

# Nepali Unicode Keyboard Layout Standardization based on Genetic Algorithm

Chetan Prajapati  
Jwalanta Deep Shrestha  
Shishir Jha

*<http://www.jwalanta.com.np/nepalikeyboard>*

May, 2008

# Acknowledgment

First of all, our sincere thanks go to the One Laptop Per Child (OLPC) Nepal for helping us initiate this project and providing us resources to conduct the research.

We are specially indebted to Shankar Pokharel and Ankur Sharma of OLPC Nepal for their interest in this research and their invaluable suggestion and ideas. Similarly we are also thankful to Bibek Poudel for providing us help and support on some of the technical details associated with the research.

**Chetan Prajapati**  
**Jwalanta Deep Shrestha**  
**Shishir Jha**

**May, 2008**

# Abstract

Recent advancement in Information and Communication Technologies (ICT) have offered a huge prospect for computing in our own language. However, there have been several challenges that have slowed down the moment. One of the biggest hinderance is the absence of standardized Nepali Keyboard Layout. The existing keyboard layouts have been highly intuitive in design rather than being scientific and statistically optimized. These layouts don't consider the basic factors like distribution of frequency load among the keys, hand alternation, and many other factors due to which they put excessive and disproportionate stress on the fingers, which on long term can cause several adverse effects. This report deals with an attempt to create an optimum Nepali keyboard layout which aims at creating a scientifically proportionate and statistically induced character mapping. While several methods of creating and optimizing a keyboard layout may exist, we have used Genetic Algorithm to find an optimized Nepali Unicode Keyboard Layout.

## **Keywords**

Nepali, Keyboard Layout, Devanagari, Unicode, Genetic Algorithm.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Unicode and Nepali</b>	<b>2</b>
2.1	How Does it Help Nepali Computing . . . . .	2
<b>3</b>	<b>Existing Layouts</b>	<b>3</b>
3.1	Traditional Layout . . . . .	3
3.2	Romanized Layout . . . . .	3
3.3	Devanagari Keyboard (Dp) . . . . .	4
<b>4</b>	<b>Corpus</b>	<b>5</b>
4.1	Analysis . . . . .	5
<b>5</b>	<b>Optimization Strategy</b>	<b>6</b>
5.1	Keyboard Representation . . . . .	6
<b>6</b>	<b>Genetic Algorithm</b>	<b>7</b>
6.1	Optimization Characteristics . . . . .	8
6.2	Overall Grade . . . . .	11
6.3	Optimization Details . . . . .	11
6.4	Learnability . . . . .	12
6.5	Genetic Operators . . . . .	12
<b>7</b>	<b>Results</b>	<b>14</b>
<b>A</b>	<b>Appendix</b>	<b>16</b>
A.1	Nepali Unicode Keyboard optimized using Genetic Algorithm . .	16
A.2	Website . . . . .	16
A.3	Overall Ideal Load Distribution . . . . .	17
A.4	Monograph Frequencies . . . . .	18
A.5	Nepali corpus analysis code . . . . .	18
A.6	Traditional Layout . . . . .	20
A.7	Romanized Layout . . . . .	20
A.8	Devanagari Keyboard (Dp) . . . . .	21

# List of Tables

6.1	Ideal Load Distribution . . . . .	9
6.2	The values of $\Phi$ . . . . .	11
6.3	The values for $\Psi(j)$ used for the computing the overall grade for a keyboard from $v_j(1 \leq j \leq 6)$ . . . . .	12

# Chapter 1

## Introduction

Recent advancement in Information and Communication Technologies (ICT) have offered a huge prospect for computing in our country. It has helped us grow opportunities and meet our development goals. However, there have been several challenges that has slowed down the initial moment that we had gained. Among one of the biggest hinderance is the absence of standardization of Nepali Keyboard layout. Due to this, different varieties of keyboard layouts are being used for computing in Nepali. Though these layouts are being used by the users, the fact that the layout and fonts are not standardized is causing a wide spread problem in data interportability and other important applications of Nepali computing.

In addition to the upper mentioned issues, present keyboard layouts have been designed without much research on how they impact the typing behaviour of the users. These layouts are irregular in terms of statistical distribution of the keys due to which the keyboard layout puts excessive and disproportionate stress on the fingers which on long term can cause adverse effects.

Though widely used forms of keyboaed layout exist, detailed analysis of the whether these are truly optimal layouts or not has not been done. The drawbacks of existing Nepali Keyboard Layouts leads to the requirement of the detailed scientific and statistical study of the Nepali corpus in designing the new standard and widely acceptable layout. Furthermore the design should be consistent with the capability of human fingers and fluency with the Nepali language structure. This layout is an attempt to create an optimum Nepali keyboard layout which aims at creating a scientifically proportionate and statistically induced character mapping. The main goal of this particular layout are:

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

## Chapter 2

# Unicode and Nepali

Fundamentally, computers just deal with numbers. They store letters and other characters by assigning a number for each one. Before Unicode was invented, there were hundreds of different encoding systems for assigning these numbers. No single encoding could contain enough characters: for example, the European Union alone requires several different encodings to cover all its languages. Even for a single language like English no single encoding was adequate for all the letters, punctuation, and technical symbols in common use. Any given computer (especially servers) needs to support many different encodings; yet whenever data is passed between different encodings or platforms, that data always runs the risk of corruption.

