# Improvements to Keyboard Optimization with Integer Programming

**Andreas Karrenbauer**[1], **Antti Oulasvirta**[1,2,3]
[1]Max Planck Institute for Informatics, [2]Saarland University, [3]Aalto University

## ABSTRACT
Keyboard optimization is concerned with the design of keyboards for different terminals, languages, user groups, and tasks. Previous work in HCI has used random search based methods, such as simulated annealing. These "black box" approaches are convenient, because good solutions are found quickly and no assumption must be made about the objective function. This paper contributes by developing integer programming (IP) as a complementary approach. To this end, we present IP formulations for the letter assignment problem and solve them by branch-and-bound. Although computationally expensive, we show that IP offers two strong benefits. First, its structured non-random search approach improves the outcomes. Second, it guarantees bounds, which increases the designer's confidence over the quality of results. We report improvements to three keyboard optimization cases.

### Author Keywords
User interface optimization; Integer programming; Keyboard layouts; Branch-and-bound; Random search methods.

### ACM Classification Keywords
D.2.2. Software: Design tools and techniques: UIs

## INTRODUCTION
This paper advances the optimization of text entry methods for human performance. In particular, we look at the letter assignment problem, or the problem of keyboard layouts. This problem is known to be NP-complete [6] and computational solutions have received increasing attention. Active research areas include the optimization of keyboards for various terminals, languages, and user groups (e.g., [4, 9, 20, 2, 8, 25]).

Given the amount of interest, it is important to try to find improvements to optimization methods. Although UI optimization is presently focusing on keyboards, advances in the keyboard design case can be useful more generally to UI design problems concerned with layouts. Examples of layout optimization include menus [2] and widgets [10].

This paper addresses a significant limitation of the presently used class of methods. So far only *random search* methods have been used. They iteratively sample the solution space [22] and consult a model of users' performance weighted by a frequency distribution of their actions. Common methods include local search and simulated annealing. This "black box" approach is convenient, because good solutions can be found quickly and no assumption must be made about the objective function. The drawback is that such methods offer no guarantee for finding the global optimum, nor do they inform distance to it. Even *if* the global optimum was found, the designer can never know this for sure. Moreover, designers have no objective criterion for stopping the optimization and no confidence that the best-found design cannot be beaten.

This paper contributes by studying *exact methods* as a complementary approach. They use a structured (non-random) search approach that guarantees the optimal solution in finite time. Identifying the globally optimal design is desirable particularly for commonly used UIs where even small changes can have a large impact. Moreover, during search, exact methods estimate bounds. In other words, they inform the designer that the best-found solution is within a certain percent from the optimum. This increases confidence and pre-empts fruitless search.

Given these benefits, why were exact methods not previously used in HCI? Until a decade ago it was easy to ignore them because global optimization of assignment problems with more than 20 items was held impossible. The simplest formulation of the virtual keyboard problem with 26 keyslots has a search space of $4 * 10^{26}$. However, efficiency has improved from 1988 to 2004 by a factor of 5,280,000 [5]. It is thus now possible to solve more meaningful problems also in HCI.

The present paper is a proof-of-concept study targeting keyboard design. Although our longer-term goal is to develop exact methods for a wider class of UI design tasks, keyboards offer a solid starting point for our research. It is the most extensively studied case of UI optimization and allows benchmarking the benefits. Moreover, even slight improvements in keyboard layouts not only improve productivity [4] but can also enhance ergonomics and learnability [17]. Furthermore, the case is analytically favorable, because the task and the objective function involved are relatively well-understood and straightforward.

We formulate various letter assignment tasks as *integer programming* (IP) problems [24]. Since there is no unique formulation, we investigate a few models w.r.t. trade-offs such as computational efficiency, memory requirements, and the strength of bound. To solve them, we build on a standard so-


*UIST 2014*, October 5–8, 2014, Honolulu, HI, USA.

ACM 978-1-4503-3069-5/14/10 ...$15.00.
http://dx.doi.org/10.1145/2642918.2647382

lution called *branch-and-bound*. Here the idea is to structure the search space as a tree and construct an "oracle" that provides bounds that tell which subtrees (branches) to disregard (prune). We show techniques for branching and bounding that are sufficiently efficient to improve over the state-of-the-art in keyboard optimization. We then demonstrate these capabilities across three keyboard optimization cases from previous literature. Our approach is directly usable in the available IP solvers such as CPLEX and Gurobi.

