1 + |\mathcal{X} \cup \{w\}|}$$

□

Note that the reciprocal ease-of-use can also be incorporated into an integer linear or quadratic program by introducing the range constraint $\sigma_{\min} \leq \sum_v y_v \leq \sigma_{\max}$, where $\sigma_{\min}$ and $\sigma_{\max}$ define a range for the number of functionalities that we would like to have in our selection. By varying this range in a branching process and shifting the bound provided by a relaxation with the ease-of-use for the minimum number of functionalities allowed in the current branch, we will eventually obtain the optimum selection subject to this non-linear ease-of-use. Note that this trick can be applied for any function that acts on the number of selected functionalities.

Finally, the assumption that user satisfaction (S) is linearly determined by the included functionalities ignores the fact that the absence of a functionality can negatively affect the perception of a design. We want to support a designer who may be concerned that *absence* of a feature may be differently penalized by users than its presence. Both the Kano model (Kano et al. 1984) and *prospect theory* (Kahneman and Tversky 1979) assume that the absence of an important functionality is appraised more critically than its presence. In prospect theory, this is captured by the following

mathematical function for prospect π_v of item v:

$$\pi_v = \begin{cases} \sqrt{s_v} & \text{if } y_v = 1, \\ -2\sqrt{s_v} & \text{if } y_v = 0. \end{cases} \tag{11}$$

However, we note that not all seemingly non-linear behavior in the contributions to the objective function yield a non-linear integer program; for example, the satisfaction has the linear formulation

$$\pi_i = 3\sqrt{s_i} \cdot y_i - 2\sqrt{s_i}. \tag{12}$$

Note that total satisfaction can be negative if a design lacks some functionality that users expect it to have.

4.5 Performance

We assessed the numerical performance of the optimizer by using two realistic datasets. Our first dataset is the note-taking application in Study 1 (see below). It has 106 functions extracted from smartphone and desktop note-taking applications and rated by 11 professional designers. The details of this set are given in the corresponding section.

To assess scalability to larger problems, our second case considers the 1,364 functions of Autodesk Maya (2016) for Macintosh, a professional 3D graphics application used for movies, game development, and architecture. The functionality set covers tools that allow creation, animation, and rendering of complex 3D properties such as dynamic fluids, hair, cloth, and particles in addition to more common 3D object methods, such as polygon and NURBS. In addition, it has interface customization features. The set was collected by manual inspection of items found in the menus in the menu bar, toolbars, and graphical interface. Included functionality was required to be specific enough to represent a possible user interaction. Larger collections of connected functions, as in a camera tool, were broken into individual functionalities. One of the co-authors (PL) with experience in exhibition design rated the functions for a hypothetical scenario in exhibition stand design.

The results confirm the theoretical result: very large problems can be solved efficiently. Excluding the time to load the input file (LP file), Gurobi (solver) can solve the note-taking problem for the input of one designer in 0.4 milliseconds and the Maya problem in 2.8 milliseconds. When we artificially add complex dependencies among functions, solving the note-taking application problem takes 1.3 milliseconds. When dependency is treated as an objective, it increases to 3.1 milliseconds. With the number of designers providing input increased from 1 to 11, solution time increased to 1.1 milliseconds. We conclude that solving instances of realistic size can be very fast on a regular computer. This property is exploited in our robust optimization heuristic.

5 EXPLORATION OF FUNCTIONALITY DESIGNS

We here describe how the optimizer is deployed for exploring design candidates. The procedure consists of five steps:

(1) Defining the task by enumerating functionalities and asking stakeholders to rate them for U, S, E, and P (or any subset, to reduce effort).
(2) Computing the corresponding Integer Liner Program (ILP).
(3) Solving i instances of the ILP using Monte Carlo sampling.
(4) Mining k diverse designs, and finding the robust design: the best compromise among all instances.
(5) Visualizing the results (k diverse designs and the robust design) in an easily understandable set visualization.

Step 1. Task Definition: First, inputs to the optimizer are provided by filling in a functionality survey. In the survey, every functionality is listed, one per row, with Likert-scale ratings (1–5)

for each objective in the columns: usefulness, business value, implementation cost, ease of use, and user satisfaction. In addition, the criteria themselves are rated for their importance. To mark dependencies, functionalities were shown in the rows and columns of a matrix. A functionality was marked dependent of another functionality in the same group by checking the field in the corresponding matrix cell. Functionalities are grouped semantically. While grouping is optional, it makes the marking of dependencies faster. After filling the survey, *confidence* for each attribute is rated. The survey takes about 1 to 2 hours to fill for a set of 120 candidate functionalities. An example is given in the Appendix.

Step 2. Integer Linear Program Formulation: The ratings are used as input to instantiate an Integer Linear Program (ILP) described in Equation 8. Following Equation (7), the score b_v of a functionality v is computed as follows:

$$b_v = \omega_U * u_v + \omega_S * s_v + \omega_E * e_v + \omega_P * (v_v - c_v) \tag{13}$$

where u_v, s_v, e_v, v_v, and c_v are the survey rating of functionality v given for its usefulness, user satisfaction, ease of use, business value, and implementation cost, respectively. The weights ω correspond to designer-defined importance of each criterion. The dependency ratings are used to define the dependency constraints in the ILP as described in Equation 8. If a functionality v is rated to be dependent of a functionality w, the constraints ensure that v is only included in the design if w is also included. Note that modeling the dependencies as constraints simplifies the effective usefulness $\bar{u}_{vw} = 0$ used in Equation 7 to the single score u_v.

