Design of an Optimal Hindi Keyboard for Convenient and Efficient Use

Priyendra S. Deshwal
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
Kanpur PIN-208016, India
priyendra.deshwal@iitk.ac.in

Kalyanmoy Deb
Kanpur Genetic Algorithms Laboratory
Department of Mechanical Engineering
Indian Institute of Technology, Kanpur
Kanpur PIN-208016, India
deb@iitk.ac.in

KanGAL Report Number 2003004

Abstract- In this paper, we present a new design of the Hindi[1] keyboard for convenient typing. We describe the ergonomic criterion we have used to evaluate and compare keyboards. This criterion is a mathematical formulation of keyboard optimality in terms of the distribution of the typing effort among the ten fingers, accessibility of commonly used keys and various other factors. Measured against this criterion, our keyboard performs more than twice as better than the standard Hindi keyboard. We also describe a genetic algorithm based optimization framework which we use to arrive at our new keyboard design. Finally, we perform some sensitivity analysis on our optimization procedure and demonstrate that our results confirm to intuitive expectations.

1 Optimization of Keyboards

The layout of the various characters on a keyboard has profound impact on the efficiency of a typist. If frequently occurring characters are not easily accessible, the rate of typing will go down. An ill-designed keyboard might place a disproportionately high load on the weaker fingers of the hand, leading to typing fatigue (even musculoskeletal injuries in the long term). Hence it is important while designing a keyboard that a significant amount to thought be devoted to determine the most optimal arrangement of characters.

For English and other European language keyboards, considerable investigation has gone into finding a suitable arrangement of characters. The standard keyboard is the QWERTY keyboard that everyone is familiar with. Also there is the Dvorak keyboard which was proposed in the 1920's and 30's by August Dvorak and William Dealey. This keyboard was a result of significant ergonomic research and is known to outperform the standard QWERTY keyboard on many factors (home row usage, for example).

Standard keyboard arrangements exist for Hindi too. However, a detailed analysis of whether these are truly optimal or better arrangements exist, has not been done. In this paper, we present such an analysis. The aims of this paper are:

- To analyze the optimality of the current standard Hindi keyboard with respect to various arrangements possible.
- To find (if it exists) a keyboard arrangement that is better than the current Hindi keyboard in terms of typing convenience and efficiency.

[1]Hindi, the third most commonly spoken language in the world after Chinese and English, is used by more than 600 million people in India.

2 A Summary of Related Research

Research has been done previously in the area of optimizing keyboard layouts using optimization techniques like Genetic Algorithms [6, 2], Simulated Annealing [5], Ant Colony Optimization [7] etc. Most of this research has been in two broad directions:

- *Normal Keyboards* - These are traditional keyboards where each character is mapped to a unique key on the keyboard. The objective of optimization in such a case is to arrange the characters on the keys in a manner that is most efficient in terms of typing time and effort. Most of the work done in this regard has been for the Roman script. This work is an effort to extend these same methods to the Hindi language. For further references related to this, refer to [5, 7].
- *Ambiguous Keyboards* - With the emergence of devices like mobile phones, ambiguous keyboards are gaining in popularity and relevance. These are keyboards where multiple characters are mapped to a single key. In such keyboards, either the user presses a key multiple times to enter a character or an ambiguity resolution mechanism predicts which character the user actually intended to type. Some approaches discussed in [4, 3] assume that the system uses a statistical model (derived from commonly used English words) to predict the most likely character that the user wanted to type depending on the characters already entered. Others wait for the user to enter the entire word and then search in a dictionary for all words

which can be formed by the key combinations that the user pressed. In both these approaches, the essential problem is that of ambiguity reduction. So the objective of the optimization in these problems is to determine how to map different sets of characters to the keys so that the resulting ambiguity is minimized.

In this paper, we focus on normal keyboards and henceforth, unless stated otherwise, a keyboard should be taken to mean a normal keyboard.

3 The Devanagari Script

Devanagari is the script in which Hindi is written. The *Bureau of Indian Standards* has standardized the character set for the Devanagari script. The standard is named the Indian Script Code for Information Interchange (ISCII). The original document issued by the Govt. of India giving detailed specification of the standard can be obtained from [1]. The character set is shown in Figure 1. Note that the script possess some conceptual differences from the Roman script. Some of these are summarized below:

- The concept of *matras*(character modifiers) has no parallels in the Roman script. These can occur as stand alone characters or be appended to other characters to modify their sound. There are twelve such *matras* in all.
- Each character has a *shiro rekha* (a horizontal line) on top of it.
- Character modifiers, basically *matras* and other special characters, can occur before, on top, below or after the main character that they modify.