## BACKGROUND: THE LETTER ASSIGNMENT PROBLEM

Given $n$ letters and $n$ keyslots, the task of the letter assignment problem is to minimize the average cost $c_{k\ell}$ of selecting letter $\ell$ after $k$ weighted by the bigram probability $p_{k\ell}$:

$$\min \sum_k \sum_\ell p_{k\ell} \cdot c_{k\ell} \tag{1}$$

It is an instance of the Koopmans-Beckman Problem [15, 6], which is NP-hard. As we will see later, this problem can be modeled with a quadratic function on binary variables. Thus, it belongs to a broad class of problems called *quadratic assignment problem* (QAP) where the goal is to minimize the total pair-wise interaction cost [22, 24].

Except for a few papers using heuristic cost functions (e.g., [9]), the *Fitts-bigram energy* is used (see also [4]). Here, the probabilities $p_{\ell k}$ are given by a bigram distribution of representative corpus of text. The cost $c_{\ell k}$ is given by Fitts' law (e.g., [19]):

$$MT = a + b \log_2 \left( \frac{D_{k\ell}}{W_\ell} + 1 \right), \tag{2}$$

where $D_{\ell k}$ is distance between the two keys containing $\ell$ and $k$, and $W_\ell$ is the width of the button that contains $\ell$. Fitts law is best understood as an estimate of movement performance achievable by skilled users. The cost for repeating the same letter is assumed to be constant, i.e. $c_{\ell\ell} = c$. Since the optimum assignment is not affected by the choice of $a, b, c$, the task collapses to distance minimization [25]. We later also consider a task that is not about distance minimization: touch typing with ten fingers on a physical keyboard.

### Previous Work Using Random Search Methods

Previous work has exclusively used random search methods to solve this problem. These methods iteratively randomly sample the neighborhood of a given solution. The variants differ in sampling strategy and update procedure.

**1)** *Local search* starts from a random position in the search space and randomly samples its neighborhood. A recent example is the 26 letter trapezoidal keyboard of Dunlop et al. [8]. **2)** *Metropolis–Hastings algorithm* is a variant of local search that uses a probabilistic decision criterion that allows it to escape from local minima. A known example is the hexagonal keyboard of Zhai et al. optimized for stylus tapping on PDAs [25]. **3)** *Simulated annealing* is an adaptation of the Metropolis–Hastings algorithm that uses a cooling schedule that is an analogue with thermodynamic free energy. A recent example is the split keyboard of Oulasvirta et al. optimized for two-thumb typing performance [20]. **4)** *Genetic algorithms* represent layouts as "genotypes" that are competitively evolved by mutations and cross-overs. A recent example is the physical keyboard layout of Goettl et al. optimized using a heuristic cost model [11]. **5)** *Ant colony optimization* is based on a biological metaphor of an ant colony cooperatively foraging for food. The method was recently used to improve the hexagonal keyboard of Zhai et al. [2].

## OVERVIEW OF THE APPROACH

We formulate the QAP as an integer programming (IP) problem in order to then study it using a standard solution: branch-and-bound. The QAP has been studied extensively in the discrete optimization community. The letter assignment problem was among the earliest problems studied within this context [21]. We here provide an overview of the approach and present technical details of our solution in the next section. A detailed review of the work is beyond the scope of this paper. The interested reader is referred to the survey of Loiola et al. [18].

The basic idea of IP is to describe the design space by *equations and inequalities*, also called constraints. The task of an Integer Programming Problem is to decide whether there is a solution that satisfies all constraints and that assigns only integers to the variables. This problem is also called Integer Feasibility Problem. Already the restriction to linear constraints and moreover to binary variables is NP-complete.[1] Though this sounds discouraging, it also shows the great modeling power that comes with IP: there are IP models for every problem in NP. Moreover, there are efficient solvers that solve integer programs much faster than a brute force search in practice. A key concept is a divide-and-conquer strategy called *branch-and-bound*, which we will briefly discuss after modeling the letter assignment problem as an IP. To this end, we present how to model keypress times as a quadratic function.

