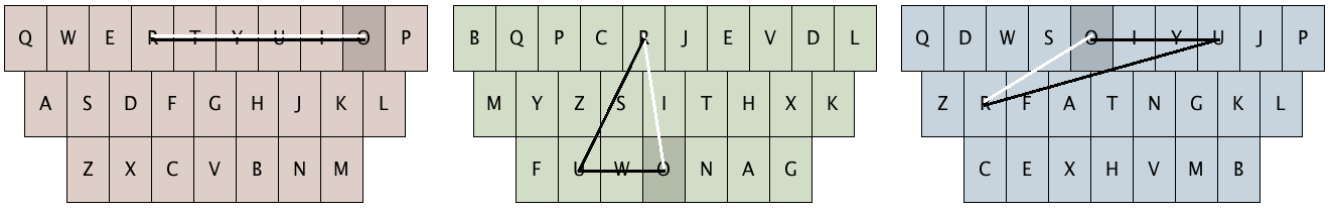
Optimizing Touchscreen Keyboards for Gesture Typing

Brian A. Smith[1,2*] **Xiaojun Bi**[2] **Shumin Zhai**[2]

[1]Columbia University
New York, NY, USA
brian@cs.columbia.edu

[2]Google Inc.
Mountain View, CA, USA
{xiaojunbi, zhai}@acm.org



(a) Qwerty (b) GK-D (c) GK-T

Figure 1. Keyboards optimized for gesture typing. The 'o' key is shaded to mark the beginning of the word gestures for "or" (white) and "our" (black). (a) The Qwerty keyboard suffers from the problem of gesture ambiguity. Many pairs of words (such as "or" and "our" shown here) share the same gesture on Qwerty. (b) The GK-D keyboard ("Gesture Keyboard—Double optimized") is the best compromise for gesture clarity and gesture speed. Here, the gestures for "or" and "our" are noticeably different. (c) The GK-T keyboard ("Gesture Keyboard—Triple optimized") is the best compromise for gesture clarity, gesture speed, and Qwerty similarity. Again, the two gestures are noticeably different.

ABSTRACT

Despite its growing popularity, gesture typing suffers from a major problem not present in touch typing: gesture ambiguity on the Qwerty keyboard. By applying rigorous mathematical optimization methods, this paper systematically investigates the optimization space related to the accuracy, speed, and Qwerty similarity of a gesture typing keyboard. Our investigation shows that optimizing the layout for gesture clarity (a metric measuring how unique word gestures are on a keyboard) drastically improves the accuracy of gesture typing. Moreover, if we also accommodate gesture speed, or both gesture speed and Qwerty similarity, we can still reduce error rates by 52% and 37% over Qwerty, respectively. In addition to investigating the optimization space, this work contributes a set of optimized layouts such as GK-D and GK-T that can immediately benefit mobile device users.

Author Keywords

Gesture typing; touchscreen keyboard optimization; text entry; word-gesture keyboard; shape writing

ACM Classification Keywords

H.5.2. [Information interfaces and presentation]: User Interfaces—Input devices and strategies.

INTRODUCTION

Gesture typing has gained large-scale adoption on mobile devices since its conception in the early 2000's [24]. Today, gesture typing can be found on all major mobile computing platforms in products such as ShapeWriter, Swype, SwiftKey, SlideIT, TouchPal, and Google Keyboard. Compared to touch typing (tapping), gesture typing offers several advantages. It supports a gradual and seamless transition from visually guided tracing to recall-based gesturing, allows users to approximate words with gestures rather than tapping each key exactly, and mitigates one major problem plaguing regular touchscreen typing: the lack of tactile feedback.

Despite these benefits, gesture typing suffers from an inherent problem: highly ambiguous word gestures. Bi, Azenkot, Partridge, and Zhai [2] showed that the error rate for gesture typing is approximately 5–10% higher than for touch typing. This problem occurs because when gesture typing, the input finger must inevitably cross unintended letters before reaching the intended one. The Qwerty layout itself further exacerbates this problem. Because common vowels such as 'u,' 'i,' and 'o' are arranged together on Qwerty, many pairs of words (such as "or" and "our") have identical gestures, and many others (such as "but" and "bit") have very similar gestures. Figure 1(a) shows the gestures for "or" and "our"—their superstrings "for" and "four" also have identical gestures. In fact, our analysis over a 40,000-word lexicon showed that 6.4% of words have another word with an identical gesture on Qwerty.

Given Qwerty's obvious problems, rearranging the keys to make word gestures more distinct should reduce the error

CHI 2015, April 18–23, 2015, Seoul, Republic of Korea.
ACM 978-1-4503-3145-6/15/04.
http://dx.doi.org/10.1145/2702123.2702357

* This work was done while Brian Smith was an intern at Google Inc.

rate when gesture typing. However, a layout optimized for gesture clarity (distinctness) might increase the length of each gesture (reducing typing speed) or may be difficult for users to learn. Many questions arise when deciding whether or not to introduce a new layout for gesture typing. For example, if the layout is exclusively optimized for clarity, to what degree will it improve in accuracy over Qwerty? What is the relationship between gesture clarity and gesture typing speed? Can we design an optimized layout similar to Qwerty in order to ease the learning process?

In this paper, we explore the layout optimization space related to gesture typing by applying a rigorous mathematical optimization. Our research not only deepens the understanding of the optimization space of gesture typing, but also contributes a set of optimized layouts that significantly outperform Qwerty (in terms of both gesture clarity and speed) and can immediately benefit mobile device users.

RELATED WORK

Stroke-Based Virtual Keyboards

Cirrin [14] and Quikwriting [17] are the first virtual keyboards designed specifically for word-level unistroke text entry. In both of those keyboards, users trace gestures that alternate from the center of a radial layout to one or more zones around the center (representing characters), with one articulation per character. However, since these gestures are completely determined by character layout without statistical pattern recognition, the letter layout has to be one-dimensional and most word-gestures defined on these layouts are very complex.

