Transformer: A Database-Driven Approach to Generating Forms for Constrained Interaction

Protiva Rahman, Arnab Nandi
The Ohio State University
Columbus, Ohio
{rahmanp,arnab}@cse.ohio-state.edu

ABSTRACT

Form-based data insertion or querying is often one of the most time-consuming steps in data-driven workflows. The small screen and lack of physical keyboard in devices such as smartphones and smartwatches introduce imprecision during user input. This can lead to data quality issues such as incomplete responses and errors, increasing user input time. We present Transformer, a system that leverages the contents of the database to automatically optimize forms for constrained input settings. Our cost function models the user input effort based on the schema and data distribution. This is used by Transformer to find the user interface (UI) widget and layout with ideal input cost for each form field. We demonstrate through user studies that Transformer provides a significantly improved user experience, with up to 50% and 57% reduction in form completion time for smartphones and smartwatches respectively.

CCS CONCEPTS

• **Human-centered computing** → **User interface programming**; **Empirical studies in HCI**.

ACM Reference Format:
Protiva Rahman, Arnab Nandi. 2019. Transformer: A Database-Driven Approach to Generating Forms for Constrained Interaction. In *24th International Conference on Intelligent User Interfaces (IUI '19), March 17–20, 2019, Marina del Rey, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3301275.3302269

1 INTRODUCTION

Many of our daily activities such as ordering food and booking appointments require us to fill out forms. While these used to be filled on paper, nowadays most forms are digitized and hence connected to a database. This shift from paper to digital has been accompanied by the rise of constrained display devices such as smartphones and tablets. Further, with the advent of wearables such as smartwatches and smart glasses, oftentimes the only mode of interaction for the user is through touch as opposed to on a physical keyboard [74], which makes form input inefficient.

This variety in types and sizes of devices makes manually designing websites for each one [22] expensive. Responsive web design makes this easier with tools such as Bootstrap [2] which adapt layout to screen size, but the designer still needs to manually specify the widget type and dimensions for every screen size. This brings added burden on the designer who now has to consider a combinatorially large number of options to find the optimal one. This can be time consuming and mentally challenging for the designer, and often requires extended domain specific usability studies [29]. In fact, it has been found that designers prefer aesthetics to efficiency [13, 67] and find it difficult to quantify the benefits of one design over another [13, 58]. While data-driven approaches [24, 50, 63, 72] are being explored for augmenting the designers' abilities, they require large datasets of interface designs, which is not always available. But every form is linked to a database which can be leveraged.

Motivating Example: Consider filling a modified version of the online application for a U.S. passport [9], on an iPhone 5s (Figure 2). There are four input text fields, which can have lengths of up to 25 characters. The gender input field uses a radio button which requires one click on a desktop, but here might require multiple taps due to its small size. The rest of the fields use dropdown options, which are presented in a slot machine style list. This means for country of birth, a user might have to scroll through 239 options in the worst case. Additionally, the user is required to scroll horizontally and vertically through the form. Date of birth and SSN require the user to navigate to the numeric keyboard. Filling an extended version of this form takes over 2 minutes (Section 4).

An optimized interface should reduce completion time and scope of error. Preference should thus be given to widgets that require fewer interactions. Further, widget sizes should be large enough for the user to make a selection accurately without scrolling and zooming. The optimized extended version of the above form (as described in Table 4) takes on average 1.5 minutes to complete which is around a 25% reduction from the original time. The time reduction stems from replacing the *dropdowns* with *radio buttons*, *rangeslider*, and *segmented controllers* (Figure 1), which reduces scrolling. Moreover, *dropdowns* require atleast two taps, while the others require one (Figure 2). Adjusting the widget and size can thus significantly improve user experience.

The widget choice depends on the device (it is easier to type on desktop than smartphone) and database that is linked to the form. Each form field corresponds to a database column. The schema, data types, and distribution dictate the form design. For `one-to-many` schema relations, we have to ensure that the parent field appears before its children. Next, the data type of the column determines the default keyboard and if certain widgets such as *rangesliders* are feasible. The column data informs the number of user interactions. For example, for text widgets, the average length of the data values

IUI '19, March 17–20, 2019, Marina del Rey, CA, USA

ACM ISBN 978-1-4503-6272-6/19/03…$15.00
https://doi.org/10.1145/3301275.3302269

determines typing time. We use these observations to develop a cost model to decide on the most appropriate widget for a form field. Optimizing forms is thus entirely database-driven and depends on the data model, allowing our methods to be used with any database implementation.

Challenges in Automating Form Design: Developers base their design decisions on years of experience and automating this knowledge is not straightforward. This is illustrated during *widget selection* for a form attribute. Generally, it is easier to select from choices than to type on a small screen. However, if the number of choices is large, the additional effort in scrolling makes it more convenient to type. But there is no definite measure of at what point the number of options is "too many".

Another challenge is in *ordering and grouping fields*. Due to the small size of screens, web pages either require the user to zoom or scroll along both dimensions. Scrolling can create issues for users, such as re-finding their place in the form. The use of pagination provides an unambiguous way to step through the entire form, one screenful at a time. While Popp et al. [59] motivate the use of scrolling over tabbed navigation, these comparisons are done for forms where tabbed navigation is discontinuous (through the use of a "Submit" button), which is not true in our case. Moreover, Petychev [56] and Sanchez [66] suggest that scrolling reduces comprehension and takes longer [23, 57] in survey tasks. Keeping these in mind, we design our system to split the form into multiple pages and reduce scrolling (if scrolling is preferred, our approach can still be used by laying out the pages one below the other). However, splitting the form into the minimum set of pages requires considering all orderings of fields, since the ordering dictates space efficiency, which is exponential in number of fields.

This is problematic because there is a *time constraint* on optimizing the form. A user could request a webpage to be optimized when they access it on their device and there are strict latency constraints whenever there is a human-in-the-loop [40]. The form then needs to be optimized on the fly, since it is not possible to store all versions of all forms. Moreover, even if forms are cached, the schema and data distribution might change with time, requiring a form which was optimized for a device earlier to be redesigned upon access at a later time. A brute force approach is not practical since grouping n fields in different pages is equivalent to partitioning the set of size n. The number of ways to do this is given by Bell numbers [42] which have a growth rate of $O(n^n)$ [54]. For a form with 16 fields, there are $B_{16} = 10,480,142,147$ subsets for which we have to calculate the cost, which can take hours, making it infeasible to consider all possible layouts.

Prior work in this area includes domain-model based optimizations [62, 67, 69, 71]. However, all of these use rules for widget selection based on the data type of the attribute, but do not leverage the attribute values. Another line of work, the SUPPLE systems [30, 31, 33], has focused on personalizing interfaces, also via cost-based layout optimization. But they require studies on preference-elicitation [31] or an ability modeling task [34], as well as a matching function where the designer specifies the appropriateness of a widget for a particular field. While work in modeling user input is necessary for personalizing interfaces, the suitability of a widget can be automatically inferred, reducing the burden on the designer.

Contributions: We introduce Transformer, which automatically optimizes form layout for a given screen size. The main contributions of this paper include: (1) A cost function of human effort in data entry that leverages the *contents of the database* to estimate the expected number of taps, slides, and scrolls; (2) Implementation details of Transformer that optimizes a web form for any screen size in $O(n^2)$ in number of fields, using the cost function. Transformer allows for automatic widget selection and provides a guide for grouping fields using database properties; (3) Empirical validation of Transformer through user studies on a smartphone and simulated smartwatch for eight forms. Our results show that there is up to 57% reduction in form completion time and up to a 93% improvement in user rating.

The paper is organized as follows: In the next section, we describe our cost model and formulate the form optimization problem. This is followed by the details of our system implementation in Section 3, experimental evaluation in Section 4, and discussion of related work in Section 5. Finally, we conclude in Section 6.

2 COST MODEL OF HUMAN EFFORT

In order to estimate the user effort during data entry, we define a cost function that leverages the database to measure widget suitability for each field. Our interactions are based on prior work in estimating task completion time [25, 38, 45], however we employ the database to quantify them. We clarify relevant terms in the context of our work in Table 1.