### Formulation of the QAP as IP

The letter assignment problem can be reformulated as an IP problem with a quadratic objective function [6]. Let $x_{\ell j}$ be a binary variable which is 1 if letter $\ell$ is assigned to keyslot $j$ and 0 otherwise. Hence, we obtain $c_{k\ell} = \sum_i \sum_j t_{ij} \cdot x_{ki} \cdot x_{\ell j}$, where $t_{ij}$ represents the movement time from key $i$ to key $j$, e.g. as in Eq. (2). By substituting $c_{k\ell}$ in Eq. (1), the goal becomes to find the

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{\ell=1}^{n} p_{k\ell} t_{ij} x_{ki} x_{\ell j} \tag{3}$$

subject to

$$\sum_{\ell=1}^{n} x_{\ell j} = 1 \qquad \forall j \in \{1, \ldots, n\} \tag{4}$$

$$\sum_{j=1}^{n} x_{\ell j} = 1 \qquad \forall \ell \in \{1, \ldots, n\} \tag{5}$$

$$x_{\ell j} \in \{0, 1\} \qquad \forall \ell, j \in \{1, \ldots, n\} \tag{6}$$

[1] Note that the keyboard optimization problem is trivially feasible: The hardness comes in when we ask whether there is a layout of at most a certain cost.

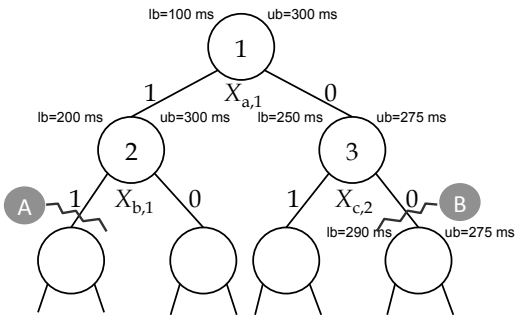



Figure 1: Branch-and-bound divides the search space (all keyboard layouts) into a tree (branch) that can be pruned based on (A) infeasibility of solutions and (B) bounds. $lb$ and $ub$ refer to the best known lower and upper bounds for the subtree rooted at the corresponding node.

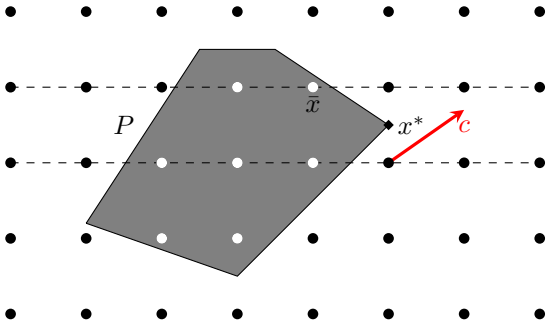



Figure 2: The gray area together with its boundary shows the linear relaxation, which contains the feasible integral solutions (white). $x^*$ is the optimum solution over the relaxation w.r.t. the linear objective function $c^T x$, whereas $\bar{x}$ is the optimum among all integral solutions.

In other words, each slot must contain exactly one letter due to the constraints (4) and each letter must be assigned to exactly one slot because of the constraints (5). The so-called *integrality constraints* (6) reflect the restriction to binary solutions as required at the beginning of this subsection.

### Branch-and-Bound

Branch-and-bound solves the QAP by performing an *implicit* exhaustive search. We imagine the search as a tree, where we make a decision at each node to partition the search space. Figure 1 provides an overview.

For the sake of presentation, we first describe a simple branching strategy, which fixes one variable in each iteration. At each node of the tree, we select a letter and assign it to an empty slot, e.g. letter 'a' to slot 1 at node 1 in Fig. 1. Such a decision is represented by one of the two children of a node, whereas the other child means the opposite, i.e. we forbid this letter-slot-combination for the corresponding subtree.