SHARK [24] and SHARK^2 [11] introduced gesture typing (also known as shape writing and the word-gesture keyboard paradigm) as we know it today. In these systems, users gesture words by swiping from letter to letter on a virtual Qwerty keyboard. The word gestures are much simpler than they are on Cirrin and Quikwriting, but since SHARK and SHARK^2 have no central "dead zone" for strokes to cross from character to character (and users must stroke over other characters instead), they suffer from an inherent ambiguity between word gestures (see Figure 1(a)). Even with sophisticated models for predicting users' intended words, Bi et al. [2] found that the error rate from gesture typing is 5–10% higher than that from touch typing.

Keyboard Layout Optimization

As has been widely published [18, 19, 21], Qwerty was designed to reduce jamming in mechanical typewriters by placing common digraphs (consecutive letter pairs) on opposite sides of the keyboard. Though this works well for two-handed or two-finger typing, researchers have long acknowledged that this is unsuitable for one-finger typing [9, 12]. There have been many proposed optimized keyboard layouts over the years for both bimanual [15] and unimanual typing [5, 6, 8, 13, 18, 20, 21]. Most of these layouts were optimized for touch typing, but the Square OSK layout [18] was optimized for stroking.

We should emphasize, however, that existing optimized layouts are predominantly optimized for typing speed (essentially minimizing finger travel distance), and that optimizing for word gesture clarity (as we optimize for along with gesture typing speed and Qwerty similarity) is an entirely different, and often conflicting, problem. As an example, the Dvorak layout arranges common letters in the home row to make bimanual typing faster [19], but this also makes word gestures more similar (and less unique, hurting gesture clarity) since many paths between keys become straight lines on the home row.

As another example, the ATOMIK [21] and Square ATOMIK [23] keyboards were optimized for speed with a bias for having keys appear in alphabetical order. Although these keyboards were tuned so that the gestures for 17 common words were short and memorable, they were not specifically optimized for gesture clarity. In fact, these keyboards predate gesture typing altogether. Other examples include Quasi-Qwerty [6], which was optimized for speed and familiarity, and the Sath keyboards [8], which were optimized for those metrics plus tap interpretation clarity for improved spell checking.

Few optimized layouts have gained widespread adoption. This is likely due to both learnability and the complexity of tapping input: users may type with one, two, or even ten fingers, and a good layout must accommodate each. However, the increasing popularity of gesture typing may offer a better chance at introducing new layouts since most users gesture words with one finger and our optimized layouts significantly improve both accuracy and speed over Qwerty.

OPTIMIZATION METRICS

Gesture Clarity

The gesture clarity metric is the most important metric in our optimization. The purpose of this metric is to measure how unique the word gestures on a keyboard layout are. We based the metric on the location channel in SHARK^2 [11] and represent each word's gesture as its *ideal trace,* the polyline connecting the key centers of the word's letters. We define the *nearest neighbor* of a word w to be the word whose ideal trace is closest to w's ideal trace. This is the word that is most likely to be confused with w when gesture typing, independent from the language model. The closer a word is to its nearest neighbor, the more likely its gesture will be misrecognized. The gesture clarity metric score for a given keyboard layout is simply the average distance (weighted by words' frequencies) between each word and its nearest neighbor on that keyboard layout:

$$\text{Clarity} = \sum_{w \in L} f_w \, d_w, \qquad (1)$$

$$\text{where } d_w = \min_{x \in L - \{w\}} d(w,x) \text{ and } \sum_{w \in L} f_w = 1.$$

L is a 40,000-word lexicon, f_w is the frequency of w, and d_w is the distance between w and its nearest neighbor. We compute the distance between two ideal traces w and x via *proportional shape matching*. Each gesture is sampled into N equidistant points, and the distance is simply the average of the distance between corresponding points:

$$d(w,x) = \frac{1}{N}\sum_{i=1}^{N} \|w_i - x_i\|_2 \quad (2)$$

Time Complexity Refinements

Since the gesture clarity metric compares the gestures of every pair of words to find each word's nearest neighbor, its time complexity is $\Theta(N \cdot |L|^2)$. Here, L is the number of words in the lexicon and N is the number of sample points in each word gesture. Its quadratic time complexity with respect to L stands in stark contrast to the time complexities of earlier optimization metrics (which are exclusively linear with respect to L), making optimization using it intractable. For our 40,000-word lexicon, there are nearly 800 million pairs of word gestures to compare for each keyboard layout that we examine during the optimization process.

To make the metric more tractable, we made two key algorithmic refinements. First, when searching for the nearest neighbor for each word, we only considered prospective neighbors that started and ended with characters that were located within one key diagonal of the word's starting and ending character, respectively. This is similar to the initial template-pruning step employed in SHARK^2 [11], where the distance threshold in this case is the diagonal length of a key. Second, we used a small number of gesture sample points N to represent each word's gesture. If N were too large, the computation would be very expensive. If N were too small, word gestures (especially longer ones) might not be represented properly, leading to incorrectly chosen nearest neighbors.

In order to see how small we could make N without affecting the integrity of our results, we performed a small experiment. First, we found each word's nearest neighbor on Qwerty using very fine sampling (N = 100). Then, we repeated this step for smaller values of N down to N = 20 and counted the number of nearest neighbors that were identical to the N = 100 case. Figure 2 shows the results. When the number of sample points is reduced to 40, 96.9% of the nearest neighbors are the same as they were before. We used this value for N in our algorithm.