Table 1: Preliminaries: Terms used in this paper

Form: An ordered list of form field, rendered as HTML.
Form Field: Each element on the form that requires input.
Widget: UI element for entering data, dependent on input type of the field, e.g., radio button, checkbox, etc.
Data: The prior values entered for the corresponding database column.
Data Type: The data type for each column as defined in the schema.
Database: Underlying relational database linked to the form. Each form field corresponds to a database column. Submitting a form is thus equivalent to issuing an `insert` or `select` query. The `where` clause of the queries constrain columns to the values entered by users in the analogous form fields.
Data Distribution: The frequency of each data value, which can be used to order options and default values in widgets.
Cardinality: Number of distinct values in a database column.
Foreign-key: Reference to a value in another table.
One-to-Many Join: Each row in Table 1 is linked to one row in Table 2. Multiple rows in Table 1 **can** be linked to same row in Table 2.
Many-to-Many Join: Each row in Table 1 can be linked to more than one row in Table 2. Multiple rows in Table 1 **can** be linked to same row in Table 2.
One-to-One Join: Each row in Table 1 is linked to exactly one row in Table 2. Two rows in Table 1 **cannot** link to the same row in Table 2.

2.1 Types of Interactions

Each user interaction can be broadly classified into four distinct categories that contribute differently to the completion time: *tap*, *scroll*, *slide*, and *typing* on soft keyboard. The first three actions roughly correspond to tapping, dragging, and flicking interactions in Lee et

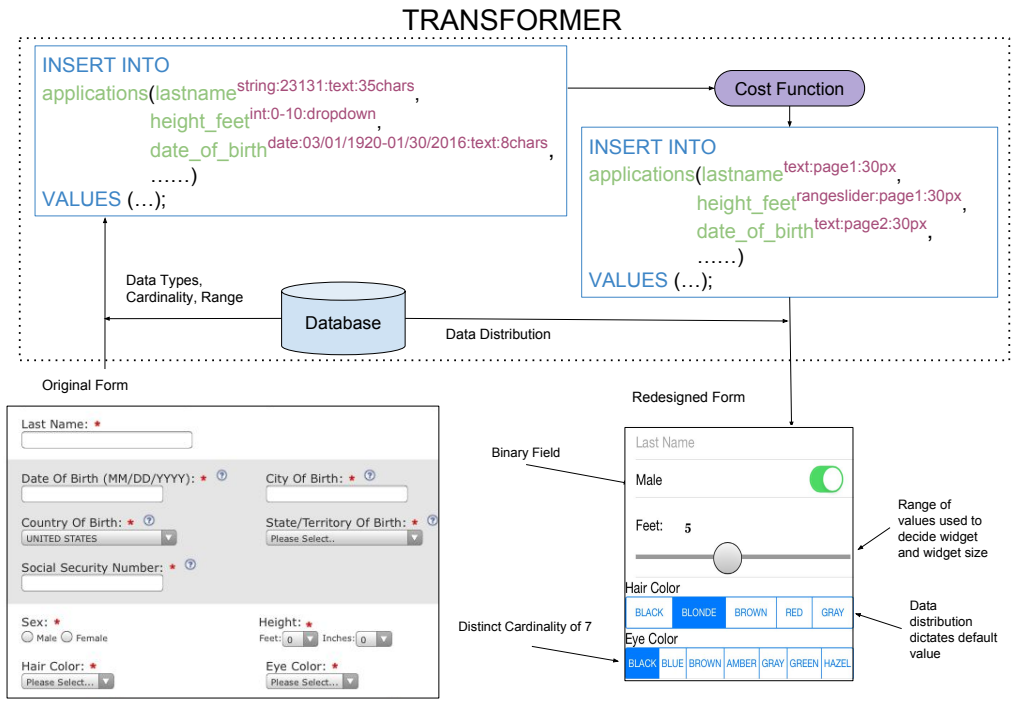

Figure 1: An overview of the steps of TRANSFORMER as it optimizes a modified version of the U.S. Passport Application Form (Form G in Table 4). The original form is used to extract the required fields (which can also be input by the user in the absence of a form) and translated to a type-annotated intermediate representation (shown here as an INSERT query) which includes the distinct cardinality and length of string for categorical / string attributes, and ranges for numerical attributes. This intermediate representation is then used to compute an optimized form based on the cost of interaction in the constrained setting, whereupon it is enriched with the ideal widget, page number and display height. This information along with the data distribution of each field is used to generate the five redesigned HTML pages, one of which is shown here.

al.'s GOMs keystroke level model for mobile touchscreen [45]. Even though typing can be modeled as a series of taps, the keyboard size is fixed in most devices and it takes more time to interact with these as opposed to radio buttons, checkboxes, etc. Note that our goal is not to merely model the task completion time but to model the *human effort* in a manner that allows for effective optimization of form layout.

2.2 Cost Function

Similar to the keystroke-level GOMS model [41], for a particular field, the input cost is the weighted sum of the tapping cost, the sliding cost, the typing cost and the scrolling cost. The cost for field i is thus:

$$cost_i = tap_i \cdot w_{tap} + slide_i \cdot w_{slide} + type_i \cdot w_{type} + scroll_i \cdot w_{scroll} \quad (1)$$

where:

- tap_i is the number of taps.
- $slide_i$ is the range of the slider, since as the range gets larger it becomes more difficult to exactly select a value on the slider.
- $type_i$ is the length of string to be typed, estimated as average length of prior input values in database.
- $scroll_i$ is the total number of options divided by the number of options visible for field i.
- w_{tap},w_{slide},w_{type},w_{scroll} represent the respective weights. We describe tuning of weights in the evaluation section.

This model is similar in structure to prior work, we augment it with the ability to estimate $tap_i, slide_i, type_i, scroll_i$ from the database to inform form design.

2.3 Problem Formulation

Given a form with n fields, and groupings of these forms into pages, each with an associated cost as defined above, we want to find the set of pages that covers each form field exactly once while minimizing the overall cost. This is an instance of the set cover problem.

THEOREM 1. *Finding the minimum cost form layout is atleast as hard as finding a minimum weight mutually exclusive set cover (Proof in Appendix).*

Thus, a solution to SCP, which is a well known NP-Hard problem [27], is necessary for finding the minimum cost form layout. However, solutions to SCP are not enough for finding minimum cost form layout. Let U be the set of form fields required for layout. Let S, the set of subsets of fields, be the candidate pages $\{S{:}\forall s \in S, s \subset U\}$.

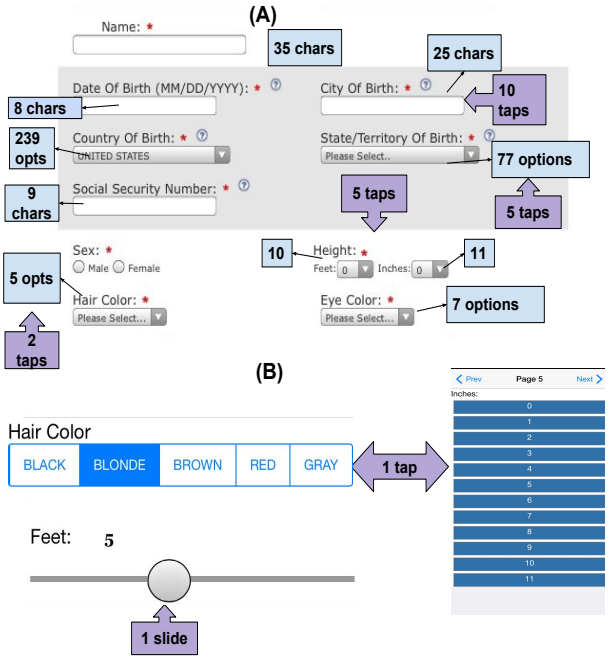

Figure 2: (a) The U.S. Passport Application form showing the maximum number of characters and options to scroll through for each field as described in motivating example. The average number of taps to make an entry is also shown. (b) Use of different widgets for the field reduces the amount of user interaction, thus reducing the time to fill the form.