4 Optimization Scheme

Before presenting our methodology, here are a few terms and definitions that shall be used throughout the remainder of this paper.

4.1 Terminology

- C: The complete set of Hindi characters.
- K: The complete set of assignable keys on a keyboard. This set does not contain keys like `RETURN` or `SPACE` as they are already mapped to standard characters.
- M: The set of macros available on a keyboard. A macro is a key combination on the keyboard that produces one character. For most keyboards, every key k is a macro and also [`SHIFT`+k] is a macro.
- Key Mapping: The key mapping of a keyboard is defined as a mapping $f : C \rightarrow M$ which determines which character is mapped to which macro.
- Monographs: Each single keystroke is referred to as a monograph. Hence in the typing of the word “the”, we have three monographs [`t`], [`h`], [`e`]. However, while typing “The” we have four monographs [`SHIFT`], [`t`], [`h`], [`e`]. The relative frequency of occurrence of a monograph m_i with respect to all possible monographs is referred to as f_{m_i}.
- Digraphs: Pairs of keystrokes hit during typing are called digraphs. For example in the typing of “the” we have two digraphs as follows ([`t`], [`h`]) and ([`h`], [`e`]). The relative frequency of occurrence of a digraph d_i with respect to all possible digraphs is referred to as f_{d_i}.

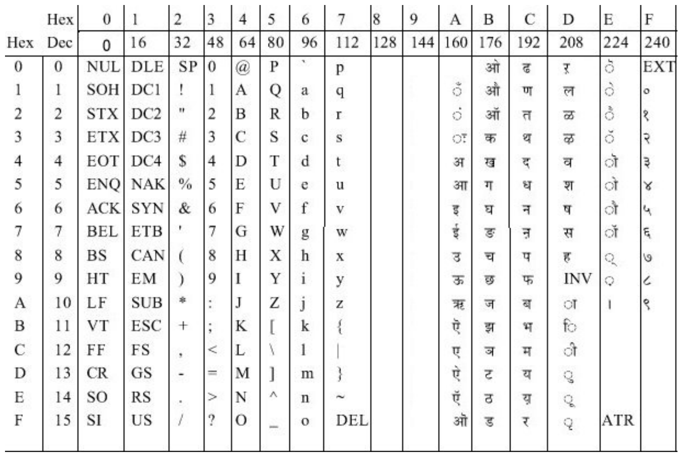
4.2 Keyboard Representation

The design of the keyboards available in the market are hardly suited for investigations regarding optimality. The physical alignment of the keys is irregular; plus there are some keys which cannot be moved (the space bar for example). Also, there are aesthetic constraints like all numbers on the keyboard should be arranged on adjacent keys. To expect the optimization procedure to automatically take these into consideration is unreasonable. So we use a keyboard abstraction proposed by [7] for our work. See Figure 2 to get an idea of the relationship between an actual keyboard and the abstract keyboard that we have used.

Once we use this abstraction, each key on the keyboard can be mapped to a unique triplet as $(hand, row, column)$. Here $hand$ is the hand that is used to hit the key while the row and column denote the position on the abstract keyboard that the key appears. Similarly, each macro can be mapped to a 4-tuple $(modifier, hand, row, col)$. Here the first component $modifier$ tells whether the macro is for the shift modified key or the normal key. Now to represent a key mapping f, we use a four dimensional array `keymap` as follows:

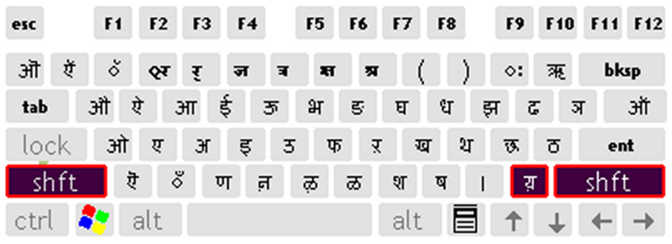
For any macro $m = (modifier, hand, row, column)$, and a character c such that $f(c) = m$, we define, `keymap[modifier][hand][row][column]`$=c$. Here the ranges of the array indices are $\{normal, shift\}$ for $modifier$, $\{left, right\}$ for $hand$, $\{0, \ldots, 3\}$ for row and $\{0, \ldots, 7\}$ for $column$. Note that the number of options in each of the four variables is an integer which can be represented as 2^n.