Suppose that we have already found a feasible binary solution with an objective value of $z$, for example by a random assignment of the letters to the keyslots, by some heuristics, or by branching down to a leave of the tree. If we now had an "oracle" that told us at some node that all feasible integral solutions in the subtree rooted at this node have objective values of *at least* $z$, then we could simply discard this entire subtree because it will not offer an improvement. This leads to two kinds of pruning that can be done: 1) based on infeasibility, e.g., two letters cannot be assigned to the same slot as shown in case A in Fig. 1, and 2) on bounds, e.g., as illustrated in case B in Fig. 1 where the lower bound is worse than the objective value of the incumbent.

It remains to show that there are effective methods to compute good lower bounds for the integer solutions in a subtree.

### Relaxation

There is a systematic approach to obtain lower bounds for an IP when the objective function and all constraints are linear. Before we show how to obtain such a setting in our case, we illustrate its benefits for branch and bound.

A system of linear constraints describes a closed convex set, which is bounded in our case (illustrated by the gray area in Fig. 2). The integral points are shown as small disks, which are white when they satisfy the linear constraints and thus correspond to feasible solutions for the IP. Since the set of feasible IP solutions is contained in the set of points that satisfy the linear constraints, the latter is called *Linear Programming relaxation* or LP relaxation. Moreover, this overestimation of the feasible solutions implies that the minimum of a function over the IP solutions is at least the minimum taken over the relaxation. Hence, an optimum solution over the relaxation yields a mathematical guarantee for the objective value of any IP solution. The difference between the optimum objective value of the IP and of the LP relaxation is called *integrality gap*.

Finding an optimum of the LP relaxation is a convex optimization problem, which can be solved efficiently, e.g. by Interior Point Methods (cf. Chapter 9 in [3]). If a variable of an optimum LP solution, say $x^*_\bullet$, is not integer, it can be used for branching, which splits the problem into two independent subproblems. That is, the set of feasible solutions is divided into disjoint pieces. In the example of Fig. 2, these are the parts on and below the bottom dashed line, on and above the top dashed line, and the strip between the two dashed lines, where the latter does not contain integer points in its interior and thus can be discarded in the search. This yields two new child nodes in the branching tree with the new constraints $x_\bullet \leq \lfloor x^*_\bullet \rfloor$ and $x_\bullet \geq \lceil x^*_\bullet \rceil$, respectively. If all variables are constrained between 0 and 1 as in the QAP, this amounts to setting the branching variable to either 0 or 1 in the respective subproblems. The interested reader is referred to the textbook [24] for further details about this method.

## LINEARIZATIONS

We present two linear formulations that are equivalent to the QAP. However, they differ when the integrality constraints $x_{\ell j} \in \{0, 1\}$ are relaxed to $0 \leq x_{\ell j} \leq 1$.

In a nutshell, the first formulation yields stronger lower bounds, but requires much more space and takes longer to solve in each node of the branch-and-bound tree. The sec-

ond one is much smaller and quicker to solve, but the lower bounds are very weak. We propose to solve both IPs in parallel to use the former to obtain good lower bounds, and the latter one to quickly explore the search space allowing the solver's built-in heuristics to improve the upper bound rapidly.

*Reformulation-Linearization Technique*

The basic idea is to substitute all non-linear terms by new variables and thus obtain a linear objective function. That is, we introduce $y_{ijk\ell} := x_{ki} \cdot x_{\ell j}$. To obtain linear constraints, Adam and Johnson [1] observed that $\sum_k y_{ijk\ell} = x_{\ell j} \cdot \sum_k x_{ki} = x_{\ell j}$. The intuition behind these additional constraints is based on the following observations. If $x_{\ell j} = 0$ then all products that contain $x_{\ell j}$ and thus all the corresponding $y$-variables must be 0 as well. Since slot $i$ contains exactly one letter, $y_{ijk\ell}$ is 1 for exactly one $k$ and 0 otherwise, provided that $x_{\ell j} = 1$. A similar arguments holds for the sum over all slots $i$. Moreover, the products $x_{ki} \cdot x_{\ell j}$ are non-negative and commutative. This yields the IP formulation