The weight of each page is the sum of the costs of the fields in that page (Equation (1)). To generate S, we need to consider all possible subsets of form fields, along with widget assignment for each field. If we only consider the field groupings without widget assignment, then for n fields, S is the power set $P(U)$ of all fields, i.e., $|P(U)| = 2^n$ subsets. However, the widget assignment affects the interaction cost and the space taken by a field, which in turn affects the scroll cost of the remaining fields in the set. Then for k widgets, for each subset $p \in P(U)$, there are $k^{|p|}$ ways of assigning widgets to a subset:

$$|S| = \sum_{p \in P(U)} k^{|p|} \leq \sum_{p \in P(U)} k^n = 2^n k^n \in O(2^n k^n)$$

There are $O(2^n k^n)$ subsets, for which costs have to be calculated and solutions to SCP such as integer linear programming are infeasible. Other common optimization solutions such as dynamic programming also require all subsets to be considered, and generating the input subsets from the form fields in itself takes exponential time. Given such constraints, we look into building each page in a bottom up approach, adding each field in a greedy fashion, with our cost function automatically pruning infeasible page options. While Transformer can fall into local optima, it effectively reduces user effort while maintaining interactive latency, as we will demonstrate in Section 4.

3 THE TRANSFORMER SYSTEM

Transformer uses nine basic widgets depicted in Figure 3, that cover various datatypes and interactions. We first describe how the relational database aids in widget selection and then present our layout algorithm.

3.1 Database Constraints

The relational model allows us to automatically constrain the use of certain widgets, which we describe below (See Table 1 for explanations of terms).

3.1.1 Schema-Level Constraints. Tables with foreign-key dependencies denote a semantic relationship which can be exploited at the UI level. Each type of dependency provides separate functionality:

(1) *One-to-Many Join:* For relations with one-to-many joins, an accordion widget can be used, where the header is the value of the foreign-key attribute and the values of the second table appear upon expanding the header. If the second table has multiple attributes, subsequent form pages can be filtered so that they only show valid values for that attribute.
(2) *Many-to-Many Join:* An accordion widget can be used for a many-to-many join as well, however, a decision has to be made on which attribute is the header and which are the children. Transformer uses the cost function to automatically select the one requiring less overall human effort, i.e., the one with lower cardinality to be the header.
(3) *One-to-One Join:* One-to-one joined relations can automatically infer the value of the second attribute from the value of the first, hence only one of them needs to be displayed.

3.1.2 Data-Level Constraints. The datatype of the attribute allows Transformer to make further restrictions on widgets and keyboards.

(1) *Numeric:* Rangesliders can only be used for numeric fields or for attributes that have a one-to-one dependency with a numeric field, denoting an order. Further, if a text field is used for a numeric attribute, the device automatically switches to the numeric keyboard.
(2) *Date:* The datepicker widget is only compatible with attributes with date datatype.
(3) *Binary:* The toggle and checkbox widgets are only compatible with binary attributes. The checkbox widget can be used if multiple binary attributes can be grouped together, either because they belong to the same table or because there are foreign-key dependencies denoting semantic relations.

3.1.3 Cardinality Constraints. The cardinality of the field plays a major role in deciding the appropriateness of a particular widget. It dictates how many keystrokes are required if a text widget is used and what distance must be scrolled to if radio buttons or dropdowns are used. A segmented controller is only used if the cardinality of the field is small enough to fit in one line of the screen.

The above constraints are encoded in the cost function. Incompatible widgets are assigned high costs to denote infinity, so that they are not selected. This automatically prunes the search space, similar to a branch and bound algorithm. Note that unlike rule-based selections, these are restrictions on widget choices, not a direct mapping

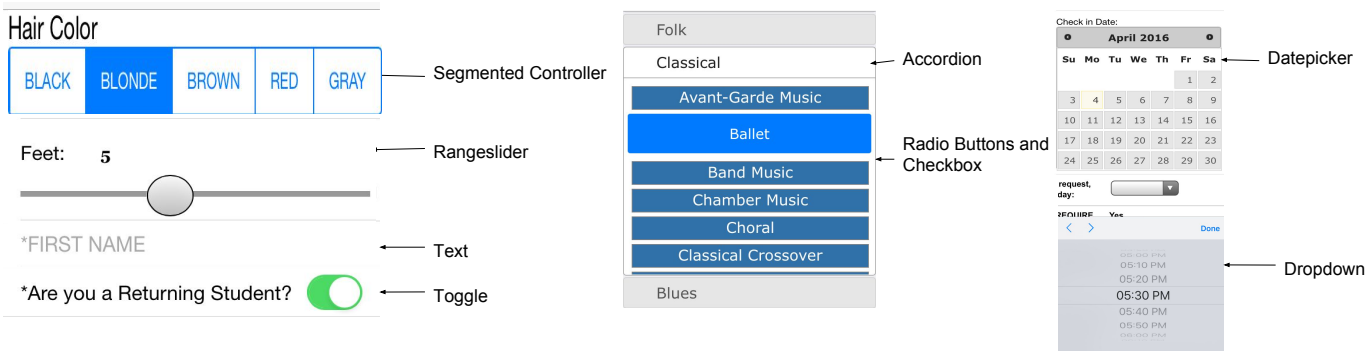

Figure 3: Widgets used by our system in the optimized forms. The segmented controller, datepicker and interactive label of the toggle allow the user to make a selection in one tap. The size of the buttons and handle of the slider make them easier to interact with and reduces scope of error.

Table 2: Description of cost calculations and restrictions for each widget. w_{tap},w_{slide},w_{type},w_{scroll} represent weights of each interaction type. The database allows us to gather all necessary information for costs and widget suitability.

Widget	Cost	Restrictions
Radio buttons	$w_{tap} + \frac{count}{disp\_ht} \times w_{scroll}$	None
Checkbox	$w_{tap} \times count + \frac{count}{disp\_ht} \times w_{scroll}$	Grouped Binary Fields
Rangeslider	$w_{slide} \times count$	Numeric data
Datepicker	$(count_{months} + count_{years}) \times w_{tap}$	Date datatype
Segmented controller	w_{tap}	None
Dropdown	$count \times w_{scroll} + 2 \times w_{tap}$	None
Accordion	$(count_{headings} + count_{options}) \times \frac{w_{scroll}}{disp\_ht} + 2 \times w_{tap}$	Foreign-key Dependency
Toggle	w_{tap}	Binary datatype
Text	$avg\ length\ of\ data \times w_{type}$	None

between data type and widget. For example, datepicker will only be considered for the date datatype, however a date attribute could use a dropdown widget if it has lower cost.

3.2 Form Layout Generation

We assume that the database attributes to be displayed are known. This can be automatically selected [39], specified by the user [70] or extracted from an existing web form for the database. Prior to selecting widgets for the attributes, we query the database to extract relevant information such as cardinality, datatype, etc. and produce an annotated list of relevant form fields. The form is then *transformed* and *rendered* as described below.

3.2.1 Transform. The transform procedure is shown in Algorithm 1. It takes as input, a list of form fields annotated with data type and range to calculate cost, and assigns a widget to each field along with a page number and the order it appears on for that page. The cost calculation for each widget and relevant constraints are summarized in Table 2. The cost calculation requires the field, widget and remaining page height. The field dictates number of attributes, widget dictates the type of interactions and page height determines the scroll cost.
The traditional greedy solution to the max cover problem, which adds the set that covers the most number of uncovered items with minimum weight is inapplicable, since calculating costs for all subsets is expensive, as mentioned in the previous section. Instead, our greedy algorithm builds the sets of fields (i.e., page groupings of fields along with the widget assignment) in a best-fit bottom up approach, at each step adding the field-widget combination with the minimum cost across all the available pages. It also compares the cost of adding it to existing pages against the cost of adding a new page. The cost of adding a new page adds the cost of an additional tap to the overall cost. The field is then added to the page which adds the minimum cost to the existing form cost. The field is annotated with widget choice and page number.