Therefore, for any possible keyboard, this representation allows us to store its key mapping as a 4-D array. Conversely, any such 4-D array will always correspond to some valid key mapping. Hence using the abstraction discussed, we have been able to establish a one-to-one correspondence between keyboards and 4-D arrays. Now for optimization it will suffice to search in the space of these 4-D arrays.

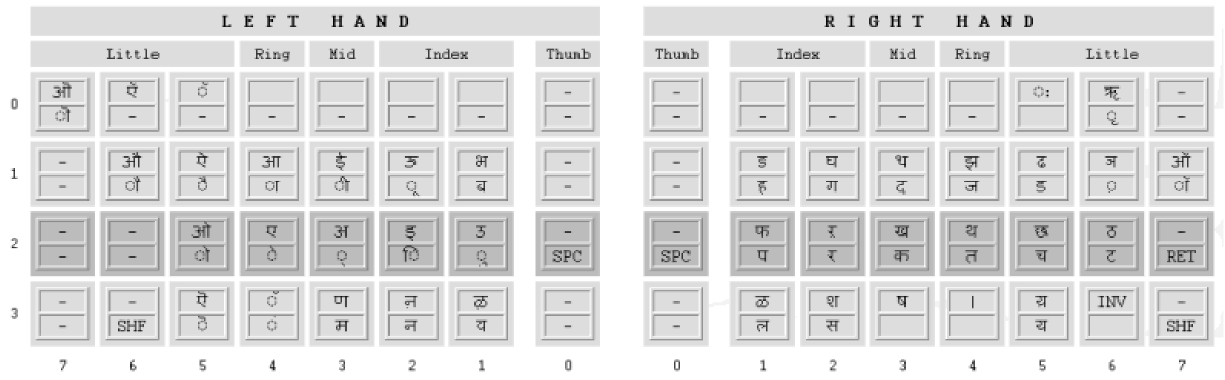


Hex	Dec \ Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
0	0	NUL	DLE	SP	0	@	P	`	p				ओ	ढ	ऱ	◌ॆ	EXT
1	1	SOH	DC1	!	1	A	Q	a	q			◌ँ	औ	ण	ल	◌े	०
2	2	STX	DC2	"	2	B	R	b	r			◌ं	ऑ	त	ळ	◌ै	१
3	3	ETX	DC3	#	3	C	S	c	s			◌ः	क	थ	ऴ	◌ॅ	२
4	4	EOT	DC4	$	4	D	T	d	t			अ	ख	द	व	◌ॊ	३
5	5	ENQ	NAK	%	5	E	U	e	u			आ	ग	ध	श	◌ो	४
6	6	ACK	SYN	&	6	F	V	f	v			इ	घ	न	ष	◌ौ	५
7	7	BEL	ETB	'	7	G	W	g	w			ई	ङ	ऩ	स	◌ॉ	६
8	8	BS	CAN	(	8	H	X	h	x			उ	च	प	ह	◌्	७
9	9	HT	EM	)	9	I	Y	i	y			ऊ	छ	फ	INV	◌़	८
A	10	LF	SUB	*	:	J	Z	j	z			ऋ	ज	ब	◌ा	।	९
B	11	VT	ESC	+	;	K	[	k	{			ऎ	झ	भ	◌ि		
C	12	FF	FS	,	<	L	\	l	|			ए	ञ	म	◌ी		
D	13	CR	GS	-	=	M	]	m	}			ऐ	ट	य	◌ु		
E	14	SO	RS	.	>	N	^	n	~			ऍ	ठ	य़	◌ू		
F	15	SI	US	/	?	O	_	o	DEL			ऒ	ड	र	◌ृ	ATR	

Figure 1: The ISCII character set for Indic scripts. The Hindi characters start from 161 onwards. Note that before 161, this character map is identical to the ASCII character map (taken from [1])



(a)



(b)

Figure 2: (a) The standard Hindi keyboard, (b) The abstraction of the original keyboard. Each key shows the normal character in the bottom cell and the shift-modified character in the upper cell. Note that some key positions are marked with a hyphen to indicate that these cannot be occupied by any character. This is done for standard keys like RETURN or number keys whose positions are fairly fixed on the keyboard. Also note that in order to compensate for the differences in the length of the fingers and the thumb, the space key which is supposed to be hit by the thumb is brought to a separate column between the two halves of the keyboard. However other keys on these columns are set to be unassignable.