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{\ell=1}^{n} p_{k\ell}t_{ij}y_{ijk\ell} \tag{7}$$

subject to

$$\sum_{\ell=1}^{n} x_{\ell j} = 1 \qquad \forall j \in \{1,\dots,n\} \tag{8}$$

$$\sum_{j=1}^{n} x_{\ell j} = 1 \qquad \forall \ell \in \{1,\dots,n\} \tag{9}$$

$$x_{\ell j} \in \{0,1\} \qquad \forall \ell, j \in \{1,\dots,n\} \tag{10}$$

$$\sum_{k=1}^{n} y_{ijk\ell} = x_{\ell j} \qquad \forall i,j,\ell \in \{1,\dots,n\} \tag{11}$$

$$\sum_{i=1}^{n} y_{ijk\ell} = x_{\ell j} \qquad \forall j,k,\ell \in \{1,\dots,n\} \tag{12}$$

$$0 \le y_{ijk\ell} = y_{ji\ell k} \qquad \forall i,j,k,\ell \in \{1,\dots,n\} \tag{13}$$

such that all binary solutions are feasible assignments and their cost is the same as in the quadratic case. It is also called level-1 *Reformulation-Linearization Technique* (RLT1). As the name suggests there are also higher levels of this techniques considered in literature (up to 3), but their size grows with a higher polynomial in $n$. Thus their memory requirements are much more demanding and as of yet impractical for $n > 27$ [12]. Since even the size of RLT1 is quite challenging for optimizing layouts with $n = 34$ characters, we do not yet consider RLT2 and RLT3 in this work.

*Kaufman-Broeckx Linearization*

We also investigate a formulation by Kaufman and Broeckx [14] that is minimal in terms of new variables. That is, instead of the $n^4$ variables $y_{ki\ell j}$ only $n^2$ variables

$$w_{\ell j} = x_{\ell j} \sum_{k=1}^{n}\sum_{i=1}^{n} p_{k\ell}t_{ij}x_{ki}$$

are introduced. Thus the objective function becomes

$$\min \sum_{\ell=1}^{n}\sum_{j=1}^{n} w_{ij}.$$

If $x_{\ell j} = 1$, then the contribution of $w_{\ell j}$ to the objective is a linear function, and it is 0 if $x_{\ell j} = 0$. Hence, it suffices to add the constraints

$$w_{\ell j} \ge 0$$

$$w_{\ell j} \ge \sum_{k=1}^{n}\sum_{i=1}^{n} p_{k\ell}t_{ij}x_{ki} - a_{ij} \cdot (1 - x_{\ell j})$$

with $a_{\ell j} = \sum_{k=1}^{n}\sum_{i=1}^{n} p_{k\ell}t_{ij}$. In fact, any upper bound on $\sum_{k=1}^{n}\sum_{i=1}^{n} p_{k\ell}t_{ij}$ will do for $a_{\ell j}$ such that the right-hand side is at most 0 whenever $x_{\ell j} = 0$ because in any optimum solution $w_{\ell j}$ will be tight at the maximum of both right-hand sides.

This formulation is much smaller in size and thus much easier to solve than RLT1, but the achieved lower bounds are much weaker. It can be strengthened by replacing the non-negativity constraints with the Gilmore-Lawler Bound [16].

**Case: Symmetric Cost Function**

We start by considering the Fitts-bigram energy case given in Eq. 1. It covers virtual keyboards operated by moving a single end-effector (e.g., stylus, finger, mouse) (e.g., [4, 8, 19, 25]). In this case, the instance becomes entirely symmetric by replacing the probabilities with $p'_{k\ell} := (p_{k\ell} + p_{\ell k})/2$. We thereby obtain a symmetric quadratic form as objective function. This symmetry can be exploited to reduce the size of the formulations by a constant factor.

**Case: Non-symmetric Cost Function**

We encounter a non-symmetric cost function in cases where the time cost of returning from slot $\ell$ to $k$ is not equivalent to that from $k$ to $\ell$. An example that we later examine is the ten-finger typing case. Here, typing "F" after "I" receives a different predicted movement time than typing "I" after "F." In these cases, we use the formulations as described above.

## RESULTS

To evaluate the approach, we consider three previously studied cases of keyboard optimization. Our goal is not to "compete" against random search. Rather the studies should be understood as proofs-of-concept for the two main advantages of IP: improved chance of finding the global optimum and the estimation of bounds.