3.2.2 Render. During this step, we take the annotated list containing field, widget, page number and ordering to generate the HTML for each page. We represent `radio button` and `checkbox` widgets as buttons, with each option on a separate row with width equal to the width of the screen. For `radio` button, clicking on a button highlights it to indicate selection and unselects anything that was previously selected. The `checkbox` is similar as well, except it allows multiple selections. For `text` input, the entire width is available for typing and the labels are displayed as placeholder text on the input field. The handle of the `rangeslider` is increased to make it easier for the user to manipulate it. The `toggle` inputs are displayed on the same line as the label and clicking on the label toggles the selection as well.

Note that Transformer is meant to provide widget and layout dimensions for a specified screen size. It is up to the application

Algorithm 1 Transform Algorithm

page : max page number, increases to accommodate fields
ht : array containing remaining display height left for each page, decreases as fields are added to page
Cost() : Returns cost of field for given widget and height
MinHeight() : Returns minimum height required for field-widget combination

```
1:  procedure TRANSFORM(field, widgets, disp_height)
2:      page ← 1
3:      ht[1] ← disp_height
4:      for k = 1 → fields.length do                ▷ loop through fields
5:          f ← field[k]
6:          cost ← ∞
7:          for i = 1 → page + 1 do                 ▷ loop through pages
8:              for j = 1 → widgets.length do
9:                  cost ← Cost(f, widget[j], ht[i])
10:                 if cost < mincost then
11:                     f.wid ← widgets[j]
12:                     f.page ← i
13:                     f.ht ← MinHeight(f, f.wid)
14:                     mincost ← cost
15:                 end if
16:             end for
17:         end for
18:         ht[f.page] ← ht[f.page] − f.ht
19:     end for
20: end procedure
```

developer to incorporate these in responsive layouts or static web pages.

3.3 Runtime Analysis

For a form with n fields or a database with n columns, our redesigned form can have a maximum of n pages since a field cannot be split across two pages. So the transform phase will take $O(n^2)$ to iterate through each page for each field in the worst case, given a constant number of widgets. This is a significant improvement over the brute force algorithm that takes exponential time. The reduction is achieved by placing fields on the first page that they fit, which uses space inefficiently since a different ordering, with different widgets might take fewer pages and hence fewer taps. However, our results show that the greedy approach is adequate.

4 EXPERIMENTAL EVALUATION

In this section, we validate that our cost function accurately estimates the form completion time, show that our algorithm maintains an interactive performance and evaluate our system on whether it improves the usability of forms. To approximate the usability, we hypothesize the following:

(1) Forms that are easier to fill, take less time to complete.
(2) Forms with easier interactions have fewer errors, since the user is able to enter information more precisely.
(3) Finally, users should rate TRANSFORMER's forms as easier to use than the original ones.

In the following paragraphs, we describe the controlled user study employed to evaluate our hypotheses.

Parameter Tuning: Our cost function is the weighted sum of all interactions. To get insights on weights, we conducted a pilot study with 15 users on eight forms (Table 3) on an iPhone 4s, which has a screen size of 960px by 640px. We observed that on average, it took one second to fill each radio button. This is corroborated by Lee et al. [45], who find that it takes .33s to tap and 1.35s to make the decision to tap. We also find that time for text and dropdowns grew linearly with number of characters to type and number of options. The latter is corroborated by Sad and Poirier [65], who fit a linear regression to word index on a dropdown lists. Based on these, we assigned a weight of 1 for tapping, 7 for typing and 3 for scrolling. For a slider, a weight of .5 was assigned, because even though tapping is easier then sliding, in many cases we prefer a slider since it uses fewer pixels vertically. Again, these are weights for selecting appropriate widgets, and not necessarily for estimating interaction time. Our aim is to illustrate the use of the database in form layout generation.

Devices: The user studies were done on an iPhone 5s running iOS 9.3.1, with 64GB storage, 1GB RAM and a 1136px by 640px screen with a resolution of 326 pixels per inch. We conducted two sets of experiments, the first one using the entire phone screen and the second using only 300px by 300px to simulate a smartwatch screen. We did not do experiments on a smartwatch due to lack of resources and experienced smartwatch users, which makes it difficult to control for training bias [40]. While smartwatches currently do not support text input and are mostly used for notifications, it demonstrates the impact of our system as screens get smaller. We focus our experiments on phones and watches, since these provide the most constrained interaction and where optimization is most critical, as opposed to desktops and tabletops.

Dataset: We evaluate our algorithm on a combination of eight real world data collection and query forms, selected to capture a variety of activities, widgets and lengths. Table 3 gives a brief description of the forms. For each form we created a database corresponding to the form attributes and filled it with synthetic data having uniform distribution. Each form was redesigned manually by the authors and with TRANSFORMER, giving three versions of each form for a total of 24 forms for each screen size. We use manual to refer to the manually optimized forms, and automated to refer to those optimized by TRANSFORMER. To control for motor memory and other factors mentioned in [52], the information filled in was standard across all users, and presented in the order it appeared in the forms.

Participants and Methods: We recruited 30 users, 15 for smartphone (7 male, median age 22) and another 15 for simulated smartwatch (7 male, median age 22), consisting of graduate and undergraduate students from the University Campus for a within-subject user study. In order to be eligible for the study, they had to be smartphone users. Figures 4 and 5 compare form completion times by expertise and phone owned by user respectively. The improvement in time is mostly consistent across user types, i.e., a two-tailed t-test did not reveal any significant difference in results. Each user filled

Table 3: Forms Used in Experiments

Form	Title & Field Description
A	**Hilton Reservation:** 1 text field, a datepicker, buttons for adding rooms (up to 26) and 2 dropdowns with 4 options [5].
B	**United Airlines:** 1 radio button, 2 text fields, a checkbox, 3 dropdowns with 3,12 and 30 options. The number of travelers is a dropdown with 8 categories, each has a stepper that goes up to 8 [8].
C	**Library Advanced Search:** 37 checkboxes, 5 dropdowns, 4 with 4 options and one with 22, 1 text field [6].
D	**Music Search:** The genres have 20 options with each having sub-options, the largest number being 302 in the form of checkboxes, 2 dropdowns with 96 options each, 2 radio buttons and a text field [1].
E	**Roommate Matcher:** 5 text fields, 8 dropdowns with 2,2,5,4,3,3,4,4 options, 3 radio button fields and 7 checkboxes and requires scrolling [3].
F	**Maintenance Request:** 4 text fields, 2 radio buttons and 5 dropdowns with 8, 2, 323, 105 and 37 options, and a pop up datepicker [4].
G	**U.S. Passport Application:** 9 text fields, 1 radio button field, 6 dropdowns with 239, 77,10,11,5 and 7 options [9].
H	**Room Reservation:** 4 text fields, 2 pop up datepickers, 4 radio button fields, 5 dropdowns with 35, 5, 96, 96 and 7 options [7].

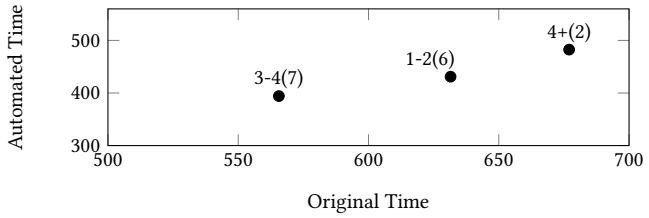

Figure 4: Completion times by daily hours spent on phone for smartphone study participants. Reduction in time is slightly more for expert users.

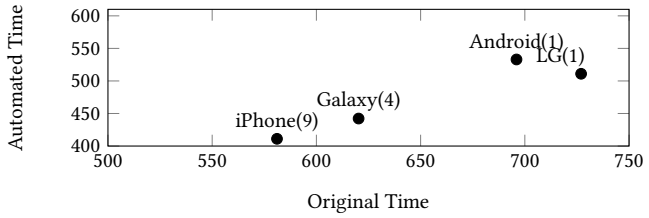

Figure 5: Comparison of form completion times by phone owned by user for smartphone study participants. Time reduction is consistent.

in the same set of 24 forms, however the order in which they filled the original version, automated redesign and manual redesign for each form was randomized to account for learning bias and was decided before we met with the user, to prevent selection bias [40]. For the experiments, we met with each user individually as follows:

(1) In a prestudy, users were asked about their phone usage.
(2) For each form, users were given the same printed information and asked to enter it on the phone provided. The printed information appeared in the same order as in the forms, to avoid any time loss in finding information.
(3) Users were asked to say out loud any time they used backspace or had to make a reselection when filling in data, as these counted as errors. The study conductor also watched to keep track of errors.
(4) Users were timed with a stopwatch from the moment they started typing till they hit submit for each form.
(5) After filling the form, the user was asked to rate the difficulty of the form. This, along with the time it took for us to load the next form, provided a two minute break to the user, to counter fatigue effects, before they repeated the procedure for the next form.