Gesture Speed

The gesture speed metric estimates how quickly users can gesture type on a keyboard layout. We based this metric on the CLC model by Cao and Zhai [7]. The model (which stands for "curves, line segments, and corners") stems from human motor control theory, and was designed to predict the amount of time it takes for a person to make an arbitrary pen stroke gesture. To do this, the model partitions the gesture into segments, where each segment is a curve (with a constant radius of curvature), a straight line, or a corner (whose interior angle does not need to be 90°). The time that it takes for a person to gesture each type of segment is modeled with a different function. For line segments, the time is modeled with a power function that echoes how people tend to gesture faster with longer lines:

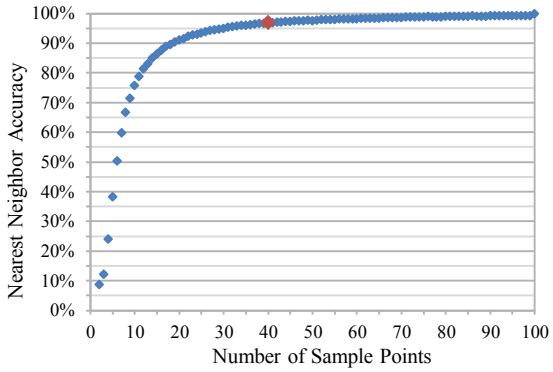


Figure 2. Word gesture neighbor sensitivity. The nearest neighbor that we find for a word depends on how finely the word gestures are sampled. Here, we show the percentage of nearest neighbors that are the same as when 100 sample points are used. The red dot signifies 40 points, the amount we used.

$$T(\overline{AB}) = m \cdot (\|\overline{AB}\|_2)^n. \quad (3)$$

Here, $\overline{AB}$ is a line segment, the output T is in milliseconds, $\|\overline{AB}\|_2$ is the length of $\overline{AB}$ in millimeters, and both m and n are constants (found to be 68.8 and 0.469 respectively in Cao and Zhai's original formulation).

A polyline gesture is simply a collection of individual line segments. The time to complete this type of gesture is modeled as simply the sum of the individual line segments' functions:

$$T(P) = \sum_{\overline{AB} \in P} T(\overline{AB}), \quad (4)$$

where P is the polyline and $\overline{AB}$ is a segment in the polyline. Although Cao and Zhai found that the angles between polyline segments (that is, of a polyline's corners) have an effect on gesture entry time, the magnitude of the effect was small: less than 40 ms per corner compared to 200–700 ms per segment. Hence, the model uses corners to delineate segments but omits their 40 ms contribution.

As with the gesture clarity metric, each word in the lexicon is represented as its ideal trace. To help compute the metric, we store a table of the weighted number of occurrences of each bigram in our lexicon. The weighted number of occurrences $o(i—j)$ of a bigram $i—j$ (for letters i and j) is calculated as follows:

$$o(i—j) = \sum_{w \in L} f_w \cdot (\#\text{ occurrences of } i—j \text{ in } w) \quad (5)$$

Here, L is the lexicon, w is a word in the lexicon, and f_w is the frequency of word w in L. Each bigram is represented by a different line segment in the CLC model. Hence, to estimate G, the average time it takes to complete a word gesture, we calculate the following:

$$G = \sum_{i,j \in \alpha} o(i—j) \cdot T(\overline{K_i K_j}) \quad (6)$$

Here, i and j are both letters in alphabet α, the set of lowercase letters from 'a' to 'z.' K_i and K_j are the key centers of the i and j keys, respectively, $\overline{K_i K_j}$ is the line segment connecting the key centers, and the function T is defined in Equation 3. Hence, G is measured in milliseconds.

The last step is to convert the gesture duration G into words per minute (WPM), a measure of typing speed. Doing so gives us our gesture speed metric score:

$$\text{Speed} = \frac{60{,}000}{G} \quad (7)$$

60,000 represents the number of milliseconds in one minute. When calculating the gesture typing speed of a keyboard layout, we do not consider the effects of the space bar or capitalization (and the Shift key). One of the key contributions of gesture typing is the fact that spaces are automatically added between word gestures, eliminating the need for 1 in approximately every 5.7 characters typed [24]. Moreover, most of today's gesture-typing systems apply capitalization and diacritics automatically.

We should also note that, because the CLC model omits the cost of gesturing corners and the cost of traveling from the end of one gesture to the beginning of the next, the calculated speeds generally overestimate the speeds at which users would actually type. Rick [18] proposed an alternative to the CLC model that is also based on Fitts's law, and although we ultimately chose to use the CLC model for our metric, we implemented Rick's model (without key taps for single-character words) to compare the models' behaviors. We found that Rick's model consistently output lower speed estimates than the CLC model, but that they both followed the same overall trend. More specifically, the mean (std. dev.) ratio between Rick's model's predicted speeds and the CLC model's predicted speeds for our final set of optimized layouts is 0.310 (0.004). After normalizing the metrics as described on the next page, the mean (std. dev.) ratio becomes 0.995 (0.016).

Qwerty Similarity

As has been thoroughly studied [18, 19, 21], the key obstacle to the widespread adoption of optimized layouts is the arduous process of learning the new layouts. The Qwerty similarity metric measures how similar a given keyboard layout is to Qwerty. By making a new layout more similar to Qwerty (and hence less alien to longtime users of Qwerty), we hope to bridge the gap between the short-term frustration of learning the new layout and the long-term benefits that the layout provides.

The metric is based on the constraint that Bi, Smith, and Zhai [6] used when creating the Quasi-Qwerty layout. In that optimization (which was for typing speed only), keys were not allowed to move more than one slot away from their Qwerty locations. Dunlop and Levine [8] later relaxed this constraint in their multi-objective keyboard optimization by using the total squared Euclidean distance between keys' positions and their Qwerty locations instead. Since a keyboard layout is essentially a grid of keys, we use the total Manhattan distance between keys' positions and their Qwerty locations to measure Qwerty similarity. Like Dunlop and Levine's metric, this allows more freedom than the hard constraint used by Quasi-Qwerty. However, unlike Dunlop and Levine's metric, individual keys are not punished so severely if they move far from their Qwerty locations. This allows us to consider layouts in which a few keys move very far from their Qwerty locations.

The Qwerty similarity metric for a given keyboard layout is computed as follows:

$$\text{Similarity} = \sum_{i \in \alpha} \left(\left| k_{i_x} - q_{i_x} \right| + \left| k_{i_y} - q_{i_y} \right| \right) \quad (8)$$

where i is a letter in alphabet α, the set of lowercase letters from 'a' to 'z,' and k_{i_x} and q_{i_x} are the x-indices of the i key on the given keyboard layout and Qwerty, respectively. Unlike K_i and K_j in Equation 6, which are points with units of millimeters, k_i and q_i are unit-less ordered pairs of integers that represent the 2D index of key i's slot in the keyboard grid. In most of today's touchscreen keyboard layouts, the second and third rows are offset from the first row by half of a key width. Hence, in order to properly calculate the Manhattan distance for this metric, we treat the second and third rows as if they are shifted to the left by another half of a key width so that the second row is left-aligned with the first row. The resulting representation of keyboard layouts is actually identical to the one used for creating Quasi-Qwerty [6]. The Qwerty similarity metric is the only one that uses this modified keyboard representation.