Step 3. Robust Optimization: First, we define an explorative optimization approach building on a robust optimization heuristic. We follow a Monte Carlo strategy of Beyer and Sendhoff (2007) . We first calculate the statistics (mean value, variance) and use these statistical parameters obtained as input in the deterministic optimization algorithm (ILP). We use the designer-given confidence ratings, or variability among survey respondents, to compute these parameters. We then sample i number of instances and solve them each to optimality in ILP. The number of instances i is defined by the designer. In our studies, we kept i between 10,000 and 100,000.

Step 4. Design Exploration: After solving each generated variant to optimality, the resulting set of optimal designs is then searched to find (1) k diverse designs and (2) one robust design, the best "compromise" design among all possible. The k diverse candidates are found in a heuristic search using Hamming distance. Search starts from a random candidate and then finds the candidate with maximum distance to itself. This procedure is repeated multiple times and the set with the largest mean maximum distance is chosen.

We also compute the "robust design," which is the design that has minimum average distance to others across all optimization tasks. Computing the robust design takes $O(n^3)$, whereas diversification is $O(n^2)$, where n is the number of designs.

Step 5. Visualization: We visualize the outcome in a list with annotated objective values, as shown in Figure 2. The objective values are normalized and then mapped to range 1 to 5 corresponding to yellow circles in the scheme. The rationale for the simple rating is that it is more intuitive to designers than a Pareto front. It makes it easier to compare designs in the four-dimensional evaluation space.

5.1 Deployment

The designer can use this method for several purposes:

(1) The single best solution can be found by examining the robust design (see "R" in Figure 2).
(2) To explore several alternatives, the designer can increase k (the number of diverse designs).

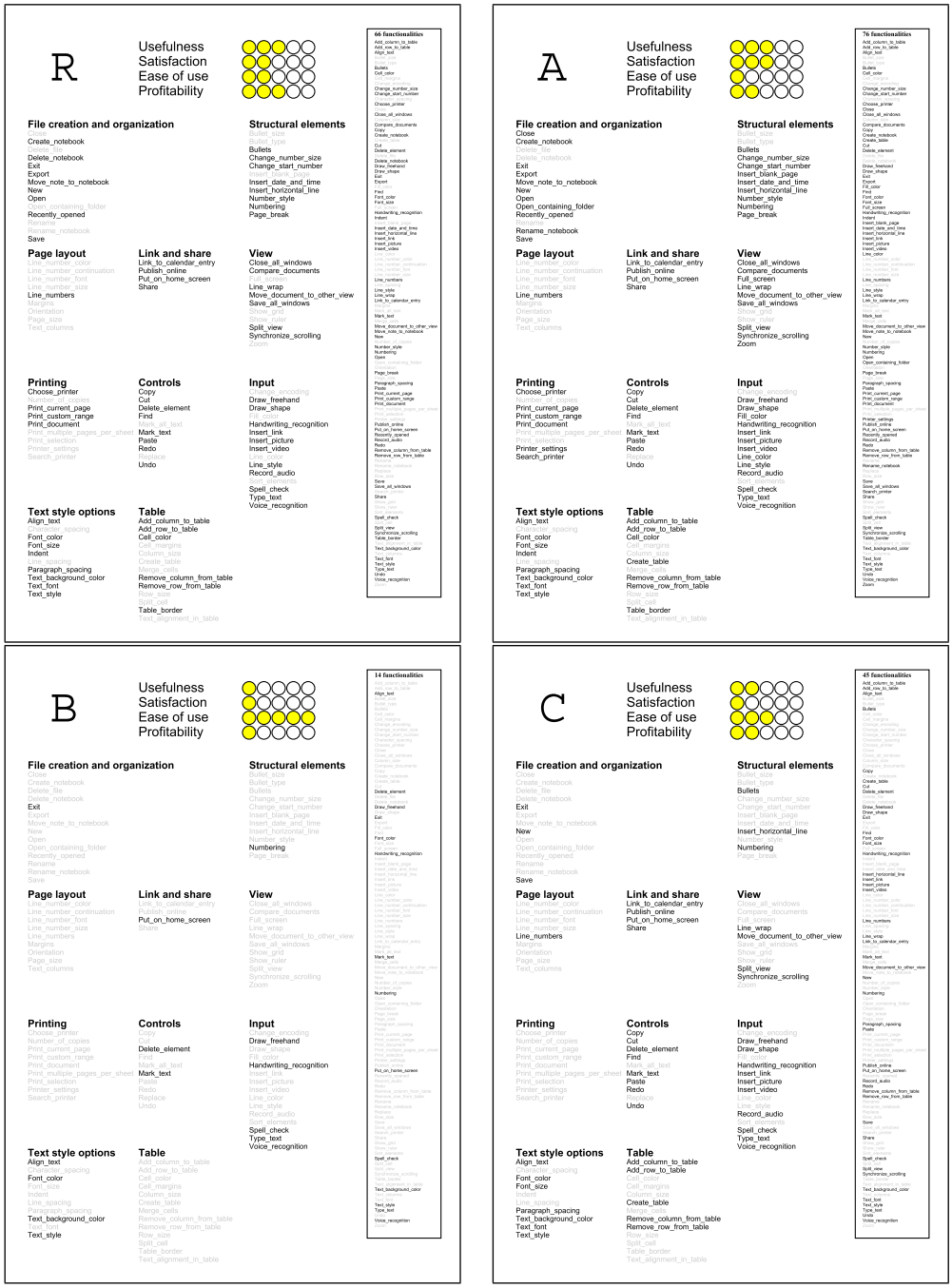

Fig. 2. A set visualization for the outputs of explorative optimization (data from Study 1). This example shows a sampling produced for the note taking application case with 106 functionalities. Top-to-bottom and left-to-right: the robust design (R) and three diverse designs: A, B, C. The diverse designs exhibit distinct trade-offs among objectives. To allow easy comparison, the outputs show normalized objective values (yellow circles), all selected functionality (black font) in alphabetical list and within their dependency-defined categories.