The smartphone experiments took 60 minutes, while the smartwatch ones took 35. For simulated watch experiments users were asked to ignore any text inputs, since smartwatches currently do not support this. When optimizing forms, we disabled the text widget on Transformer so that fields that required text input were not shown on the form. Due to challenges in rendering and availability of forms in an overly constrained size, we were unable to render the original form B in the $300px \times 300px$ variant. As a result, we compared optimized B on watch screen size against the original rendered on the phone as a best case scenario: it would take at least as long[1] to fill these forms on a watch screen as it would on the phone, if not longer.

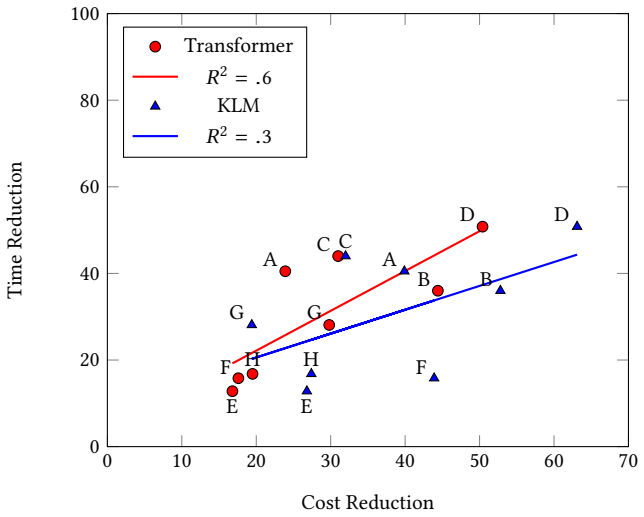

Figure 6: Cost function Validation for Smartphone Study: Change in time vs. change in cost between original and automated forms, using Transformer cost and the KLM cost. Our cost is a close estimation of the completion time, with Pearson correlation coefficient of .6.

[1] as empirically verified by our experiments

Table 4: Smartphone Result Summary: Relative difference of averaged times and ratings between original and automated forms of 15 users along with p-values of two tailed paired t-tests for smartphone study. Assuming normality, the time decrease was statistically significant at alpha level of .05 for most forms. Starred items had a medium Cohen effect while the rest had a large effect. Absolute values shown in Figures 8 and 12.

Form	Time Decrease (%)	p-value	Rating Increase (%)	p-value
A	40.5	0.00	5.5	0.28
B	35.93	0.00	42.9	0.00
C	44	0.00	70	0.00
D	50.79	0.00	46.1	0.00
E	12.76*	0.1	7.2*	0.41
F	15.85*	0.09	22.5	0.03
G	28.19	0.00	46	0.00
H	16.81	0.03	18.75*	0.06

Table 5: Smartwatch Result Summary: Relative difference of averaged times and ratings between original and automated forms of 15 users along with p-values of two tailed paired t-tests for smartwatch study. Starred items had a medium Cohen effect while the rest had a large effect. Time reduction is greater for forms which took longer to fill (D,E,F). Absolute values shown in Figures 9 and 13.

Form	Time Decrease (%)	p-value	Rating Increase (%)	p-value
A	19.85	0.03	34.12	0.03
B	16.75	0.05	9.76*	0.5
C	29.61	0.0	12.32*	0.5
D	57.64	0.0	93.33	0.0
E	44.92	0.0	33.4	0.0
F	44.53	0.0	71.66	0.0
G	17.95	0.03	33.95	0.06
H	26.36	0.0	20.81*	0.26

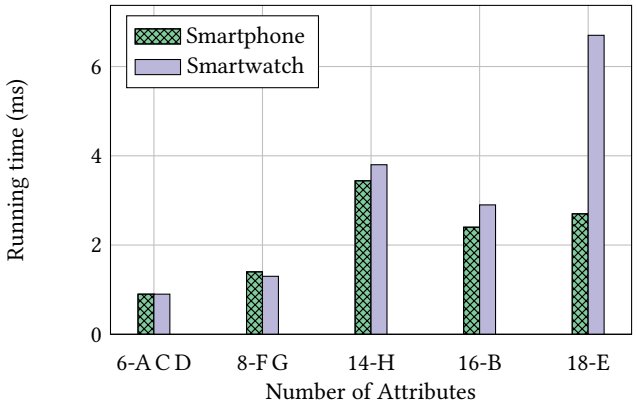

Figure 7: Running time of layout generation in milliseconds against number of attributes for smartphone and smartwatch. The corresponding forms are listed next to attribute sizes. For values with multiple forms, their averages are shown. Even though there is an increase in time with number of attributes, the time is under 10 ms for upto 18 attribute.

4.1 Cost Function Validation

In order to validate our cost function, we compare if the reduction in cost between the original and redesigned form is an accurate representation of the reduction in form completion times, using data from the smartphone user study. We also compared our cost with Lee et al.'s keystroke level model (KLM) for gaming interactions on a smartphone.

Figure 6 shows the comparison for each form, while Figure 8 has the absolute times. Our cost reduction is mostly proportional to the time reduction, with a Pearson correlation coefficient of .6. It does better than KLM, which has a correlation coefficient of .3, probably because those weights were tuned for gaming applications and not widget selection.

Form A is an outlier in our model; this is the room booking form for *hilton hotel* with fields for *check in* and *check out* date. Clicking on either field brings up a pop up datepicker on which both dates can be selected. Many users would accidentally change the *check in* date while trying to select the *check out* date, which is not captured by our cost model. Our model is generally accurate in estimating difficulty.

4.2 Performance

Figure 7 shows running time of widget assignment and layout generation for forms with varying number of form fields on the smartphone and the smartwatch. For forms with the same number of fields, their average is shown. Running time increases with number of form fields and decrease in screen size. This is because smaller screens require the form to be split into multiple pages and hence, more combinations are explored. Inspite of this, Transformer finishes in under 10 milliseconds for forms containing up to 18 fields, with no human input. This is a significant improvement over brute force which takes hours for even 10 fields.

4.3 Form Completion Times

Figures 8 and 9 show the average completion times of 15 users for the original, automatically optimized and manually optimized version for the 8 forms for the smartphone and smartwatch study respectively. This reflects an average reduction of 30–32% and a maximum reduction of 50–57% for smartphone and smartwatches, as shown in Tables 4 and 5. Assuming normality, a two tailed t-test showed that the results were mostly statistically significant.

The low decrease in Forms E and F can be explained by the fact that they had many dropdown fields which were replaced with scrollable interfaces, which was still easy enough to fill on the phone. On the smartwatch however, the difficulties caused by the small widget sizes are amplified and forms E and F have higher improvements. The absolute completion time (for both optimized and original forms) is lower for watches, due to the removal of text fields. Another result of this is that, forms with a fewer text fields, such as E, have higher improvements, as opposed to those

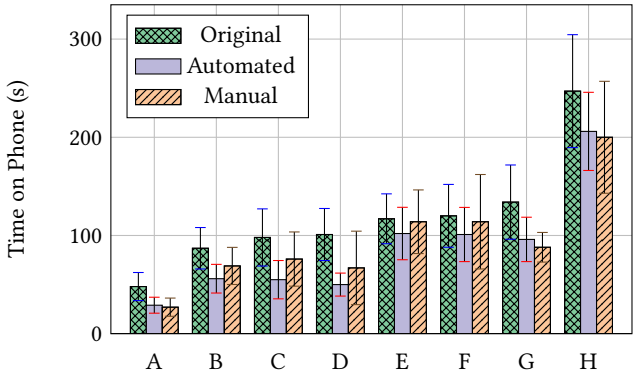

Figure 8: Average form completion time of 15 users for 8 forms on smartphone. Automatically optimized forms were always quicker to complete.