OPTIMIZATION PROCEDURE

We frame the problem of designing a touchscreen keyboard for gesture typing as a multi-objective optimization, where the three objectives are improving (1) gesture clarity, (2) gesture speed, and (3) Qwerty similarity. There are multiple ways of judging how well a layout meets these objectives. One way is to create a simple objective function that somehow combines the objectives' associated metric scores

(for example, by summing the scores in a linear combination). However, such an approach would force us to decide how much each metric should count for in deriving a single optimal layout, when in fact we are more interested in understanding the behavior of each of the metrics and the inherent tradeoffs between them.

As a result, although we still employ a simple objective function as part of our optimization's second phase, we use another approach called Pareto optimization for the optimization at large. Pareto optimization has recently been used to optimize both keyboard layouts [5] and keyboard algorithms [4]. In this approach, we calculate an optimal set of layouts called a *Pareto optimal set* or a *Pareto front*. Each layout in the set is *Pareto optimal*, which means that none of its metric scores can be improved without hurting the other scores. If a layout is not Pareto optimal, then it is *dominated*, which means that there exists a Pareto optimal layout that is better than it with respect to at least one metric and no worse than it with respect to the others. By calculating the Pareto optimal set of keyboard layouts rather than a single keyboard layout, we can analyze the tradeoffs inherent in choosing a keyboard layout and give researchers the freedom to choose one that best meets their constraints.

Our optimization procedure is composed of three phases, described in detail in the subsections below.

Phase 1: Metric Normalization

In the first phase, we perform a series of optimizations for each metric individually to estimate the minimum and maximum possible raw values for each metric. We then normalize each of the metric's scores in a linear fashion so that the worst possible score is mapped to 0.0 and the best possible score is mapped to 1.0. Normalizing the scores allows us to weight the metrics appropriately in Phase 2.

We use local neighborhood search to perform the optimizations. In order to more reliably find the global extrema instead of local extrema, we incorporate a simulated annealing process similar to the Metropolis random walk algorithm [10, 20]). Each optimization starts with a random keyboard layout using the same footprint as Qwerty and runs for 2,000 iterations. At each iteration, we swap the locations of two randomly chosen keys in the current layout to create a new candidate layout. If the new layout is better than the current layout, we keep the new layout with 100% probability. Otherwise, we only keep the new layout with a probability specified by a user-controlled "temperature." Higher temperatures increase this probability, and allow us to escape from local extrema.

In total, we performed 10–30 optimizations for each metric. We found that the range for the raw gesture typing clarity metric scores was [0.256 key widths, 0.533 key widths], that the range for the raw gesture typing speed metric scores was [50.601 WPM, 77.929 WPM], and that the range for the raw Qwerty similarity metric scores was [0, 148]. Qwerty's raw scores for the three metrics are 2.390 mm, 62.652 WPM, and 0, respectively.

Phase 2: Pareto Front Initialization

In this phase, we generate an initial Pareto front of keyboard layouts by performing even more local neighborhood searches. The searches are identical to the ones we perform in Phase 1, except this time we seek to maximize the score from linear combinations of all three metric scores. We use 22 different weightings for the linear combinations and perform roughly 15 full 2000-iteration local neighborhood searches for each weighting. The purpose is to ensure that the Pareto front includes a broad range of Pareto optimal keyboard layouts.

The Pareto front starts out empty at the very beginning of this phase, but we update it with each new candidate keyboard layout that we encounter during the searches (at each iteration of each search). To update the front, we compare the candidate layout with the layouts already on the front. Then, we add the candidate layout to the front if it is Pareto optimal (possibly displacing layouts already on the front that are now dominated by the candidate layout). The candidate layout is added whether it is ultimately kept in the particular local neighborhood search or not. This approach is similar to the one that Bi et al. [4] used to optimize keyboard correction and completion algorithms.

Phase 3: Pareto Front Expansion

In the last phase, we perform roughly 200 passes over the Pareto front to help "fill out" the front by finding Pareto optimal layouts that are similar to those already on the front. In each pass, we swap two keys in each layout on the front to generate a set of candidate layouts, then update the front with any candidate layouts that are Pareto optimal. This phase is similar to the optimization used by Dunlop and Levine [8]. However, by including Phase 2, we can ensure that all possible solutions are reachable without the need to swap more than two keys at a time.

Optimization Parameters

We based our optimization's keyboard representation on dimensions of the Nexus 5 [16] Android keyboard. Since most of today's touchscreen keyboards have very similar profiles, our results should be applicable to any touchscreen keyboard. Each key is represented by its entire touch-sensitive area (with boundaries placed between the center points of neighboring keys) and is 109 × 165 px (6.22 × 9.42 mm) in size.

Our lexicon consists of 40,000 words. Before starting the optimization, we converted words with diacritics to their Anglicized forms ("naïve" to "naive," for example), removed all punctuation marks from words (such as "can't"), and made all words completely lowercase. Since gesture typing systems automatically handle diacritics, capitalization, and punctuation marks within words, this should not hurt the integrity of our optimization.

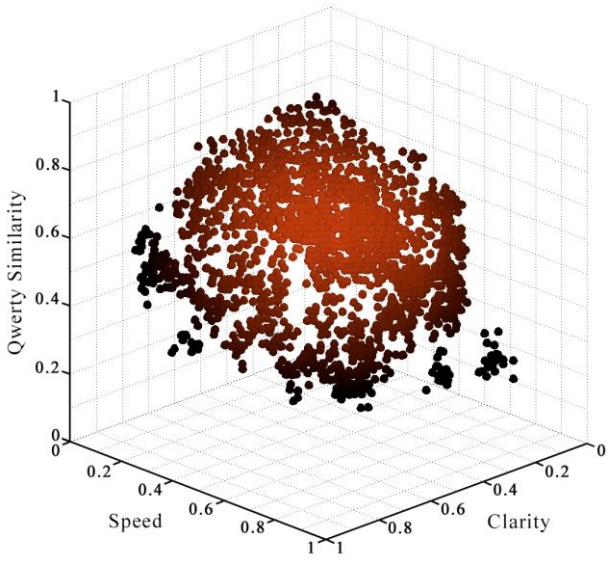


Figure 3. 3D Pareto front. The keyboard layouts with lighter colors are farther from the origin.