Table 1: Ideal load distribution values used for the analysis. The table lists coefficients for each row and column. To get the value of $f_{m_i}^{desired}$ for any key, multiply the coefficients for the row and column taken from this table and multiply by 0.5 so that the load should be shared equally between each hand.

Ideal Load Distribution		
S. No	**Row**	**Column**
1	17.39%	15.70%
2	19.57%	10.58%
3	47.83%	15.70%
4	15.21%	23.40%
5		18.27%
6		6.73%
7		5.45%
8		4.17%

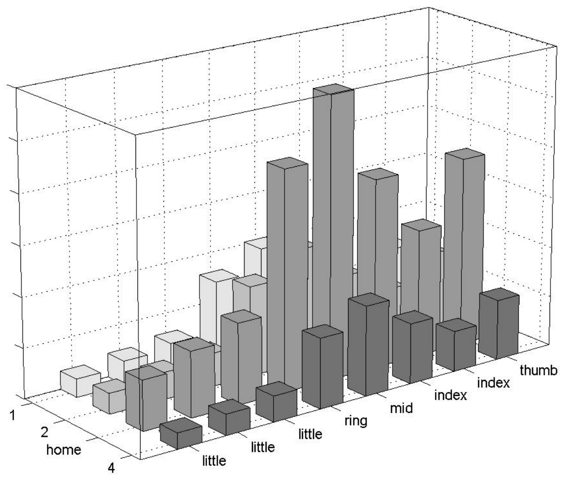


Figure 3: A 3-D graph for the $f_{m_i}^{desired}$ values for the left half of the keyboard.

4.3 Optimality of Keyboards

At a very macro level, an optimal keyboard for any script should have the following qualities:

- Allow for minimum typing effort,
- Maximize typing speed,
- Reduce typing errors,
- Allow easy learning of the touch typing method.

Some ergonomic studies have been done regarding keyboards. See [8] for an example. A concrete definition of the optimality of a keyboard is provided by [7]. This definition is motivated by the qualities listed above. Their definition is discussed here and used as a standard metric in this work.

According to the method proposed, each keyboard is evaluated on six criteria and the final score of the keyboard is taken as a weighted sum of these six individual scores. These six criteria are:

4.3.1 Load Distribution

Each finger of the hand has a certain strength. Note that while typing, the total load on the fingers is constant. However, it would be highly desirable if this total load can be distributed among the fingers in proportion of their relative strengths. Therefore, the index finger which is the strongest should share most of the total load. Along similar lines, keys in the middle row are the most easily accessible and therefore these should contain the most frequently occurring characters. Hence, keys near the center of the keyboard (which are hit by the index finger) and in the middle row should share the maximum fraction of the total load. Formally stated, we can assign an ideal load distribution between all the monographs and the performance of any keyboard can be measured by calculating the deviation that the actual load distribution for this keyboard has with the ideal distribution. Hence, let f_{m_i} be the observed relative frequency of a monograph m_i having ideal relative frequency $f_{m_i}^{desired}$ and let Ξ_i^m be the set of all monographs. So on this index the performance of the keyboard is measured as:

$$v_1 = \sum_{m_i \epsilon \Xi_1^m} (f_{m_i} - f_{m_i}^{desired})^2$$

The values used for $f_{m_i}^{desired}$ are given in Table 1[2]. These values for the left half of the keyboard are also shown graphically in Figure 3. To interpret the values given in Table 1, consider a typical monograph $m = (hand, row, column)$, then the value for $f_m^{desired}$ can be obtained by multiplying the respective fractions given in Table 1 for row and $column$ and then multiplying by a factor of 0.5 so that the total load is divided equally between the left and right hands.

4.3.2 Modifier Overhead

Every time a modifier such as the `SHIFT` key has to be pressed, it adds to the inefficiency of the keyboard. Hence, the most commonly used characters should be assigned to normal macros while the lesser used characters must be assigned to shift modified macros. The performance of the keyboard on this index is measured by a factor v_2 and is calculated by dividing the total number of keys pressed by the number of characters that were typed. Hence if the text to be typed was “The”, the total number of characters is 3

[2]The keyboard in [7] used nine columns. Since we have used eight columns, we redistribute the contribution of the ninth column to other eight columns.

while the number of keys pressed is 4. Hence the modifier overhead will be 1.33.