Figure 3 shows the winner layouts and provides an overview of the optimization tasks. The cases differ in 1) keyboard layout, 2) the size of search space, 3) objective function, and 4) the existence of previous optimized layouts. In all cases, we minimize movement time and assume constant button width and no whitespace between buttons. Two of the keyboard types (a, c) are actually in use by billions of users.

For each case, we used one computer with 32 Intel Xeon 2.70Ghz processors (16 physical, 32 virtual cores) and 256GB RAM. Both formulations for a case were solved in

parallel as proposed. Gurobi 5.6 was used with the Interior Point Method to solve the root relaxation. The results reported here were obtained within two days.

#### Case 1: 26 Letter Trapezoid Layout for Touchscreens

This case considers the trapezoid-shaped virtual keyboard common on smartphones with touchscreens. The search space is $4 \times 10^{26}$. A previous case of this layout is given in the multi-objective case of Dunlop and Levine [8].

Our objective function uses the Zhai-Hunter-Smith variant of Fitts' law [25] ($a = 0.0, b = 0.084, c = 0.127$). The bigram distribution updates previously used distributions. We combined two distributions representative of modern text entry with smartphones, weighed 50/50: the mobile email set [23] and the NUS SMS corpus [7]. We assume that the spacebar is fixed to the elongated key on the bottom. The baselines are Qwerty (33.65 wpm) and the Dvorak Simplified Keyboard (DSK) (33.97 wpm). Computation was carried out using the symmetric case described above.

The winner keyboard, presented in Figure 3 was computed within 34 hours. The predicted wpm for the winner keyboard is 43.10, which is within 4% of the global optimum. It is an improvement of 28% over Qwerty.

#### Case 2: The Zhai-Hunter-Smith Hexagonal Layout

Our second case considers the hexagonal layout familiar from the work of Zhai et al. [25]. The search space is $9 \times 10^{30}$. This is a keyboard operated on PDAs by using a stylus.

In our task, 29 characters are considered as shown in Figure 3. For direct comparison of results, we use the same objective function of Zhai et al. It was also used in a follow-up paper by Bailly et al. that found the improved Justhci layout [2]. The alphabetical, Metropolis [25], and Justhci [2] keyboards achieve predicted rates of 32.37, 42.02, and 42.38 wpm, respectively. For IP solution, we use the symmetric case described above.

The winner keyboard, presented in Figure 3 was computed within 15 hours and lies within 7% of the global optimum. Unlike in Justhci and Metropolis, the spacebar is not in the middle. In comparison to Metropolis, not only letters E, A , I, but also O, is close to the spacebar. The predicted wpm is 42.65. It is an improvement of 0.6% over justhci, 1.5% over metropolis, and 32 % over the alphabetical layout.

#### Case 3: 3-Row Physical Keyboard for Ten-Finger Typing

Our third case considers the full 34 character 3-row physical keyboard. The case is motivated as there are aspirations to challenge Qwerty and optimize layouts for different languages [9]. Our task involves the 26 letters and the 8 most frequent characters in our corpus. The search space is $3 \times 10^{38}$.

As the movement model, we use a previously ignored regression model of ten-finger typing [13]. The Hiraga model considers various effects of expert touch typists: finger and hand differences, and effects of moving fingers up/down on the key columns. More precisely, it predicts movement time $t_{ij}$ (msec) from keyslot $i$ to $j$ as

$$t_{ij} = 185.8 - 40h + 18.3r - 11.0f + 0.154R + 1.07F, \quad (14)$$

where $h$, $r$, $f$, are parameters describing hand transition, row transition, and finger transition, respectively, and $R$ and $F$ are weights for row and fingers. See the original paper for values. The caveat of this model is that the parameters were obtained based on a single expert typist. However, its predictions for the difference between Qwerty and DSK matches well with empirical literature [17]. We are not aware of validating evidence for the existing heuristic models (e.g., [9, 11]).

As the frequency distribution, we use a selection of chat room data, mobile email, newspaper English, SMS data, and Tweets, with uniform weighing[2]. Our baselines are the alphabetical, Qwerty, and DSK layouts, which receive 74.49, 73.21, and 76.59 wpm, respectively. For solution, we used the non-symmetric approach described above.