(3) To find the best compromise among several stakeholders, each should fill in the survey with confidence levels marked (see Study 2).
(4) To customize a product to different user groups, one can fill in one survey to represent each group.
(5) To simplify or upgrade a product, one can set an additional constrain to the integer programming formulation insisting that the total number of features is reduced or increased from the original.

Running a single optimization is very fast (a few milliseconds on a regular laptop computer). Robust optimization is more costly. Depending on the time budget, different number of Monte Carlo instances (i) can be computed. In the two studies reported in the following, we ran 10,000–100,000 instances and used $k = 6$. Running these instances took between 10 and 30 minutes on a regular laptop computer.

6 EMPIRICAL EVALUATIONS

We evaluated the approach first in a controlled design study wherein professional designers were asked to carry out a design task (for a note-taking application with 106 functionalities) with or without the optimizer. The method is similar to controlled design studies where a realistic task is given and performance is evaluated by examining the outputs. In our case, the participants evaluated their own designs, comparing computer-generated designs to their own. For our second study, we report results from a field deployment at a telecommunications company. The goal here was to get feedback on approach in a corporate setting with multiple stakeholders and a real product.

6.1 Study 1: Controlled Design Study

The purpose of the first study was to evaluate the usefulness of the explorative optimization tool in a controlled scenario with professional designers, and compare it to their traditional approach to select functionalities. Participants were presented a scenario in which they were hired by a company to design a note-taking application for mobile devices. Given a list of 106 potential functionalities, their task was then to create a set of designs by selecting the respective functionalities, once manually, and once by using our explorative optimization tool. The order of these two conditions was counterbalanced across participants.

6.1.1 Participants. Thirteen novice-to-intermediate-level interaction designers participated in the study. They had between one and nine years of experience at a company where they held titles ranging from "user experience designer" to "user interface evaluator."

6.1.2 Scenario. Participants were asked to imagine a scenario in which they were hired by a company to design a note-taking application. This is a basic application available on most devices and does not require any expert knowledge specific to a field of use. The scenario specified a user group (students from 18–25 years), example usage scenarios, and project constraints (e.g., should be completed within 6 months). Participants were free to make further assumptions if necessary.

6.1.3 Materials. The seed list of 106 potential functionalities was derived based on existing note-taking applications for desktop computers and mobile devices. It is large enough to allow interesting variations in design, while being manageable during the limited time of the study. For easier exploration, the functionalities were pre-grouped into 10 dependency-defined categories, such as "file creation and organization," "link and share," or "text style options." The full set can be seen in the visualizations of Figure 2. It spans basic functionalities such as "type text," "open file," or "save file," but also more advanced ones such as "change encoding," "insert video," or "change text

background color." Some functionalities are dependent on others, e.g., "paste" depends on "copy," or "change bullet size" depends on the functionality to "create bullet list." They also vary widely in the evaluation criteria on which participants were asked to rate them (see below). For example, functionalities such as "publish online," or "put on home screen" may satisfy the expectations of a mobile user, but could be very costly to implement and not even useful for the basic purpose of taking notes. Such functionalities are of particular interest when exploring different designs with different objectives and usage scenarios.

6.1.4 Procedure. The study consisted of three phases: (1) a functionality evaluation phase, (2) a design phase, and (3) an evaluation phase for the designs produced—completed after both of the first two phases were finished. Together the phases lasted 120–180 minutes per participant. The order of the first two phases was randomized between participants, with five participants completing the design phase first.

In the functionality evaluation phase, participants were asked to fill in a web-based survey in the location most suitable for them, with the requirements being that it had to be void of all disturbances and the survey should be completed in one sitting without major pauses. During the survey, the subjects had the chance to ask the experimenter questions on the survey by phone. Most surveys were completed in about 1 hour or less (maximum reported by participants was 2 hours). The survey consisted of evaluating the usefulness, business value, cost, ease of use, satisfaction, and dependency of the 106 possible functionalities of the note-taking application. The first five properties were evaluated on a scale of 1 to 5, with 1 representing a low value and 5 being a high value. To mark dependencies, functionalities of the same semantic group were shown in the rows and columns of a matrix. A functionality was marked dependent of another functionality in the same group by checking the field in the corresponding matrix cell. The full survey with all instructions is shown in the Appendix. After completion, a participant's answers were fed to the optimizer, which produced different designs, using the model.

In the design phase, the subjects were asked to produce 3–6 designs by selecting the functionalities from among the 106 presented. This phase was completed in the presence of the experimenter. The subjects were given a list of the functionalities on paper and asked to mark on the paper which should be included in the design. One list of functionalities represented one design. This phase took about 1 hour to complete.

After the first two phases had been completed, the subjects were asked to present their designs and to explain their advantages and disadvantages, while also sorting them from best to worst. After evaluating their own designs, participants were asked to do the same for the ones produced by the optimizer. The final part of the evaluation phase consisted of evaluating both the designer's and the optimizer's designs in the same batch. The evaluation phase and comments made by the designers were recorded and transcribed for analysis. This part took about 10 or 15 minutes. All responses and rationale were documented.