4.3.3 Hand Alternation

Fast and comfortable typing is assured if keys that are to be struck consecutively are on opposite sides of the keyboard. The reason is that this allows one hand to move to the next key's position while the other is in the process of hitting the current key. In order to quantify this hand alternation indicator, we add up the frequency of the digraphs which are typed by using one hand only. The hand alternation index is calculated as:

$$v_3 = \sum_{d_i \epsilon \Xi_3^d} f_{d_i}$$

where Ξ_3^d is the set of all digraphs which are typed using one hand only.

4.3.4 Consecutive Usage of Same Finger

The same concept of hand alternation can be extended to fingers as well. If consecutive keys are hit by the same finger, it might lead to inefficiency in typing. This index is calculated by summing up the frequencies of all diagraphs which require both keys to be hit by the same finger. However, instead of simply summing up the frequencies of these diagraphs, these frequencies are first weighted by a distance coefficient. The greater the distance between the two keys of a diagraph, the more penalizing a consecutive usage. The relevant set Ξ_4^d is therefore the set of digraphs which are typed using the same finger. Therefore,

$$v_4 = \sum_{d_i \epsilon \Xi_4^d} f_{d_i} dist(d_i)$$

The distance coefficient is given by the Manhattan distance function:

$$dist(d_i) = |c_2 - c_1| + |r_2 - r_1|$$

where c_2 and c_1 are the respective columns of the two keys of which the diagraph is made and r_2 and r_1 the corresponding rows.

4.3.5 Big Steps by Fingers of the Same Hand

When the same hand is used for two consecutive hits, large distances which require awkward hand posture lead to slow and laborious typing. This happens for diagraphs whose component keys have a vertical distance greater or equal to one. The relevant set Ξ_5^d is therefore the set of digraphs which are typed using the same hand, but not the same finger and the vertical distance between the two keys is greater than or equal to one row. A weight coefficient depending on the two fingers used is assigned to each digraph. The index is calculated as:

$$v_5 = \sum_{d_i \epsilon \Xi_5^d} \kappa(d_i) f_{d_i}$$

The values for the coefficients $\kappa(i, j)$ were taken from [7] and are presented in Table 2.

Table 2: The values for κ used for computing v_5(Taken from [7])

	Thumb	Index	Mid	Ring	Little
Thumb	0	0	0	0	0
Index	0	0	5	8	6
Mid	0	5	0	9	7
Ring	0	8	9	0	10
Little	0	6	7	10	0

4.3.6 Hit Direction

Ergonomic research has revealed that for digraphs typed using one hand only, the preferred hit direction is from the little finger towards the thumb. This is the finger movement that most people find natural. Ξ_6^d is therefore the set of all digraphs which are produced by using one hand only and whose hit direction is not the preferred one. This index is calculated as:

$$v_6 = \sum_{d_i \epsilon \Xi_6^d} f_{d_i}$$

4.3.7 Overall Grade

A natural way to define a global grade from the six indices $v_j (1 \leq j \leq 6)$ is to take a weighted sum. Each indicator can be assigned weights based on their relative importance and a single objective optimization can then be performed on the resulting overall score. However, these indices have different ranges and units and therefore cannot just be simply added. They are first divided by the corresponding indices of a reference keyboard[3] $v_{j,ref} (1 \leq j \leq 6)$. Once the indicators are turned dimensionless they can be multiplied by a relative weight coefficient $\gamma(j)$ and added. The values of weights $\gamma(j)$ are represented in Table 3.

Using these values, the final score of a keyboard can be evaluated as:

$$V = \sum_{j=1}^{6} \gamma(j) \frac{v_j}{v_{j,ref}}$$

[3]In actual practice, the reference keyboard used was the standard Hindi keyboard.

Table 3: The values for $\gamma(j)$ used for computing the overall grade for a keyboard from $v_j(1 \leq j \leq 6)$ (Taken from [7])

Relative Weights of the Various Factors	
Index	**Relative Weight**
Load Distribution	0.45
Modifier Overhead	0.5
Hand Alternation	1.0
Consecutive Usage of same finger	0.8
Big Steps	0.7
Hit Direction	0.6

4.4 Optimization Details

With the keyboard abstraction presented in Section 4.2, optimization is essentially reduced to search in the space of 4-D arrays. A Steady-State Genetic Algorithm may be used for this search with many variations for parameters like mutation probability, population size etc. A typical combination of the parameters is provided below:

- Population Size - 500
- Number of generations - 2000
- Probability of Mutation - 0.1
- Probability of Crossover - 0.9
- Selection Operator - Tournament Selection
- Steady-State GA Overlap - 50% of the population is inherited from the parent pool

4.5 Genetic Operators

We use the following selection, mutation and crossover operators during the search.