Optimization Runtime

Due to the complexity and scope of our work, it took four machines (with 32 threads apiece) running continuously over the course of nearly three weeks to obtain the results presented below.

OPTIMIZED KEYBOARD LAYOUTS

Figure 3 shows the final Pareto front of keyboard layouts optimized for gesture typing. Overall, the front is composed of 1,725 keyboard layouts chosen from the 900,000+ candidate layouts that we examined in all. No single layout on the front is better than all of the others—each layout is better than the others in some way, and the tradeoffs that are inherent in choosing a suitable layout from the front are reflected in the front's convex shape.

More specifically, the front can be viewed as a three-dimensional design space of performance goals that one can choose from for different usage scenarios. Layouts with high gesture clarity scores, gesture speed scores, and Qwerty similarity scores are more apt to exhibit lower error rates, expert-level gesture entry times, and initial gesture entry times (respectively) than those with low scores. However, since each layout on the front represents a compromise between these three goals, the choice of layout for a particular user or usage scenario depends on the relative importance of each goal. For example, a fast but less accurate user may prefer a layout biased towards clarity, while a user who gesture types very accurately may prefer a layout biased toward speed. Nevertheless, if we know nothing about users' preferences or wish to choose a layout that can best accommodate a wide variety of preferences, it is reasonable to use one that is in the middle of the convex surface (serving each goal on a roughly equal basis) as Dunlop and Levine did [8].

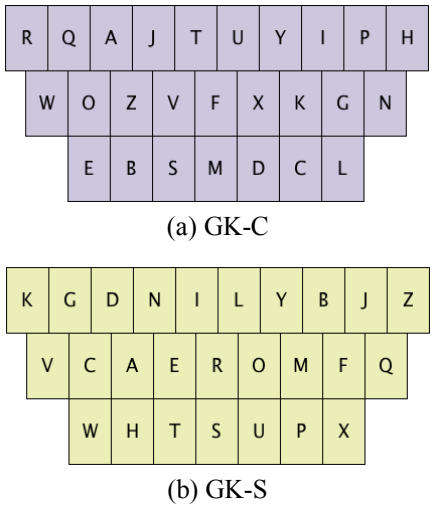


Figure 4. Single-optimized keyboard layouts. (a) Our GK-C keyboard ("Gesture Keyboard—Clarity") is optimized for gesture typing clarity only. (b) Our GK-S keyboard ("Gesture Keyboard—Speed") is optimized for gesture typing speed only.

We will now highlight layouts optimized for each of the three metrics as well as layouts that serve roughly equal combinations of metrics. These layouts may serve as useful references to researchers and designers, and will later (in the user study) help us test the effectiveness of our optimization and its associated metrics.

Single-Optimized Keyboard Layouts

Figure 4(a) shows GK-C ("Gesture Keyboard—Clarity"), the layout optimized exclusively for gesture typing clarity. Figure 4(b) shows GK-S, which was optimized exclusively for speed. The layout optimized for Qwerty similarity is simply Qwerty itself, and is shown in Figure 1(a).

Double-Optimized Keyboard Layout

Figure 1(b) shows GK-D (where the 'D' stands for "double-optimized"). This keyboard offers a roughly equal compromise between gesture typing clarity and gesture typing speed without regard to learnability (Qwerty similarity). To find this layout, we projected the three-dimensional Pareto front onto the clarity–speed plane to derive a 2D Pareto front between clarity and speed, then chose the layout on the 2D front that was closest to the 45° line. Figure 5 shows the 2D Pareto front and GK-D.

Triple-Optimized Keyboard Layout

Figure 1(c) shows GK-T, where the 'T' stands for "triple optimized." This keyboard offers a roughly equal compromise between all three metrics: gesture typing clarity, gesture typing speed, and Qwerty similarity. It is the one on the three-dimensional Pareto front that is closest to the 45° line through the space. As Figure 5 illustrates, it is possible to accommodate the extra dimension of Qwerty similarity without a big sacrifice to clarity and speed.

Layout	Gesture Typing Clarity		Gesture Typing Speed		Qwerty Similarity	
	Normalized	Raw (key widths)	Normalized	Raw (WPM)	Normalized	Raw (key slots)
Qwerty	0.489	0.391	0.441	62.653	1.000	0
Sath Trapezoidal [8]	0.568	0.413	0.704	69.843	0.730	40
GK-C	1.038	0.543	0.317	59.256	0.554	66
GK-S	0.283	0.334	1.000	77.940	0.311	102
GK-D	0.743	0.462	0.739	70.793	0.324	100
GK-T	0.709	0.452	0.704	69.830	0.716	42
Square ATOMIK [21]	0.362	0.356	0.878	74.591	N/A	N/A
Square OSK [18]	0.381	0.361	0.979	77.358	N/A	N/A

Table 1. Keyboard metric score comparison. Shaded rows signify previous layouts.

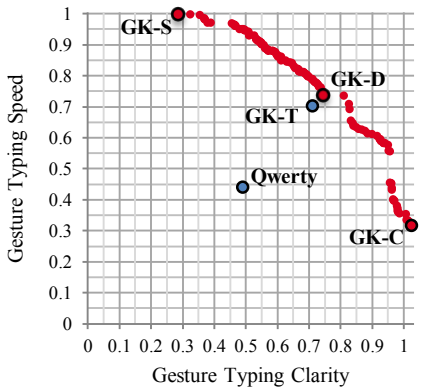


Figure 5. 2D Pareto front for gesture typing clarity and gesture typing speed. GK-D, our double-optimized layout, is the point on the front nearest the 45° line. Note that Qwerty is far worse in both dimensions than GK-D, and that GK-T (which accommodates yet another dimension) is only slightly worse on these two dimensions than GK-D.