6.1.5 Results. In all of the cases except one, we set the optimizer to produce six distinct designs, one robust and five alternate ones emphasizing different properties of the functions. In the outlier case, the optimizer produced only one design. This was a result of the participant filling in the survey by using only the extrema of the response range, thereby rendering all of the optimizer's results the same.

Figure 2 shows an example outcome based on the ratings of one participant. The same visualization was used to show results to the designer. It shows all selected functionalities (black font) in alphabetical list and within their labeled category. For easy comparison, each set shows its normalized objective values simplified as yellow circles. In the figure, the first image on the top left (marked with R) shows the robust design. It consists of 66 functionalities that yield a medium

Table 2. Rankings of Designs Study 1

Participant	Ranking	Rank optimized design	Rank of robust design
1	C, 4, 1, R, F, D, 2, 3, E, A, D	1	4
2	3, R, 2, B, 1, D, F, A, E, G	2	2
3	1, 2, R, 3, F, A, D, E, B, C	3	3
4	A, 3, 2, C, 4, D, 1, E, B, R, F, 5	1	9
5	E, R, 4, F, A, D, 2, 1, C, B, 3	1	2
6	1, 3, 2, C, F, A, E, B, R, D	4	9
7	1, 2, 3, D, 4, R, E, A, B, C, F	4	6
8	3, 2, 1, 4, 5, R	6	6
9	A, 1, 3, 2, E, R, D, F, B, C	1	6
10	A, 2, E, 3, C, R, D, F, 1, B	1	6
11	A, 3, R, E, 2, D, R, C, 1, B	1	3

Optimizer-produced designs were ranked best in 6 out of 11 times.
Note: Optimized designs are marked alphabetically (A,B, . . .), robust design with R, and designer-produced designs numerically (1,2, . . .).

score for each of the objectives, as given by the ratings of the designer. The rest shows three diverse designs (A, B, C) which largely vary in the number of functionalities and objective scores. For example, B shows a very minimalistic design with only 14 functionalities optimized for ease of use. It only contains the most essential functionalities that were rated to be easy to use by the designer, such as "type text," "exit," or "font color."

Participants produced 3–5 designs without the optimizer. Six of the designers produced only the minimum number, three designs, and two designers produced five distinct designs. In six cases, the optimizer phase was completed first. Two participants decided not to continue taking part in the study due to time constraints, both during the functionality evaluation phase.

When asked to rank computer-supported and unsupported designs, in eight cases (out of 11), at least one of the optimizer's designs was included in the participant's list of the top three designs. Table 2 shows the rankings. In six cases, one of the optimizer's designs was selected as the best design, above the designer's own designs. In two cases, the participants felt that the optimizer's results were poor when compared to their own designs. These two placed all of their own designs ahead of the optimizer's. As mentioned, in one of these cases, the input values given by the designer were uniform, and only single design (marked R) was returned.

Table 3 shows three functionality sets generated by a designer "Amanda": "minimum," "added functionality," and "fancy one." Amanda's solution process started with a minimum design, created a variant by adding collaborative functionality to it, and then tried to diversify it by making it more "visually oriented" by adding handwriting and related functionality. As the table shows, the three designs share many basic functionality. She considered the second design (added functionality) the best. In the optimization phase, she evaluated designs "A" and "E" the best. Figure 3 shows these two highest designs with two lowest ranked optimized designs for Amanda. When interviewed about it, her rationale for ranking the designs was two-fold and considered (1) fit with the provided task scenario and (2) the number of functionality: she considered having excess functionalities a shortcoming. She finally ranked design "A" (by the optimizer) to be better than her own designs, because it had "more interesting special features that fit the product."

In post-study interviews, all of the participants except two reported that the optimizer had aided them in the design process. Participants who used the optimizer first felt that they had a better