4.5.1 Selection

Both the Roulette Wheel Selection and Tournament Selection can be used. In actual usage, performance with both these selection operators was found to be comparable.

4.5.2 Mutation

Given our representation of the keyboard as a 4-D array, mutation can be very intuitively defined as simply swapping the values at two points in the array. On the keyboard, it would translate to swapping the macros mapped for two of the characters. Suppose the parent key mapping is f and the characters randomly selected for swapping are c_1 and c_2. Then the mutated mapping f' is given as follows:

$$f'(c) = \begin{cases} f(c) & , \forall c \neq c_1, c_2 \\ f(c_2) & , c = c_1 \\ f(c_1) & , c = c_2 \end{cases}$$

4.5.3 Crossover

Intuitively the crossover operator that we use, takes a part of the keyboard from one of the parents and copies it verbatim into the child. With this some of the characters are assigned a macro in the child. For the remaining characters, the other parent is consulted for a macro and the nearest available free macro is then assigned to the character in the child. Formally stated, given two parent key mappings (f_{p_1}, f_{p_2}) and a random partition of C (the set of Hindi characters) $C = C_1 \bigcup C_2$, a child mapping f_c is defined as:

$$f_c(c) = \begin{cases} f_{p_1}(c) & , \forall c \in C_1 \\ nearest(f_{p_2}(c)) & , \forall c \in C_2 \end{cases}$$

Here $nearest(m)$ for a macro m is defined as a macro m' that is unassigned to any character and is as close to m as possible on the actual keyboard. Note that this $nearest$ function is required for this case because it is quite possible that the macro $f_{p_2}(c)$ may already be assigned to some other character while copying part of the mapping from f_{p_1}.

4.6 Subsequent Local Search

To ensure that there was no local minima in the vicinity of the optima found by the genetic algorithm, a simple local search procedure is implemented. Two kinds of local search are used:

4.6.1 Repeated Mutations

Starting from the solutions obtained, the solutions are repeatedly mutated. This is equivalent to a local search because the mutation operator that we use ensures that a single mutation will always produce a keyboard that will be exactly similar to the original keyboard except at the points which were swapped.

4.6.2 Swapping Shift-Modified and Actual Characters

After the above procedure, a different mode of local search is conducted. For this we simply swap the normal and shift-modified characters for each key one by one and evaluate the resulting keyboard. If at any point the resulting keyboard comes out to be better, we restart this procedure from the first key. This is done till an iteration comes about without any swapping taking place. At this point, the keyboard produced is reported to be an optimal solution.

5 Implementation Details

- GA Framework: The entire code is written in C++ using the GALib library developed at MIT. This is an excellent open source implementation of many common Genetic Algorithms and is available for free

download from:
`http://lancet.mit.edu/galib-2.4/`.

- Test Data: The evaluation function that we use assumes that we know the relative frequencies of each monograph and diagraph. Since these frequencies cannot be estimated for general Hindi text, we simulate the typing of a large body of Hindi text (15 MB of text in ISCII format) taken from various sources on the Internet and compute the monograph and diagraph frequencies for the same.

6 Results

We present our results for a number of scenarios. These include experiments with selection operators like Roulette Wheel selection, Tournament selection etc. Also we vary parameters like population size and all the various probability parameters to get an idea of how the solutions behave under different parameter values. The results we obtained are summarized below:

- See Figure 4 for the best keyboard design that we were able to evolve.
- See Table 4 for a summary of the results of various experiments with parameter values.
- Our keyboard has a much better score (1.848) than the current standard keyboard (4.05). Note that on our criteria of comparing keyboards, a better keyboard has a smaller score value. Therefore, our keyboard outperforms the standard Hindi keyboard by more than a factor of two.
- In the course of our experiments with various parameters, we observed that the final solutions obtained always had one peculiar property. In all of them a sizeable majority of the vowels were always clustered on one particular side of the keyboard. Our explanation for this hinges on the fact that in normal words sounds and vowel sounds generally alternate. This is to say that all elementary consonant sounds are followed by a vowel sound. Now given the high number of vowels in the Hindi language and the high weight given to the hand alternation index, it follows naturally that all the vowels should arranged on one side of the keyboard while the commonly used consonants like *ka*, *na* etc should occupy the other side.

7 Sensitivity Analysis

Throughout this work we have used a number of parameters. Although we have tried our best to use standard values, the values of these parameters can still be debated upon. In order to convince the reader that our optimization method does not depend critically on the particular parameter values we have chosen, we present sensitivity analysis showing how our solutions vary when we change the weights that we have assigned to the different indices on which we evaluate our keyboards.