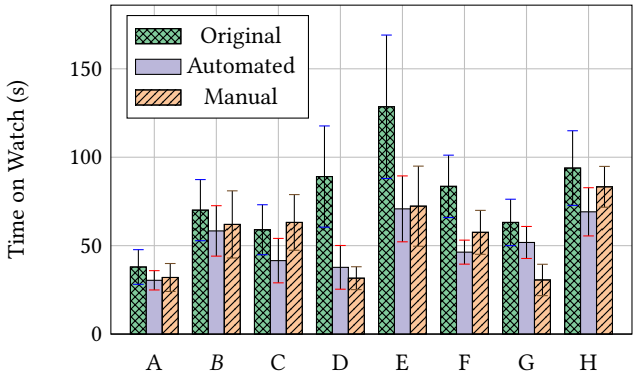

Figure 9: Average form completion time of 15 users for 8 forms for smartwatch sizes. The automatically optimized forms were always quicker to complete, even when compared against original forms which used the whole phone screens (B). Forms that took longer to fill (E) had larger time reductions.

with more, such as Form H. The difference between manual and automatic forms were not statistically significant.

4.4 Error Rates

Figures 10 and 11 shows the average error rates for each form on the phone and watch respectively. On the phone, the decrease in error rates is statistically significant for forms A and B. Most errors stemmed from text fields, as seen from the high error rates for forms G and H, which remained mostly unchanged in the optimized forms. Except for form D, the automated form had fewer errors than the original, showing that change in widgets can improve data quality on phone screens. On the constrained screen of the watch however, the larger widget sizes often led to accidental selection, which the user had to correct. This was not the case for the original forms where the user had to deliberately look for and select the widget. It is interesting to note that even though form E had the

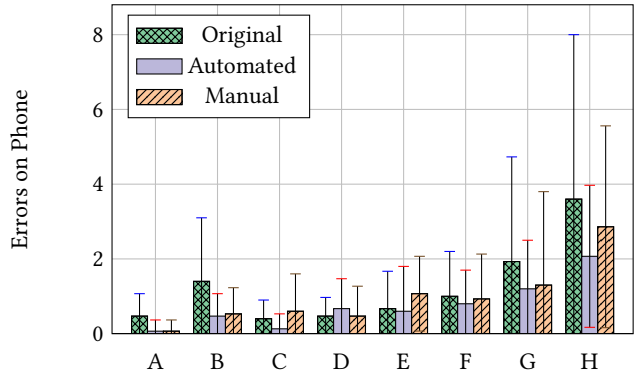

Figure 10: Average errors of 15 users for 8 forms on phone. While results are not statistically significant for all except A and B, the automated version had fewer errors for all but form D.

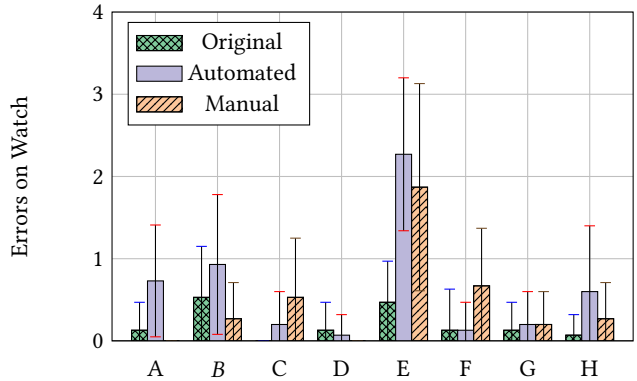

Figure 11: Average errors of 15 users for 8 forms on watch. The number of errors was higher on the optimized forms due to accidental selection.

highest number of errors, it still had a significant improvement in completion time, i.e., it was faster for the user to correct errors on the optimized form than to fill the original.

4.5 User Ratings

Figure 12 and 13 show the average rating of users on the usability of the form, 1 – very difficult, 10 – very easy. In many cases, the manual forms were rated as being easier to fill out mainly because the fields were grouped semantically during page splits and fields such as months were ordered meaningfully as opposed to the alphabetic ordering of the automated ones. For the most part, the automated forms were rated significantly higher, implying that simply changing widgets to improve interaction provides a noticeably better user experience, even without improving aesthetics.

4.6 Discussion

We find that redesigned forms by Transformer using data customized widgets provided significant reduction in time and was

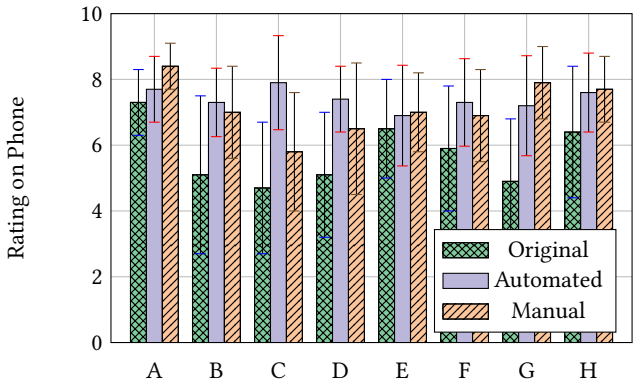

Figure 12: Average ratings of 15 users for 8 forms on the phone. A rating of 8 corresponded with the highest level of ease, while 1 corresponded with the lowest. The automated forms were rated higher for all.

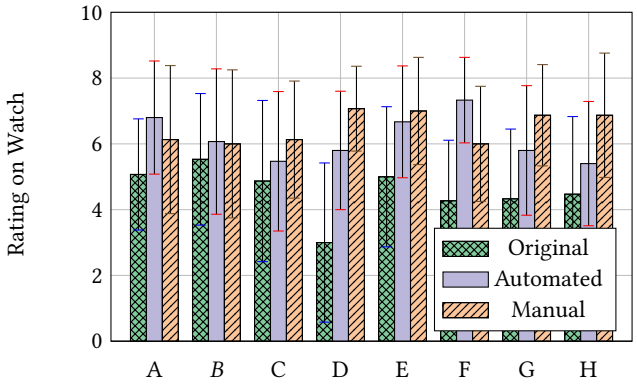

Figure 13: Average ratings of 15 users for 8 forms on the watch. Automated forms were usually rated higher than original.

preferred by the users over the original ones. We gathered the following insights from the user studies:

(1) The most time consuming element of the original forms is the small widget size which requires users to zoom in. Thus, it is important to specify a minimum widget size that the user can easily interact with as well as one that requires minimal scrolling. Optimizations such as making labels interactive for radio buttons and checkboxes, increasing the size of the handle of the rangeslider reduced time.
(2) After users zoom in, they need to scroll in both directions for two columnar forms and even if the fields are placed in a single column the users needed to scroll horizontally to look at the labels. Design considerations such as placing the label for text widgets as the placeholder in the widget as well as displaying labels on top eliminated this in the redesigned form. Vertical scrolling by itself does not add to the time if the users are scrolling in one direction, demonstrated by the low times of manual Form A.
(3) The slot machine style dropdowns in iOS are time consuming. Form D had the option of limiting results by specifying the start and end year. Both of these fields were represented as dropdowns ranging from 1920 to 2016. In the manual redesigned form each year this was split into three fields one for century having two options: 19 and 20 and the latter two for the year each ranging from 0 to 10. Even though users found this representation odd, they were able to fill in the form much quicker, with significant reduction in time. Further, in form B we transformed the dropdown for month with a range slider and radio button in the manual and automated ones respectively, which also helped decrease the completion time.
(4) Text fields require on average 20s and our high weight for it captures this. Specially for fields like name, the autocorrect feature of the phone comes into play and the user has to hit backspace many times leading to high error rates.
(5) Our cost does not capture ambiguity in the language and structure of the form, it merely evaluates the difficulty from a user interaction perspective, as can be seen from the underestimation of the original cost of form A.

5 RELATED WORK

Transformer is related to research in five areas: modeling human effort, data-driven interfaces, automated form generation and interaction on constrained interfaces.