The winner keyboard, presented in Figure 3, was found within 7 minutes. The predicted wpm for the keyboard is 78.36. It is an improvement of 7.0% over Qwerty, and 2.3% over DSK. This is computed to within 0.95% of the global optimum. We consider this a significant achievement given the size of the search space. The layout has a prominent DSK-like feature: the letters on the middle row have a large overlap with those in DSK. Another interesting feature is that spacebar and period are next to each other.

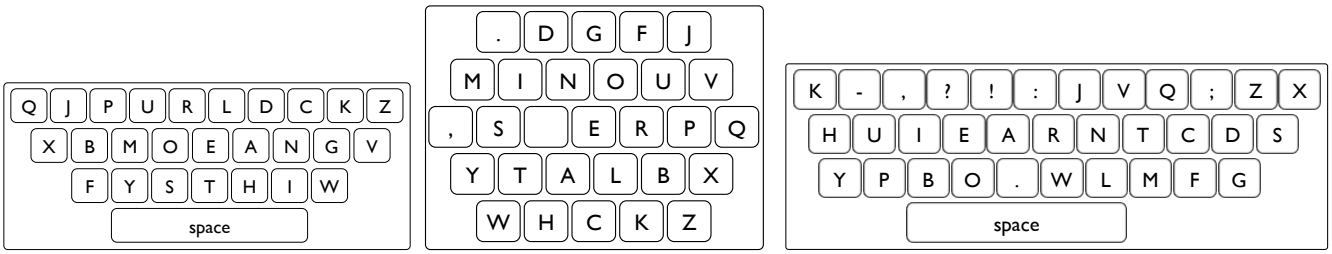
## DISCUSSION

Our long-term goal is a general framework for using IP in UI optimization together with random search methods. The challenge is that although IP is already a general optimization approach, performance is heavily dependent on the modeling of case-specific properties and the choice of methods. The results presented in this paper for keyboards are promising. Our techniques allowed for solving the "classic" 26 letter Fitts-bigram case to a (guaranteed) 4 % range from optimality and approaching optimality to a 1 % in the case of physical keyboards, even if the search space was much larger. To our knowledge, this is the first time that optimality bounds have been presented in user interface optimization in HCI.

In the future, IP should be extended to cover other important factors in keyboard optimization. One of them is error correction: Here the task is to make the local neighborhood of a key as statistically unambigous as possible [8]. Another is predictive text entry: Here the task is to assign letters on a reduced-key grid in order to minimize the number of keypresses per character. The third challenge is to consider how users at various level of expertise should be taken into account. In contrast, existing cost functions consider only expert performance. The fourth is the varying size of keys. Here the challenge is to optimize not only the positions of letters but also their individual size [25]. We presently see no reason why IP could not be extended to cover these problems. However, success will depend analytical work to find efficient models for IP solvers.

We support further research efforts by offering instructions on our project homepage for further cases in keyboard optimization. The page informs how to replicate our results and

[2]All distributions and case data are released on the project hompage.




(a) Virtual trapezoidal: Pred. 43.10 wpm (b) Virtual hexagonal: Pred. 42.65 wpm (c) Physical 3-row standard: Pred 78.36 wpm

Figure 3: The best keyboard designs identified for three keyboard optimization tasks.

extend the models to other problems by using familiar logical terms and clauses.

## CONCLUSION

Because keyboard design problems often include enormous search spaces, it is important to research efficient and appropriate optimization methods. This paper has presented an integer programming (IP) approach and applied it to three cases of keyboard layout optimization. The results are favorable, showing clear improvements achieved over previous baselines. Moreover, IP can inform the designer on how far from the optimum a given design is, or how much a particular decision is going to cost. Our results show that some very large problems can be computed to a 1% margin from optimality. We emphasize that IP is complementary to other approaches. That is, we may run them in parallel and layouts might be injected as new incumbents in the branch-and-bound process. Moreover, standard solvers like CPLEX and Gurobi offer APIs to interact with the search via callbacks. In turn, heuristic approaches benefit from IP because the latter reports a guarantee for the quality of the solution. In the future, we aim to extend this approach to other layout problems in HCI.