Table 4: A summary of the results obtained as part of our experiments. Note that with higher population sizes, the generations taken for convergence decreases considerably even though the ultimate scores obtained are more or less comparable for all population sizes

Roulette-Wheel Selection	
Scenario (# genomes, # generations)	Keyboard score
(50, 2000)	2.008
(250, 390)	2.007
(250, 2000)	1.937
(500, 256)	2.008
(500, 284)	1.942
(500, 2000)	1.848
Tournament Selection	
Scenario (# genomes, # generations)	Keyboard score
(50, 2000)	2.090
(250, 390)	1.972
(250, 2000)	1.903
(500, 256)	2.060
(500, 284)	2.050
(500, 2000)	1.974

7.1 Introduce Right-Left Asymmetry

In our earlier analysis, left and right hands were both given weights of 0.5 each. Our first intuition was to change this to see how the solutions change. The expectation was that all the commonly used characters should move over to the side which was given higher preference. So we used weight values (0.7, 0.3) with higher preference being given to the right hand. When the optimizations were carried out using these, the solutions did actually reflect what we expected. In order to evaluate whether the right side was indeed being given more preference, we compared the key-hits on corresponding keys on both sides of the keyboard. Taking the home row as an example, see Table 5 for a comparison of the key-hits on different sides of the keyboard.

7.2 Large Weight to Modifier Overhead

In this case, we gave modifier overhead an unnaturally large value. Our expectation was that by doing this we should be able to ensure the following two things:

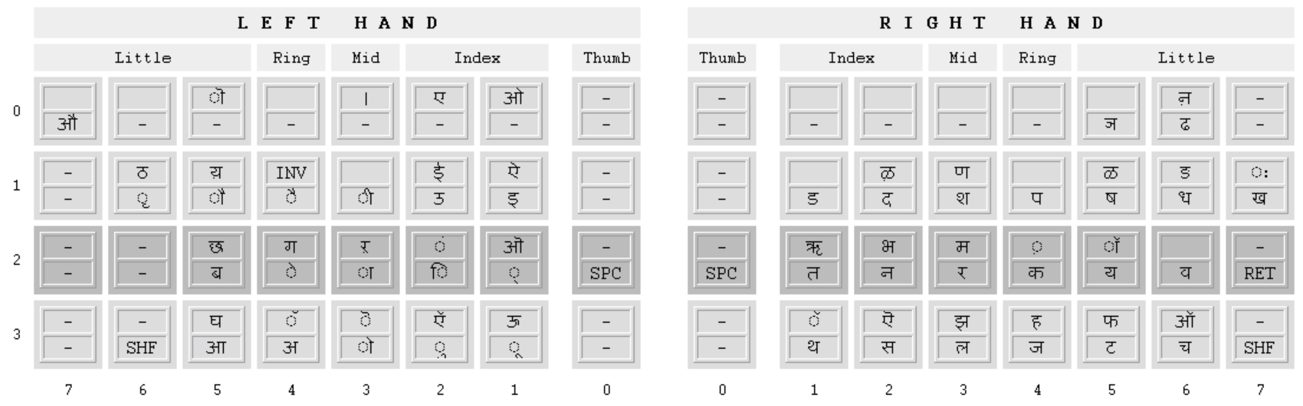


Figure 4: The best keyboard that we were able to evolve. Parameters used were Roulette-Wheel selection, 500 population size and 2000 generations. The score of this keyboard on our evaluation index was 1.848 whereas the standard Hindi keyboard evaluates to 4.05. Keeping in mind that smaller scores represent better keyboards, this represents more than two-fold improvement over the standard keyboard.

Table 5: The number of keystrokes on home-row keys in different columns. Note that with higher preference being given to the right side, all the right hand values are significantly greater than the corresponding left hand values.

	Left side	**Right side**
Key at column 1	476,481	762,243
Key at column 2	491,876	705,620
Key at column 3	830,650	1,092,755
Key at column 4	511,078	724,854
Key at column 5	264,944	455,045

- All the commonly used characters should be assigned to normal and not shift modified keys. Moreover, the final keyboard should have all normal macros assigned to some character i.e. all the unoccupied slots on the keyboard should be from the set of shift modified macros.
- Number of shift-keystrokes for typing any text should be reduced.