Modeling Human Effort: Various models have been proposed for predicting human input such as Fitts', GOMS and ACT-R [15, 17, 28, 41, 68]. These models require extensive studies for estimating parameters and need to be conducted for each new input paradigm, i.e., touch, gestures. There have been different variations of these models for input modes [25, 38, 45], smartwatches [10], differently-abled users [11, 60] as well as for specific widgets [65]. These are orthogonal to our work, since they do not optimize form layouts, but can be used to inform weights in our model.

Data-Driven Interfaces: Declarative models include the mastermind project [69], AIDE [67],Vairamuthu et al.'s multi-criteria recommender system [72], MenuOptimizer [13], model-based [26, 59, 61] and rule-based UI generation [16, 62, 71]. While these systems use the database to link to widgets, the linking is specified through user defined rules, layout hints, parameters, etc. The database contents are not leveraged.

In the database community, data-driven approaches have been employed mainly to create query interfaces that provide cues on the database schema and eliminate reliance on querying language. These include keyword search querying[48, 49, 75], natural language querying [37, 47, 51] and forms [39, 70]. While these works use the data to help the user specify queries, our focus is on employing the data to improve the user interface for constrained displays. More recently, tools [14, 24, 43, 50, 63] are being created to augment the designers' abilities but they require a store of prior layout designs.

Automated Form Generation: The SUPPLE systems[30–33] frame adaptive layout generation as an optimization problem similar to us. However, they require the designer to provide a matching function to denote the appropriateness of a widget for every device. This is precisely the problem that Transformer solves through the use of the database backend. Other work in automated form creation include [19, 20, 73], which do not optimize input widgets. Constraint solving algorithms such as Cassowary [12] have been widely adapted to layout pages, but solving constraints for widgets and layout together takes exponential time.

Interaction on Constrained Interfaces: While research has focused on adapting web pages for constrained display devices [22, 36, 44, 64], no prior work has addressed specifically adapting forms for these devices. The techniques used are mainly applicable for browsing webpages and fail to address the unique challenges of form filling and leveraging the data to select widgets. A subarea of work has focused on smartwatch interactions. Study results in [18, 35] show that text input is feasible on a smartwatch since users average 20 to 30wpm(words per minute) on the small screen, indicating that watches could be used for forms in the future. In [53], Nebeling et al. take a different approach and employ crowdsourcing to allow the watch user to write long essays. Recently, alternative keyboards have been considered for text entry [21, 46, 55]. This line of work can be used in combination with ours to improve text entry on smaller devices.

6 CONCLUSION AND FUTURE WORK

In this paper we present Transformer, a system that designs an optimized form for any given display dimension by leveraging form database and minimizing a tunable cost function that models the user's difficulty. The database schema and distribution are used to estimate the number of user interactions required, which serve as input to the cost function. Our experimental evaluation showed that the forms produced by Transformer effectively reduced the completion time and were ranked higher than the original forms. In terms of future work, we would like to focus on two areas:

(1) *Autocompletion:* When implemented for text fields, instead of using the average length of the string, the length of longest common prefix would be used to calculate typing cost. This is mostly applicable for query forms, since in data entry forms, the user could be making an unseen entry where autocompletion might not work.
(2) *Semantic Grouping:* The manual forms often performed better due to their semantic structure. In the future, both the schema as well as label similarity can be analyzed for assigning semantic costs. We also hope to extend our cost model to capture aesthetics, such as finding the optimal distance between fields or choosing color schemes.

ACKNOWLEDGMENTS

This work is supported by the NSF under awards IIS-1422977, IIS-1527779, CAREER IIS-1453582.

REFERENCES

[1] AllMusic Advanced Music Search. https://www.allmusic.com/advanced-search/.
[2] Bootstrap. http://getbootstrap.com/.
[3] Boston University Roommate Matching Form. https://www.bu.edu/sth/admissions/enroll/finding-housing-in-boston/roommate-matching-form/.
[4] Facilities Maintenance Request Form. https://s2f.osu.edu/.
[5] Hilton Room Reservation. https://www3.hilton.com/en/index.html/.
[6] Library Advanced Search. https://osu.worldcat.org/advancedsearch.
[7] Room Reservation Request. https://ohiounion.osu.edu/meetings_events/space_requests/classroom_requests.
[8] United Airlines Flight Search. https://www.united.com/en/us/.
[9] U.S. Passport Application Form. https://pptform.state.gov/.
[10] S. Al-Megren. A predictive fingerstroke-level model for smartwatch interaction. *Multimodal Technologies and Interaction*, 2(3):38, 2018.
[11] S. Al-Megren, W. Altamimi, and H. S. Al-Khalifa. Blind flm: An enhanced keystroke-level model for visually impaired smartphone interaction. In *IFIP Conference on Human-Computer Interaction*, pages 155–172. Springer, 2017.
[12] G. J. Badros et al. The cassowary linear arithmetic constraint solving algorithm. *TOCHI*, 2001.
[13] G. Bailly, A. Oulasvirta, T. Kötzing, and S. Hoppe. Menuoptimizer: Interactive optimization of menu systems. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 331–342. ACM, 2013.
[14] G. Bhatia, Y. Fu, K. Kowalczykowski, K. W. Ong, K. K. Zhao, A. Deutsch, and Y. Papakonstantinou. Forward: Design specification techniques for do-it-yourself application platforms. In *WebDB*, 2009.
[15] X. Bi, Y. Li, and S. Zhai. Ffitts law: modeling finger touch with fitts' law. In *SIGCHI*, 2013.
[16] F. Bodart and J. Vanderdonckt. On the problem of selecting interaction objects. In *BCS HCI*, pages 163–178, 1994.
[17] S. K. Card. *The psychology of human-computer interaction*. CRC Press, 2017.
[18] B. S. Chaparro et al. Is touch-based text input practical for a smartwatch? In *HCI International 2015-Posters*. Springer, 2015.
[19] K. Chen, H. Chen, N. Conway, J. M. Hellerstein, and T. S. Parikh. Usher: Improving data quality with dynamic forms. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1138–1153, 2011.
[20] K. Chen et al. Shreddr: pipelined paper digitization for low-resource organizations. In *ACM Symposium on Computing for Development*, 2012.
[21] X. A. Chen et al. Swipeboard: a text entry technique for ultra-small interfaces that supports novice to expert transitions. In *UIST*, 2014.
[22] Y. Chen et al. Adapting web pages for small-screen devices. *Internet Computing, IEEE*, 2005.
[23] M. P. Couper and G. J. Peterson. Why do web surveys take longer on smartphones? *Social Science Computer Review*, 2016.
[24] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 845–854. ACM, 2017.
[25] K. El Batran and M. D. Dunlop. Enhancing klm (keystroke-level model) to fit touch screen mobile devices. In *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*, pages 283–286. ACM, 2014.
[26] J. Falb, S. Kavaldjian, R. Popp, D. Raneburger, E. Arnautovic, and H. Kaindl. Fully automatic user interface generation from discourse models. In *Proceedings of the 14th international conference on Intelligent user interfaces*, pages 475–476. ACM, 2009.
[27] U. Feige. A threshold of ln n for approximating set cover. *JACM*, 1998.
[28] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954.
[29] M. Fleshman, I. Argueta, C. Austin, H. Lee, E. Moyer, and G. Gerling. Facilitating the collection and dissemination of patient care information for emergency medical personnel. In *Systems and Information Engineering Design Symposium (SIEDS), 2016 IEEE*, pages 239–244. IEEE, 2016.
[30] K. Gajos and D. S. Weld. Supple: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*, pages 93–100. ACM, 2004.
[31] K. Gajos and D. S. Weld. Preference elicitation for interface optimization. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 173–182. ACM, 2005.
[32] K. Z. Gajos, D. S. Weld, and J. O. Wobbrock. Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, 174(12-13):910–950, 2010.
[33] K. Z. Gajos, J. O. Wobbrock, and D. S. Weld. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 231–240. ACM, 2007.
[34] K. Z. Gajos, J. O. Wobbrock, and D. S. Weld. Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 1257–1266. ACM, 2008.
[35] M. Gordon, T. Ouyang, and S. Zhai. Watchwriter: Tap and gesture typing on a smartwatch miniature keyboard with statistical decoding. In *Proceedings of the*