Discussion

Table 1 shows the metric scores for our optimized layouts as well as previous optimized layouts. Together, these optimized layouts give us a good understanding of what is possible in the optimization space for gesture typing.

First, we can improve gesture clarity by 38.8% by optimizing for clarity alone: GK-C's raw metric score is 0.543 key widths while Qwerty's is 0.391 key widths. Likewise, we also see that we can improve gesture speed by 24.4% by optimizing for speed alone (resulting in GK-S).

Second, the 2D Pareto front for gesture clarity and gesture speed (Figure 5) shows that these two metrics conflict with each other. It forms a roughly -45º line, indicating that optimizing for one leads to the decrease in the other. As GK-C and GK-S illustrate, the clarity metric tends to arrange common letters far apart in a radial fashion while the speed metric clusters common letters close together.

However, despite the conflict, it is possible to arrange common letters close together while keeping word gestures relatively distinct, achieving large improvements in both clarity and speed. In GK-D (our double-optimized keyboard), letters in common n-grams such as "*the*," "*and*," and "*ing*" are arranged together while the n-grams themselves are spaced apart. This arrangement offers a 17.9% improvement in gesture clarity and a 13.0% improvement in gesture speed over Qwerty.

Third, accommodating Qwerty similarity (as GK-T does) does little harm to gesture clarity or gesture speed. GK-T's gesture clarity is only 0.01 key widths lower than GK-D's, and GK-T's predicted speed is only 1 WPM lower than GK-D's. Meanwhile, GK-T's Manhattan distance from Qwerty is just 42 key slots, while GK-D's is 102 key slots.

Comparison with Previous Optimized Layouts

The key difference between our proposed keyboard layouts and previous optimized layouts is that our layouts are optimized for multiple gesture typing factors while previous layouts are predominantly optimized for tapping speed. As Table 1 shows, previous layouts such as Sath Trapezoidal [8], Square ATOMIK [23], and Square OSK [18] have high gesture speed scores but low gesture clarity scores.

USER STUDY

Since the main focus of this work is to computationally discover the optimization space for gesture typing, the conclusions that we have made so far are based on theoretical metrics. Of the three metrics that we established, only one (gesture speed) is based on a model that directly predicts its respective performance goal (in this case, words per minute). The others, gesture clarity and Qwerty similarity, do not directly measure their performance goals (error rate and learnability, respectively). Hence, we performed an empirical study to give us a sense of how the metric scores correspond to real performance and whether the optimization itself is effective.

Experimental Setup

In the study, participants gesture typed a set of 22 words with each keyboard layout using a Nexus 5 [16] smartphone in portrait mode. As in Bi et al. [5], participants had to gesture each word seven times in succession. We instructed

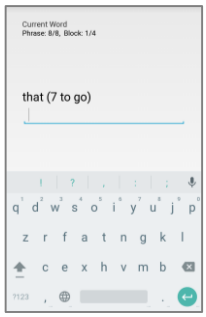


Figure 6. User study application. The layout shown is GK-T.

the participants to gesture as quickly as possible and ignore any errors, which, for us, achieved two goals. First, it allowed us to stress test the keyboard's gesture decoder by providing it very sloppy gestures (the resulting data is also more differentiable in evaluating accuracy). Second, it simulated a type of expert input behavior: entering words first and coming back to fix mistakes later.

Our study was a within-subject design that tested three keyboard layouts: Qwerty (our baseline), GK-D (the roughly equal compromise for clarity and speed only), and GK-T (the roughly equal compromise for clarity, speed, and Qwerty similarity). To conduct the experiment, we created Android implementations of GK-D and GK-T based on the Android [1] keyboard, and developed an Android application (Figure 6) to collect users' gesture typing data.

All participants started with Qwerty but used the other two layouts in alternating order. The first three words served as a warm-up phase to familiarize participants with the task (we did not collect their data), and the other 19 words are from the list proposed by Zhai and Kristensson [22]: "*the and you that is in of know not they get have were are bit quick fox jumps lazy*." These words cover all letters of the English alphabet and approximate both letter frequencies and digraph frequencies in English. They were divided into groups of four or five with short breaks in between.

To measure gesture typing accuracy, we compare each committed word with the respective requested word using a strict binary string equality comparison. The committed word is the word that appears after the participant lifts his or her finger from the screen and the keyboard algorithm applies any word corrections that it sees fit. To measure gesture entry times, we recorded either (1) the length of time from when a word was presented on the screen to when the word was committed (for the first repetition of a word), or (2) the length of time between when the last word was committed to when the current word is committed (for subsequent repetitions of a word).

The entry times for the first repetitions of words offer a rough (but by no means perfect) perspective of our keyboards' learnability, while the entry times for latter repetitions of the word are a rough estimate of expert-level entry times. The rationale for the latter is that by repeating the same word in a row, users will reach a stage where the

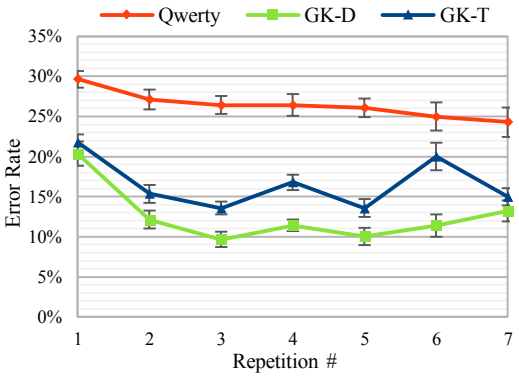


Figure 7. Error rates across 14 participants for Qwerty, GK-D, and GK-T. GK-D's and GK-T's average error rate is 52% and 31% less than Qwerty's, respectively. Error bars indicate standard errors.

input behavior is mostly governed by motor control ability, which reflects expert input behavior. However, this is only a limited proxy for the study of the complex learning process and expert-level typing performance at scale, which may require a longitudinal logging study of real keyboard use, notwithstanding privacy and other challenges associated with such methods.