So we set all other weights to 0.3 and modifier overhead was set at 2.5. Upon running the optimizing procedure with these weights, both the expectations were achieved. The final keyboard did indeed have no empty slots in the place of normal keys. In terms of the number of times the `SHIFT` keys were pressed while typing our simulation text, for normal keyboards this number was 2,474,006. For the keyboards evolved after making these changes to the weights, this number fell to 1,175,413 indicating that the shift key was used less than half as frequently.

7.3 Large Weight to Hand Alternation

In this trial, we gave a large value to the hand alternation index. The weights were 0.3 for all the other indices and 2.5 for hand alternation. The expected result was that the characters should distribute themselves in such a way that the most commonly used digraphs consist of keys from opposite sides of the keyboard. In order to see the effect of these changes, we ran the optimization procedure using two sets of weights: one is already given above, and for the second we gave equal weights to all the indices. We determined the 100 most common digraphs for both these keyboards. These are the key-pairs that are most commonly hit. Among these for the case with equal weights, we got 12 digraphs among the top 100 which were on the same side of the keyboard. For the other case however, this number was just 4. Hence, out of a total of 100 most common digraphs, only 4 were such that they were typed by the same hand. This shows that by giving a large value to the hand alternation, we were indeed able to ensure that very rarely does the user need to use the same hand consecutively.

8 Conclusions and Future Research

In this paper, we have presented a keyboard design for a new Hindi keyboard that is better than the current standard Hindi keyboard in terms of typing convenience. We make such a confident claim because we have evaluated our design on a formal mathematical evaluation criterion as explained above and compared the results with that of the standard Hindi keyboard. Finally, we also performed a sensitivity analysis to provide convincing proof that our methods do satisfy intuitive expectations. This work however, is far from complete and we can at once identify a few directions in which research could still be conducted.

- Right now, our method focusses on optimizing a weighted sum of the six indices discussed in this paper. Alternatively, one could do a multi-objective optimization in which all six indices (or a carefully chosen subset of these) could serve as the objectives.
- The standard Hindi keyboard has pairs of similar sounding characters mapped to the same key. The consonants *ka* and *kha* for example. Another example would be *ga* and *gha*. With such an arrangement, it becomes very easy for a typist to remember the positions of the keys on the keyboard. However, such a constraint is not modelled in our system. In general, nowhere have we focussed on the *learnability* of a keyboard. Incorporating that into our keyboard model and our evaluation function would be one of the primary directions for extending this research.
- Currently we are using a very generic representation scheme and also very generic mutation and crossover operators. It is expected that use of specialized representations and operators will definitely improve the results that we have obtained so far.
- This work basically focusses on capturing the concept of *optimality* of a keyboard in mathematical terms and tries to find a keyboard which is optimal in terms of this mathematical criterion. However it must be obvious, that no amount of mathematical jugglery can actually replace real human testing of keyboards. Since the keyboards are to be used finally by humans, only they can truly evaluate and decide which one is better. Actual testing by humans should be the next logical step for this research.

Bibliography

[1] Bureau of Indian Standards, Manak Bhavan, 9 Bahadur Shah Zafar Marg, New Delhi PIN 110002, India. *Indian Script Code for Information Interchange - ISCII*, 1991.

[2] David E. Glover. Solving a complex keyboard configuration problem through generalized adaptive search. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 12–31, 1987.

[3] Gregory W. Lesher and Bryan J. Moulton. A method for optimizing single-finger keyboards. In *Proceedings of the RESNA 2000 Annual Conference*, pages 91–93, 2000.

[4] Gregory W. Lesher, Bryan J. Moulton, and D. Jeffrey Higginbotham. Optimal character arrangements for ambiguous keyboards. *IEEE Transactions on Rehabilitation Engineering*, 6(4):415–23, 1998.

[5] Lissa W. Light and Peter G. Anderson. Designing better keyboards. *AI Expert*, page 20, September 1993.

[6] B. J. Oommen, J. S. Valveti, and J. R. Zgierski. Application of genetic algorithms to the keyboard optimization problem. Technical report, Carleton University, Ottawa, Canada, 1989.

[7] Marc Oliver Wagner, Bernard Yannou, Steffen Kehl, Dominique Feillet, and Jan Eggers. Ergonomic modelling and optimization of the keyboard arrangement with an ant colony optimization algorithm. Technical report, Laboratoire Gnie Industriel, cole Centrale Paris, France, 2001.

[8] Shumin Zhai, Alison Sue, and Johnny Accot. Movement model, hits distribution and learning in virtual keyboarding. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24. ACM Press, 2002.