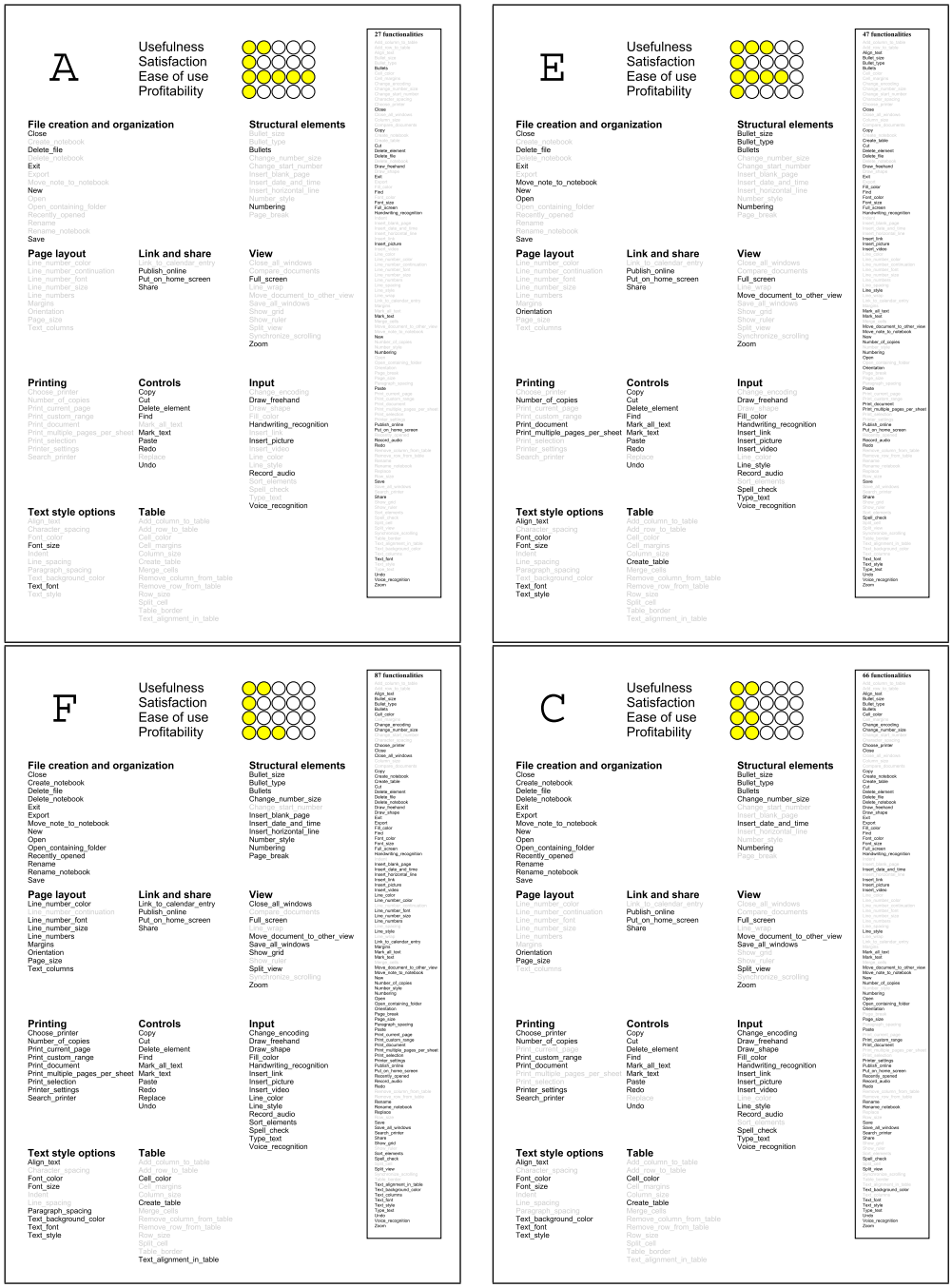

Fig. 3. Two best-ranked (top row) and two lowest-ranked (bottom row) optimized designs by a participant (Amanda).

Table 3. Functionality Sets Generated by "Amanda" in Study 1

"Minimum"	"Added functionality"	"Fancy one"
Close	Close	Close
Create notebook	Create notebook	Create notebook
Delete notebook	Delete notebook	Delete notebook
Exit	Exit	Exit
New	New	New
Open	Open	Open
Open containing folder	Open containing folder	Open containing folder
Rename	Rename	Rename
Save	Save	Save
Font size	Font size	Font size
Delete	Publish online	Publish online
Bullets	Share	Copy
Close all	Copy	Cut
Save all	Cut	Delete
Type text	Delete	Find
	Find	Paste
	Paste	Bullets
	Bullets	Close all windows
	Close all windows	Full screen
	Save all windows	Save all windows
	Insert link	Zoom
	Insert picture	Draw freehand
	Type text	Handwriting recognition
		Insert link
		Insert picture
		Insert video
		Record audio
		Spell check
		Type text
		Voice recognition

First design was a minimum set, which Amanda augmented in the second design. The last design explored a "Fancy" idea.

understanding of the available functionalities and that they were able to better employ this knowledge in producing their own designs in the design phase. Participants who, alternatively, completed the design phase first felt that the evaluation phase uncovered functionalities and viewpoints that they may not usually consider in the design process. All but two participants felt that the optimizer was able to produce designs that correctly reflected their evaluation of the functionalities, and that it could possibly act as an aid in the design process. Only one participant felt that the optimizer was unable to provide any help or additional knowledge to the design process.

On the negative side, the rating phase was described as exhausting and requiring extensive knowledge in various areas. The two participants who decided not to complete the study explained that the time-consuming and exhaustive nature of this phase was the reason for termination of their participation. Some participants also felt that the optimizer might be unable to produce good

results in situations wherein design is focused on improving an existing design. More value was attributed to the exploration of alternative approaches early on in design.

6.2 Study 2: Deployment at a Telecommunications Company

In our second empirical evaluation, we worked with a telecommunications company on a real product. Our goal was to qualitatively evaluate the approach in a multidisciplinary team. Our hypothesis was that this approach might help in the functionality selection process for new or updated applications by providing a method for better exploration of the full design space, resolving conflicts among stakeholders by providing numerical assessments for decisions, or by offering experts in a given area different perspectives to design. In collaboration with the company, we selected an existing service—an online platform for recording TV channels and video on demand—for which new updates with more functionalities are developed and launched regularly.

6.2.1 Task Definition and Optimized Designs. We first identified 79 existing functionalities within the application, then extended this set with 84 possible new functionalities provided by the company. We created a survey similar to that in the first study, in which the 163 functionalities were to be rated with respect to their usefulness, business value, cost, ease of use, and satisfaction.

The survey was sent to 15 employees of the company, directly involved in the development of the online service. They were asked to rate each functionality with respect to the aspects that best matched their expertise and that they felt comfortable rating. Eight of the employees responded to the request: one designer, one business manager, three developers, and three technical product owners. They rated between one and five aspects each, with seven people rating the functionalities for their usefulness, four for business value, two for cost, four for ease of use, and six for satisfaction. Dependencies were defined by one employee only, and later refined by one of the authors.