A total of 14 volunteers (9 female, 5 male) participated in the experiment. 3 were age 18–25, 9 were 26–35, and 2 were 36–45. 8 of them primarily use Android smartphones and the rest iPhones. 13 were at least somewhat familiar with gesture typing, and 5 were at least somewhat familiar with alternative keyboard layouts. All of them were right-handed. Each experiment lasted less than an hour.

Experimental Results

Error Rate

Figure 7 shows participants' overall error rates with each layout and how those error rates changed as participants made successive repetitions of each word. The mean (std. dev.) error rate for Qwerty, GK-D, and GK-T were 26.4% (7.2%), 12.6% (6.6%), and 16.6% (6.2%), respectively. This means that the error rates for GK-D and GK-T were 52% and 37% less than Qwerty, respectively. The keyboard layout has a significant main effect on the overall error rate ($F_{2,26} = 35.46$, $p < 0.001$). Pairwise mean comparison over all repetitions showed that the differences were significant ($p < 0.01$) for every pair of keyboards except GK-D vs. GK-T. For Repetitions 2–6, however, the difference is significant ($p < 0.01$) for every pair of keyboards.

Initial Gesture Entry Time

Figure 8 shows how long, on average, it took participants to gesture words per repetition × layout. We noticed that participants often planned out their gestures in the first repetition, but resorted to motor memory in later repetitions. The mean (std. dev.) initial entry time was 2,655 ms (502 ms), 5,870 ms (1,190 ms), and 5,468 ms (1,140 ms) for Qwerty, GK-D, and GK-T, respectively. The keyboard

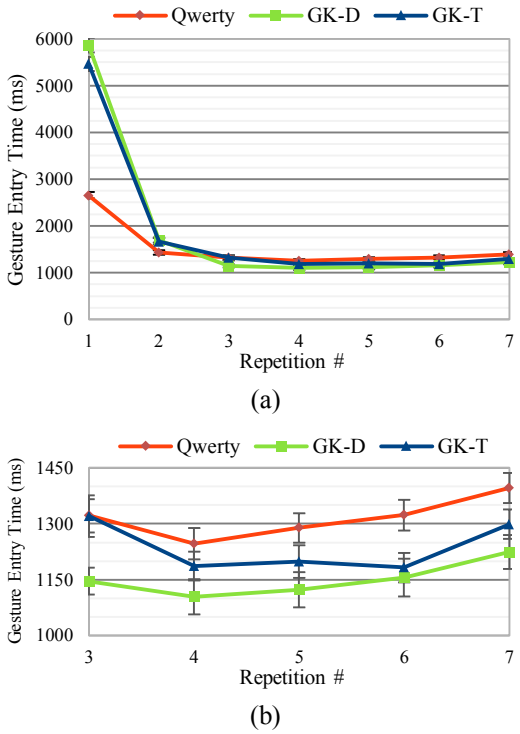


Figure 8. Gesture entry times across 14 participants for Qwerty, GK-D, and GK-T. Error bars indicate standard errors. (a) The initial entry time (Repetition 1) using GK-D and GK-T is over twice as long as it is using Qwerty. (b) For Repetitions 3–7 (approximating expert usage), the average entry time for GK-D and GK-T are 12.5% and 6.0% faster (respectively) than they are for Qwerty.

layout has a significant main effect on the initial entry time ($F_{2,26} = 75.26$, $p < 0.001$). Pairwise mean comparison showed that the differences were significant ($p < 0.01$) for each pair of keyboards except GK-D vs. GK-T.

Expert-Level Gesture Entry Time

Figure 8(b) shows the expert-level entry time (Repetitions 3–7) in detail. The mean (std. dev.) entry time in this case is 1,315 ms (300 ms) for Qwerty, 1,150 ms (333 ms) for GK-D, and 1,237 ms (310 ms) for GK-T. The keyboard layout has a significant main effect on the expert-level entry time ($F_{2,26} = 12.46$, $p < 0.001$). The expert-level entry time for GK-D and GK-T is 12.5% and 6.0% faster than that for Qwerty. Pairwise mean comparison showed the differences were significant ($p < 0.01$) for each pair of keyboards except GK-D vs. GK-T.

Discussion

The results from the user study lead to several findings, although we stress (as described earlier) that they are limited by the fact that our experiment was conducted in a single session. First, keyboards optimized for gesture clarity are more accurate than those without. The error rates for GK-D and GK-T are 52% and 37% less than Qwerty, respectively. Second, including both gesture typing clarity and gesture typing speed in the optimization process results in layouts that outperform Qwerty in terms of both accuracy and expert typing speed. Both GK-D and GK-T significantly outperform Qwerty in both of these metrics. Third, considering the Qwerty similarity metric has only minor effects on accuracy and speed. The differences we observed between the expert-level entry times for GK-D and GK-T were not statistically significant. Finally, the Qwerty similarity metric is not very effective in improving learnability. Though the mean initial entry time for GK-T was lower than that of GK-D, we did not observe a statistically significant difference between the two. This is likely due to the relative leniency of the Qwerty similarity metric compared to Quasi-Qwerty's hard constraint [6] and Dunlop and Levine's squared distance metric [8].

The findings also give us a better sense of how a layout's gesture clarity and Qwerty similarity scores correspond to real performance (recall that the gesture speed scores are based on an empirically derived model). For example, the gesture clarity score increase from 0.489 in Qwerty to 0.743 in GK-D (an increase by 0.254—see Table 1) corresponds to a decrease in the mean error rate from 26.4% to 12.6%. Yet, the Qwerty similarity score increase from 0.324 to 0.716 does not improve learnability as we have defined it, while the increase from 0.716 to 1.000 drastically does.

Still, we do not know the exact relationship between these two metrics' scores and the corresponding real-world performance measures. Conversely, the problem of finding metrics that empirically model gesture typing error rate and keyboard learnability remains to be solved. In the case of learnability, Quasi-Qwerty's constraint improves it [6] but cannot be used as a continuous model, and squared distance has not yet been shown to model it in its various degrees. Determining those relationships requires empirically testing many more layouts (using a variety of values for each metric score), and remains promising future work.