## ACKNOWLEDGEMENTS

This research was funded by the Max Planck Center for Visual Computing and Communication and the Cluster of Excellence for Multimodal Computing and Interaction at Saarland University. Code and data are released on the project homepage at http://resources.mpi-inf.mpg.de/keyboardoptimization/.

## REFERENCES

1. Adams, W. P., and Johnson, T. A. Improved linear programming-based lower bounds for the quadratic assignment problem. *DIMACS 16* (1994), 43–75.
2. Bailly, G., et al. Menuoptimizer: Interactive optimization of menu systems. In *Proc. UIST*, ACM (2013), 331–342.
3. Bertsimas, D., and Tsitsiklis, J. *Introduction to Linear Optimization*, 1st ed. Athena Scientific, 1997.
4. Bi, X., Smith, B. A., and Zhai, S. Multilingual touchscreen keyboard design and optimization. *Human–Computer Interaction 27*, 4 (2012), 352–382.
5. Bixby, R., and Rothberg, E. Progress in computational mixed integer programming. *Annals of Operations Research 149*, 1 (2007), 37–41.
6. Burkard, R. E., and Offermann, D. M. J. Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. *Zeitschrift für Operations Research 21*, 4 (1977), B121–B132.
7. Chen, T., and Kan, M.-Y. Creating a live, public short message service corpus: The nus sms corpus. *Language Resources and Evaluation 47*, 2 (2013), 299–335.
8. Dunlop, M., and Levine, J. Multidimensional pareto optimization of touchscreen keyboards for speed, familiarity and improved spell checking. In *Proc. CHI*, ACM (2012), 2669–2678.
9. Eggers, J., et al. Optimization of the keyboard arrangement problem using an ant colony algorithm. *European Journal of Operational Research 148*, 3 (2003), 672–686.
10. Gajos, K., and Weld, D. S. Supple: automatically generating user interfaces. In *Proc. IUI*, ACM (2004), 93–100.
11. Goettl, J. S., Brugh, A. W., and Julstrom, B. A. Call me e-mail: arranging the keyboard with a permutation-coded genetic algorithm. In *Proc. Applied Computing*, ACM (2005), 947–951.
12. Hahn, P. M., et al. A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem. *INFORMS J. on Computing 24*, 2 (Apr. 2012), 202–209.
13. Hiraga, Y., et al. An assignment of key-codes for a japanese character keyboard. In *Proc. Computational Linguistics*, ACL (1980), 249–256.
14. Kaufman, L., and Broeckx, F. An algorithm for the quadratic assignment problem using bender's decomposition. *European Journal of Operational Research 2*, 3 (1978), 207–211.
15. Koopmans, T. C., and Beckmann, M. Assignment problems and the location of economic activities. *Econometrica: Journal of the Econometric Society* (1957), 53–76.
16. Lawler, E. L., and Wood, D. E. Branch-and-bound methods: A survey. *Operations research 14*, 4 (1966), 699–719.
17. Lewis, J., Potosnak, K., and Magyar, R. Keys and keyboards. *Handbook of Human-Computer Interaction* (1997), 1285–1315.
18. Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., and Querido, T. A survey for the quadratic assignment problem (2007). 657–690.
19. MacKenzie, I. S., and Zhang, S. X. The design and evaluation of a high-performance soft keyboard. In *Proc. CHI*, ACM (1999), 25–31.
20. Oulasvirta, A., et al. Improving two-thumb text entry on touchscreen devices. In *Proc. CHI*, ACM (2013), 2765–2774.
21. Pollatschek, M., Gershoni, N., and Radday, Y. Optimization of the typewritter keyboard by computer simulation. *Angewandte Informatik 10* (1976), 438–439.
22. Rao, S. S. *Engineering optimization*. John Wiley & Sons, 2009.
23. Vertanen, K., and Kristensson, P. O. A versatile dataset for text entry evaluations based on genuine mobile emails. In *Proc. MobileHCI'11*, ACM (2011), 295–298.
24. Wolsey, L. A. *Integer Programming*, vol. 42. Wiley New York, 1998.
25. Zhai, S., Hunter, M., and Smith, B. A. Performance optimization of virtual keyboards. *Human–Computer Interaction 17*, 2-3 (2002), 229–269.