On the basis of the survey results, we set the optimizer to produce, in all, seven distinct designs with respect to three purposes: achieving (1) the best compromise across all stakeholders and all aspects; (2) the best design in the technical product owners' view; and (3) the best user-centered design, with a lower weight on profitability.

6.2.2 Workshop and Interviews. We then organized a 45-minute workshop in which six of the eight employees who completed the survey participated. In the workshop, we first showed a short presentation in which we explained the research topic and the motivation for the project. We then distributed the seven optimized designs (six diverse and one robust), as well as their survey ratings, and gave them 5 minutes for their individual assessment of the designs. We asked them to mark informative aspects, positive elements, and negative parts of the designs, with the task of afterwards discussing what would be the best design for the next update of their application. The discussion was free and not researcher-moderated. It was held in two groups because three of the participants remotely joined the meeting. However, the discussion of the best design quickly shifted towards feedback on the optimization approach, which was held in one large group.

6.2.3 Results. Similar to the first study, the participants stated that filling in the survey form took a long time. One commented that it was "tricky to evaluate without actually doing stuff." Instead of everybody completing the survey on his or her own, they would prefer to work together and would "rather have a discussion" on the rating of each functionality in a varied group of experts. They understood that the optimized designs are the result of their subjective input values and noted that, instead of various experts working closely with the applications, they would take into account ratings based on usage statistics, user surveys, and external experts not directly related to the project—who could provide an opinion from "outside the box." To reduce the number

of functionalities to be rated, they also suggested rating groups of functionalities related to the same concept, rather than individually.

In assessing the optimized designs, they were particularly interested in (1) the minimal design, which identified the most important functionalities with respect to different objectives, based on their assumptions. It contained 29 functionalities, fewer than half the number in the current application. Surprising to us, they were also very interested in (2) the functionalities *not* included in the optimized designs. Those were rated low by most people and considered unnecessary, a fact that never came up in prior meetings. In light of this, they suggested creating usage statistics for these functionalities, to better assess their real usage and thereby verifying or adjusting their beliefs, and removing functionalities if they are not used.

The participants could imagine the optimization approach as an interactive tool used throughout the development of the application, and a tool that grows in tandem with it. In this kind of use, ratings should be updated as new values come in, based on user feedback and analytics, and only a few functionalities should be added and rated at once. Such a "tracing tool", similar to the concept of design rationale (MacLean et al. 1991), would make assumptions and decisions transparent and traceable throughout the project.

The participants also noted that they would find the optimization approach itself more useful in the early process of planning a new application, in which the goal is to find a minimum viable product, instead of using it to decide on 1–2 functionalities to be added to a new update. Lastly, they suggested adding one objective for which functionalities should be rated: the risk, in particularly with respect to its implementation. For example, a functionality may be cheap to implement but depend on a third-party API. This increases the risk because the implementation or operation of that API cannot be controlled by the company itself.

We conclude that the approach may be more useful in the early stages of a design project than in the development of new updates. However, throughout the project the company could benefit from a tool that captures all previously made assumptions, raise discussion around them, and lets the company update various aspects of it as usage data come in, and quickly shows the new opportunities and trade-offs arising from the updates.

7 DISCUSSION

This article has investigated a new idea on algorithmic support for a creative problem in early-stage interaction design. We defined functionality selection formally and developed a combinatorial optimization method to solve the task. Building on interviews of designers, we defined it as an optimization task with four main objectives. The objective function goes beyond previous work by addressing usability, usefulness, satisfaction, as well as business objectives (USEP). To assist designers in the exploration of ideas, we implemented a proof-of-concept in exploratory optimization with the goal of producing diverse functionality sets, sets that they might not normally entertain. A key assumption was that input to the optimizer are inherently uncertain at this stage of design, and that interesting designs can be generated by exploring diverse interpretations of the inputs algorithmically. Despite issues discussed below, designers reported that they gained better understanding of functionalities and could employ this knowledge in elaborating designs. In the light of findings from the two studies, we conclude that the main benefit of this approach lies not in deciding if a particular functionality should be included in a design but in broadening the designer's understanding of the landscape of possible designs.

We see many opportunities to develop the approach in the future. The first is the objective function. Our task definition rests on the presumption that the four objectives (usefulness, satisfaction, ease-of-use, and profitability) are independent of each other. The relationship among them was left to the consideration of the designer. That satisfactory results could be obtained at all may be

attributed to experienced designers' ability to envision just from a functionality set how a user interface might look like and how it might be used. To extend the scope of this approach beyond professional designers, and support unexperienced designers and even novices, we need to solve how functionality selection is related to user interface design. However, the downside of this will be that defining a task will be less straightforward, as objectives and constraints related to the UI need to be defined.

The second opportunity addresses perhaps the greatest limitation of this approach: the time that it takes for a designer to define a selection task. It took about 10–30 minutes for a designer to rate about 100 functionalities for just one criterion. While we believe that this investment is reasonable, at least when put to the context of typical investments to design projects, this issue needs to be addressed to make the approach suitable for larger projects. For example, in Study 2, we proposed that an expert fills only certain objectives, and we later combine the answers of different stakeholders automatically into a single task definition simply by weighing the according to stated confidence. But one can explore other ways to facilitate the process of filling out the ratings survey, such as workshops where different stakeholders come together to fill the survey while discussing the design, thus creating common ground. This was suggested by our participants in Study 2. Different parts of the survey might also be filled by clients or users, or based on automatically collected data such as usage logs.