CONCLUSION

The present work, for the first time, defines a multidimensional optimization space for gesture typing (comprising gesture clarity, gesture speed, and Qwerty similarity) and systematically explores that space. In the process, we contribute a set of optimized layouts such as GK-D (optimized for both gesture clarity and gesture speed) and GK-T (optimized for gesture clarity, gesture speed, and Qwerty similarity) that can immediately benefit users. Though limited, our empirical study of these layouts led to the following findings.

First, optimizing the layouts for gesture clarity drastically improves gesture typing accuracy. By incorporating gesture clarity as an optimization dimension, GK-D and GK-T reduced error rates by 52% and 37% over Qwerty, respectively. Second, gesture clarity and gesture speed conflict with each other, but despite the conflict, incorporating both in the optimization process leads to

superior performance over Qwerty with respect to both metrics. GK-D and GK-T, for example, improved expert-level entry times by 12.5% and 6.0% over Qwerty, respectively. Third, Qwerty similarity as we have defined it has only a minor conflict with gesture clarity and gesture speed, but is not effective in improving learnability.

FUTURE WORK

Although the nature, size, and complexity of this work have surpassed its precedents in the literature [6, 8, 12, 18], many questions beyond the scope of this work require further research. These include further, larger, and longitudinal empirical studies of the multiple optimality dimensions. Further empirical investigation may redefine some or all of the optimality dimensions identified in this work in order to advance the gesture typing paradigm toward new shorthand writing systems that tolerate user errors, require minimal visual attention and motor effort, and remain easy to learn.

ACKNOWLEDGEMENTS

We would like to thank Kurt Partridge, other colleagues at Google, and the CHI reviewers for their valuable insights and contributions to this work.

REFERENCES

1. Android Developers. https://source.android.com.
2. Bi, X., Azenkot, S., Partridge, K., and Zhai, S. Octopus: evaluating touchscreen keyboard correction and recognition algorithms via "remulation." In *Proc. CHI 2013*, ACM Press (2013), 543–552.
3. Bi, X., Chelba, C., Ouyang, T., Partridge, K., and Zhai, S. Bimanual gesture keyboard. In *Proc. UIST 2012*, ACM Press (2012), 137–146.
4. Bi, X., Ouyang, T., and Zhai, S. Both complete and correct? Multiobjective optimization of a touchscreen keyboard. In *Proc. CHI 2014*, ACM Press (2014), 2297–2306.
5. Bi, X., Smith, B.A., and Zhai, S. Multilingual touchscreen keyboard design and optimization. *Human–Computer Interaction 27*, 4 (2012), 352–382.
6. Bi, X., Smith, B.A., and Zhai, S. Quasi-Qwerty soft keyboard optimization. In *Proc. CHI 2010*, ACM Press (2010), 283–286.
7. Cao, X. and Zhai, S. Modeling human performance of pen stroke gestures. In *Proc. CHI 2007*, ACM Press (2007), 1495–1504.
8. Dunlop, M.D. and Levine, J. Multidimensional Pareto optimization of touchscreen keyboards for speed, familiarity, and improved spell checking. In *Proc. CHI 2012*, ACM Press (2012), 2669–2678.
9. Getschow, C.O., Rosen, M.J., and Goodenough-Trepagnier, C. A systematic approach to design a minimum distance alphabetical keyboard. In *Proc. RESNA 1986*, RESNA (1986), 396–398.
10. Hastings, W.K. Monte Carlo sampling methods using Markov chains and their applications. In *Biometrika 57*, 1 (1970), 97–109.
11. Kristensson, P.O. and Zhai, S. SHARK^2: a large vocabulary shorthand writing system for pen-based computers. In *Proc. UIST 2004*, ACM Press (2004), 43–52.
12. Lewis, J.R., Kennedy, P.J., and LaLomia, M.J. Development of a digram-based typing key layout for single finger/stylus input. In *Proc. Hum. Fact. Ergon. Soc. 1999*, Sage Publications (1999), 415–419.
13. MacKensie, I.S. and Zhang, S. The design and evaluation of a high-performance soft keyboard. In *Proc. CHI 1999*, ACM Press (1999), 25–31.
14. Mankoff, J. and Abowd, G.D. Cirrin: a word-level unistroke keyboard for pen input. In *Proc. UIST 1998*, ACM Press (1998), 213–214.
15. Oulasvirta, A., Reichel, A., Li, W., Zhang, Y., Bachnynskyi, M., Vertanen, K., and Kristensson, P.O. Improving two-thumb text entry on touchscreen devices. In *Proc. CHI 2013*, ACM Press (2013), 2765–2774.
16. Nexus 5 – Google. http://www.google.com/nexus/5/
17. Perlin, K. Quikwriting: continuous stylus-based text entry. In *Proc. UIST 1998*, ACM Press (1998), 215–216.
18. Rick, J. Performance optimizations of virtual keyboards for stroke-based text entry on a touch-based tabletop. In *Proc. UIST 2010*, ACM Press (2010), 77–86.
19. Yamada, H. A historical study of typewriters and typing methods: from the position of planning Japanese parallels. *Information Processing 2*, 4 (1980), 175–202.
20. Zhai, S., Hunter, M., and Smith, B.A. The Metropolis keyboard—an exploration of quantitative techniques for virtual keyboard design. In *Proc. UIST 2000*, ACM Press (2000), 119–128.
21. Zhai, S., Hunter, M., and Smith, B.A. Performance optimization of virtual keyboards. *Human–Computer Interaction 17*, 2 (2002), 229–269.
22. Zhai, S. and Kristensson, P.O. Interlaced QWERTY – accommodating ease of visual search and input flexibility in shape writing. In *Proc. CHI 2008*, ACM Press (2008), 593–596.
23. Zhai, S. and Kristensson, P.O. Introduction to shape writing. *Text Entry Systems: Mobility, Accessibility, Universality*. Morgan Kaufmann, 2010. 139–158.
24. Zhai, S. and Kristensson, P.O. Shorthand writing on stylus keyboard. In *Proc. CHI 2003*, ACM Press (2003), 97–104.
25. Zhai, S. and Kristensson, P.O. The word-gesture keyboard: reimagining keyboard interaction. *Communications of the ACM 55*, 9 (2012), 91–101.