2016 CHI Conference on Human Factors in Computing Systems, pages 3817–3821. ACM, 2016.
[36] J. He et al. A flexible content adaptation system using a rule-based approach. *TKDE*, 2007.
[37] G. G. Hendrix et al. Developing a natural language interface to complex data. *TODS*, 1978.
[38] P. Holleis, F. Otto, H. Hussmann, and A. Schmidt. Keystroke-level model for advanced mobile phone interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1505–1514. ACM, 2007.
[39] M. Jayapandian and H. Jagadish. Automated generation of forms-based database query interface. *VLDB*, 2008.
[40] L. Jiang, P. Rahman, and A. Nandi. Evaluating interactive data systems: Workloads, metrics, and guidelines. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1637–1644. ACM, 2018.
[41] B. E. John and D. E. Kieras. Using goms for user interface design and evaluation: Which technique? *TOCHI*, 1996.
[42] J. Katriel. On a generalized recurrence for bell numbers. *Journal of Integer Sequences*, 2008.
[43] K. Kowalzcykowski, A. Deutsch, K. W. Ong, Y. Papakonstantinou, K. K. Zhao, and M. Petropoulos. Do-it-yourself database-driven web applications. In *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR?09)*. Citeseer, 2009.
[44] C. Kulkarni and S. Kiemmer. Automatically adapting web pages to heterogeneous devices. *CHI'11*, 2011.
[45] A. Lee, K. Song, H. B. Ryu, J. Kim, and G. Kwon. Fingerstroke time estimates for touchscreen-based mobile gaming interaction. *Human movement science*, 44:211–224, 2015.
[46] L. A. Leiva et al. Text entry on tiny qwerty soft keyboards. In *CHI*. ACM, 2015.
[47] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *VLDB*, 2014.
[48] G. Li et al. Efficient type-ahead search on relational data: a tastier approach. In *SIGMOD*, 2009.
[49] G. Li et al. Efficient fuzzy type-ahead search in tastier. In *ICDE*, 2010.
[50] Y. Li and T.-H. Chang. Auto-completion for user interface design, Aug. 16 2016. US Patent 9,417,760.
[51] Y. Li et al. Nalix: an interactive natural language interface for querying xml. In *SIGMOD*, 2005.
[52] MacKenzie et al. Text entry for mobile computing: Models and methods, theory and practice. *Human–Computer Interaction*, 2002.
[53] M. Nebeling et al. Wearwrite: Crowd-assisted writing from smartwatches. In *CHI*, 2016.
[54] B. Niamir. Attribute partitioning in a self-adaptive relational data base system. 1978.
[55] S. Oney et al. Zoomboard: a diminutive qwerty soft keyboard using iterative zooming for ultra-small devices. *SIGCHI*, 2013.
[56] A. Peytchev et al. Web survey design paging versus scrolling. *Public opinion quarterly*, 2006.
[57] A. Peytchev and C. A. Hill. Experiments in mobile web survey design. *Social Science Computer Review*, 2010.
[58] S. E. Poltrock and J. Grudin. Organizational obstacles to interface design and development: two participant-observer studies. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(1):52–80, 1994.
[59] R. Popp, D. Raneburger, and H. Kaindl. Tool support for automated multi-device gui generation from discourse-based communication models. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 145–150. ACM, 2013.
[60] A. Quezada, R. Juárez-Ramírez, S. Jiménez, A. Ramírez-Noriega, S. Inzunza, and R. Munoz. Assessing the target?size and drag distance in mobile applications for users with autism. In *World Conference on Information Systems and Technologies*, pages 1219–1228. Springer, 2018.
[61] D. Raneburger, H. Kaindl, and R. Popp. Strategies for automated gui tailoring for multiple devices. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, pages 507–516. IEEE, 2015.
[62] D. Raneburger, R. Popp, and J. Vanderdonckt. An automated layout approach for model-driven wimp-ui generation. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 91–100. ACM, 2012.
[63] D. Ritchie et al. D.tour: Style-based exploration of design example galleries. In *UIST*. ACM, 2011.
[64] V. Roto et al. Minimap: a web page visualization method for mobile phones. In *SIGCHI*, 2006.
[65] H. H. Sad and F. Poirier. Modeling word selection in predictive text entry. In *International Conference on Human-Computer Interaction*, pages 725–734. Springer, 2009.
[66] C. A. Sanchez and J. Wiley. To scroll or not to scroll: Scrolling, working memory capacity, and comprehending complex texts. *The Journal of the Human Factors and Ergonomics Society*, 2009.
[67] A. Sears. Aide: A step toward metric-based interface development tools. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 101–110. ACM, 1995.
[68] R. St Amant, T. E. Horton, and F. E. Ritter. Model-based evaluation of cell phone menu interaction. In *CHI*, 2004.
[69] P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools: The mastermind approach. In *Engineering for Human-Computer Interaction*, pages 120–150. Springer, 1996.
[70] L. Tang, T. Li, Y. Jiang, and Z. Chen. Dynamic query forms for database queries. *TKDE*, 2014.
[71] V. Tran, J. Vanderdonckt, M. Kolp, and S. Faulkner. Generating user interface from task, user and domain models. In *Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services, 2009. CENTRIC'09. Second International Conference on*, pages 19–26. IEEE, 2009.
[72] S. Vairamuthu, A. Anthoniraj, S. M. Anouncia, and U. K. Wiil. User interface design recommendations through multi-criteria decision analysis. In *Knowledge Computing and Its Applications*, pages 269–293. Springer, 2018.
[73] C. Vassilakis et al. Exploiting form semantics and validation checks to improve e-form layout. *International journal of Web engineering and technology*, 2005.
[74] Wired Magazine. In Less Than Two Years, A Smartphone Could Be Your Only Computer. http://www.wired.com/2015/02/smartphone-only-computer//.
[75] H. Wu et al. Seaform: Search-as-you-type in forms. *VLDB*, 2010.

7 APPENDIX

7.1 Proof of Theorem 1

The input of set cover problem (SCP) consists of a universe of elements U, a set of subsets S such that $\forall s \in S, s \subset U$ and a weight function that assigns a weight to each subset in S. A solution to this problem is a set of subsets $X \subset S$ such that each element of U belongs to exactly one element of X and weight of X is minimized over all subsets of S.

Reduce Form Layout (FLP) to SCP: Given a form with n fields $f_1, ..f_n$, we can map each field to an element in the universe of SCP, that is

$$U = \{f_1, f_2, ..., f_n\}$$

Each possible page layout can be mapped to sets in S of SCP:

$$\forall x \in \{1, .., n\}, s_i \in S, f_x \in page_i \implies f_x \in s_i, \tag{2}$$

The weight of each $s_i \in S$ is the sum of costs of each field in s_i as defined in Equation (1):

$$weight(s_i) = \sum_{f \in s_i} cost_f \tag{3}$$

A solution X to the minimum weight mutually exclusive SCP then gives a minimum cost form layout where each $s_i \in X$ is a page in the minimum cost layout. Since the sets in X are mutually exclusive and it is a cover, each field appears on exactly one page, and minimum weight implies minimum cost.

Reduce SCP to FLP: Given an SCP instance consisting of $U, S, weight$, each element in the universe becomes a form field in FLP.

$$F = \{f_1, f_2, ..., f_n\} = U = \{u_1, u_2, ..., u_n\}$$

Each subset in S in SCP can be mapped to a page layout in Pages in FLP, such that if u_1, u_2 are in the subset $s_i \in S$ then, fields f_1, f_2 are in $page_i \in Pages$:

$$\text{Pages} = \text{S}$$
$$\forall x \in \{1, .., n\} : f_x \in s_i \implies f_x \in page_i \tag{4}$$

Similarly the subset weights correspond to the page cost:

$$cost(page_i) = weight(s_i)$$

A solution X giving the minimum cost FLP is then a solution to minimum weight exact SCP where each $page_i \in X$ is corresponding to $s_i \in S$ is a part of the minimum cost layout.
Thus, a solution to SCP, which is a well known NP-Hard problem [27], is necessary for finding the minimum cost form layout.