Third, the method is limited to the selection of functionalities that have already been defined and enumerated. Enumeration of functionalities is common in projects that follow requirements engineering or usability engineering, but might be more challenging in projects where the discovery of functionalities is part of the challenge, such as concept design, or that follow lean or agile practices. To better accommodate such domains, one needs to develop more interactive tools for quickly defining and revising functionality candidates. On a related note, an interface needs to be developed that allows easily defining selection tasks and controlling the optimizer. Our optimizer presently runs from the command line. In both studies reported above designers worked with the survey tool and a researcher used the optimizer and presented the outputs to the designers. It will also be interesting to explore how this approach can be combined with design efforts downstream. This is motivated by the observation that designers felt that it is hard to evaluate functionality without a UI. This necessitates a tool that makes it easy to iterate and change hypotheses. Despite these challenges, the results are largely positive and warrant more attention to computational methods in interaction design.

We conclude that computational approaches in interaction design, when appropriately formulated, might help address one of its recognized problems: a gap between practice and theory (Roedl and Stolterman 2013; Rogers 2004). Optimization offers a way for designers to express their knowledge in an explicit, scrutinizable format, and yet use it directly to generate ideas for design with relatively little effort. The approach does not require deep understanding of optimization. Perhaps the most significant suggestion of the present work is that optimization methods can be made useful in a domain normally assumed to be outside the scope of the "reductionist" stance (Schön 1983). Optimization methods and machine learning have revolutionized several professions, and we hold that it is important to study their applications in interaction design. That the problems interaction designers face have escaped optimization methods cannot be explained by the fact that these problems are prohibitively complex, since they are hardly more complex than other product-related decisions where successful approaches already exist. There is more room for computational methods also in this space.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

We thank anonymous reviewers, Pauli Miettinen, Kasper Hornbaek, Enrico Bartolini, and Janin Koch for helpful suggestions.

REFERENCES

Lauralee Alben. 1996. Defining the criteria for effective interaction design. *Interactions* 3, 3 (1996), 11–15.

M David Albritton and Patrick R. McMullen. 2007. Optimal product design using a colony of virtual ants. *European Journal of Operational Research* 176, 1 (2007), 498–520.

Anthony J. Bagnall, Victor J. Rayward-Smith, and Ian M Whittley. 2001. The next release problem. *Information and Software Technology* 43, 14 (2001), 883–890.

Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. 2013. Menuoptimizer: Interactive optimization of menu systems. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*. ACM, 331–342.

Nigan Bayazit. 2004. Investigating design: A review of forty years of design research. *Design Issues* 20, 1 (2004), 16–29.

David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636.

Hans-Georg Beyer and Bernhard Sendhoff. 2007. Robust optimization–A comprehensive survey. *Computer Methods in Applied Mechanics and Engineering* 196, 33 (2007), 3190–3218.

Louis L. Bucciarelli. 1994. *Designing Engineers*. MIT press.

Niv Buchbinder, Moran Feldman, Joseph Seffi Naor, and Roy Schwartz. 2015. A tight linear time (1/2)-Approximation for unconstrained submodular maximization. *SIAM Journal on Computing* 44, 5 (2015), 1384–1402. DOI: http://dx.doi.org/10.1137/130929205 arXiv:http://dx.doi.org/10.1137/130929205

Rainer E. Burkard and Dipl Math J. Offermann. 1977. Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. *Zeitschrift für Operations Research* 21, 4 (1977), B121–B132.

Bill Buxton. 2010. *Sketching User Experiences: Getting the Design Right and the Right Design: Getting the Design Right and the Right Design*. Morgan Kaufmann.

Stuart K. Card, Allen Newell, and Thomas P. Moran. 1983. The psychology of human-computer interaction. Lawrence Erlbaum Associates, Inc.

John M. Carroll. 1997. Human-computer interaction: Psychology as a science of design. *Annual Review of Psychology* 48, 1 (1997), 61–83.

Chun-Chih Chen and Ming-Chuen Chuang. 2008. Integrating the Kano model into a robust design approach to enhance customer satisfaction with product design. *International Journal of Production Economics* 114, 2 (2008), 667–681.

Nigel Cross. 2004. Expertise in design: An overview. *Design Studies* 25, 5 (2004), 427–441.

Nigel Cross. 2006. *Designerly Ways of Knowing*. Springer.

Nigel Cross. 2011. *Design Thinking: Understanding How Designers Think and Work*. Berg.

Alan Dix. 2009. *Human-Computer Interaction*. Springer.

Kees Dorst and Nigel Cross. 2001. Creativity in the design process: Co-evolution of problem–solution. *Design Studies* 22, 5 (2001), 425–437.

Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta. 2001. Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*. ACM, 69–76.

Jerry Alan Fails and Dan R. Olsen Jr. 2003. Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*. ACM, 39–45.

Donald L. Fisher. 1993. Optimal performance engineering: Good, better, best. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 35, 1 (1993), 115–139.

Krzysztof Gajos and Daniel S. Weld. 2004. SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*. ACM, 93–100.

Elizabeth Goodman, Erik Stolterman, and Ron Wakkary. 2011. Understanding interaction design practices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1061–1070.

Lars Hallnäs and Johan Redström. 2006. *Interaction Design: Foundations, Experiments*. Textile Research Centre, Swedish School of Textiles, University College of Borås and Interactive Institute.

Stefan Holmlid. 2009. Interaction design and service design: Expanding a comparison of design disciplines. In *Proceedings of the Nordes Conference*. 2 (2009).

Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. 2001. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM* 48, 4 (July 2001), 761–777. DOI: http://dx.doi.org/10.1145/502090.502096

Jianxin Roger Jiao, Timothy W. Simpson, and Zahed Siddique. 2007. Product family design and platform-based product development: A state-of-the-art review. *Journal of Intelligent Manufacturing* 18, 1 (2007), 5–29.

Bonnie E. John, Konstantine Prevas, Dario D. Salvucci, and Ken Koedinger. 2004. Predictive human performance modeling made easy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, 455–462. DOI: http://dx.doi.org/10.1145/985692.985750

Daniel Kahneman and Amos Tversky. 1979. Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society* (1979), 263–291.

Noriaki Kano, Nobuhiko Seraku, Fumio Takahashi, and Shinichi Tsuji. 1984. Attractive quality and must-be quality. *Journal of the Japanese Society for Quality Control* 14, 2 (1984), 39–48.

Andreas Karrenbauer and Antti Oulasvirta. 2014. Improvements to keyboard optimization with integer programming. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST'14)*. ACM, New York, NY, 621–626. DOI: http://dx.doi.org/10.1145/2642918.2647382

William R. King and Jun He. 2006. A meta-analysis of the technology acceptance model. *Information & Management* 43, 6 (2006), 740–755.

A. Gürhan Kök and Marshall L. Fisher. 2007. Demand estimation and assortment optimization under substitution: Methodology and application. *Operations Research* 55, 6 (2007), 1001–1021.

Viswanathan Krishnan and Karl T. Ulrich. 2001. Product development decisions: A review of the literature. *Management Science* 47, 1 (2001), 1–21.

Ranjitha Kumar, Jerry O. Talton, Salman Ahmad, and Scott R. Klemmer. 2012. Data-driven web design. In *Proceedings of the International Conference on Machine Learning, ICML*.

Lissa Light and Peter Anderson. 1993. Designing better keyboards via simulated annealing. *Miller Freeman Publishers*.

Jonas Löwgren and Erik Stolterman. 2004. *Thoughtful Interaction Design: A Design Perspective on Information Technology*. The MIT Press.

Jock Mackinlay. 1986. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics (TOG)* 5, 2 (1986), 110–141.

Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. 1991. Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction* 6, 3–4 (1991), 201–250.

Joe Marks, Brad Andalman, Paul A. Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, and others. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., 389–400.

Jakob Nielsen. 1994. *Usability Engineering*. Elsevier.

Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2014. Learning layouts for single-pagegraphic designs. *IEEE Transactions on Visualization and Computer Graphics* 20, 8 (2014), 1200–1213.

Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2015. Designscape: Design with interactive layout suggestions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1221–1224.

Dan Olsen. 1992. *User Interface Management Systems: Models and Algorithms*. Morgan Kaufmann Publishers Inc.

James Orlin. 1988. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC'88)*. ACM, New York, NY, 377–387. DOI: http://dx.doi.org/10.1145/62212.62249

James B. Orlin. 1993. A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.* 41, 2 (April 1993), 338–350. DOI: http://dx.doi.org/10.1287/opre.41.2.338

Antti Oulasvirta. 2017. User interface design with combinatorial optimization. *IEEE Computer* 50, 1 (2017), 40–47.

Fabio Paterno. 2012. *Model-based Design and Evaluation of Interactive Applications*. Springer Science & Business Media.

Jenny Preece, Helen Sharp, and Yvonne Rogers. 2015. *Interaction Design-beyond Human-Computer Interaction*. John Wiley & Sons.

Angel R. Puerta. 1997. A model-based interface development environment. *IEEE Software* 14, 4 (1997), 40–47.

A. T. Purcell and John S. Gero. 1998. Drawings and the design process: A review of protocol studies in design and other disciplines and related research in cognitive psychology. *Design Studies* 19, 4 (1998), 389–430.

David J. Roedl and Erik Stolterman. 2013. Design research at CHI and its applicability to design practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, 1951–1954. DOI: http://dx.doi.org/10.1145/2470654.2466257

Yvonne Rogers. 2004. New theoretical approaches for HCI. *Annual Review of Information Science and Technology* 38, 1 (2004), 87–143.

Dan Saffer. 2010. *Designing for Interaction: Creating Innovative Applications and Devices*. New Riders.

Donald A Schön. 1983. *The Reflective Practitioner: How Professionals Think in Action*. Vol. 5126. Basic Books.

Alexander Schrijver. 2000. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B* 80, 2 (2000), 346–355. DOI: http://dx.doi.org/10.1006/jctb.2000.1989

Allan Shocker and V. Srinivasan. 1974. A consumer-based methodology for the identification of new product ideas. *Management Science* 20, 6 (1974), 921–937.

Michael Terry, Elizabeth D. Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. 2004. Variation in element and action: Supporting simultaneous development of alternative solutions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 711–718.

Debora V. Thompson and Michael I. Norton. 2011. The social utility of feature creep. *Journal of Marketing Research* 48, 3 (2011), 555–565.

Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and explore with a layout optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS'16)*. ACM, New York, NY, 543–555. DOI: http://dx.doi.org/10.1145/2901790.2901817

Terry Winograd. 1997. The design of interaction. In *Beyond Calculation*. Springer, 149–161.

Laurence A. Wolsey. 1998. *Integer Programming*. J. Wiley & Sons, New York (N.Y.), Chichester, Weinheim.

Shumin Zhai, Michael Hunter, and Barton A. Smith. 2002. Performance optimization of virtual keyboards. *Human-Computer Interaction* 17, 2–3 (2002), 229–269.

Received September 2016; revised June 2017; accepted July 2017