Unicode provides a unique number for every character used in the computer, no matter what the platform, no matter what the program, no matter what the language. This completely minimizes the conflicts and data corruption caused by the incompatible coding system.

Unicode enables a single software product or a single website to be targeted across multiple platforms, languages and countries without re-engineering. It allows data to be transported through many different systems without risk of data being corrupted.

### 2.1 How Does it Help Nepali Computing

Traditionally Nepali fonts such as Himali, Preeti, Kantipur, Sama etc. has been developed and used to fulfill the need of documents required in Nepali language. All these fonts, though use the Devnagari font as their base, have different coding system. This brings about a lot of complication in downloading the documents from one pc to other, especially when the document prepared in a particular font doesn't get downloaded in the other computer in the absence of the same font in the latter one.

To view the document, the exact font had to be installed in the receiving computer. Sometimes even a single page of document may contain several kinds of fonts which make the downloading process even more complicated and time consuming. But with the Unicode Devnagari font installed in the keyboard, this problem is completely minimized.

## Chapter 3

# Existing Layouts

Currently, three major keyboard layouts exist for typing in Nepali Unicode.

1. Traditional Layout
2. Romanized Layout
3. Devanagari Keyboard (Dp)

### 3.1 Traditional Layout

The Traditional Keyboard Layout has been derived from the layout used in Nepali (mechanical) typewriters (see Appendix A.6). The unicode version of this was developed by Madan Puraskar Pustakalaya (MPP). Although it directly corresponds with the traditional layout and might be easier for anybody already familiar with the traditional layout, it has the following pitfalls:

- It is more of an intuitive design rather than scientific and statistically correct,
- Does not consider the aspects of learnability,
- It is not unicode compatible making it difficult to port to other platforms,
- It is designed considering key-sticking problem of type-writers which has become obsolete.

### 3.2 Romanized Layout

Romanized Layout has been developed by Madan Puraskar Pustakalaya (MPP) and has been designed to ease Nepali typing for those who are already familiar with English keyboard layout (see Appendix A.7). However it has following pitfalls:

- Though a very good keyboard layout for english users, due to the absence of proper arrangement of alphabets, it is not statistically optimized,

- The keys are placed haphazardly, i.e., the alphabets are not arranged according to the load distribution attribute of keyboard design,
- Though it is unicode based and easy to use, even in long term, it does not increase the typing speed,
- User needs to know how to type in english and should have knowledge of the language before he can start relating to the Nepali layout,
- Learnability is not considered (except for english users).

### 3.3 Devanagari Keyboard (Dp)

Devanagari Keyboard (Dp) Layout has been developed by Dhak Prasad Upreti (Bishnu). Although not popular in mainstream use, this Keyboard Layout has been statistically modeled and corresponds to the frequency of letters in Nepali (see Appendix A.8). However, this layout has following pitfalls:

- The corpus analysis deals with only single letter frequency (monograph) and no study of pair of letters (digraph) is done.
- The letters are arranged solely using the frequency of individual letters and no optimization parameter is maintained.
- The layout is not optimized for typing characteristics like modifier overhead (shift key), hand alteration, finger alteration and hit direction.
- The layout is designed manually and no machine assisted optimization is done.

## Chapter 4

# Corpus

The corpus was created using gathering texts from leading Nepali e-newspaper and wiki web-sites like eKantipur, Hamro Samachar and Nepali Wikipedia. Altogether 7,797 articles were collected for this purpose which consisted of 2,452,937 words.

### 4.1 Analysis

The frequency analysis has been done in both letter and syllable level (i.e., monograph and digraph). The result are too big to be included with this report. It can be found at website mentioned at Appendix A.2. The Python scripts that were used to calculate the frequencies are in Appendix A.5.

The following characters were selected for the layout which represent the Nepali written language, as a subset of Devanagari Script.

Consonants	क ख ग घ ङ च छ ज झ ञ ट ठ ड ढ ण त थ द ध न प फ ब भ म य र ल व श ष स ह
Vowels	अ आ इ ई उ ऊ ऋ ए ऐ ओ औ
Joiners	ा ि िी ु ू ृ े ै ो ौ ् ॅ ं ः

## Chapter 5

# Optimization Strategy

Keyboard layout optimization is a discrete combinatorial problem which is constrained by various factors such as statistically oriented optimization, learnability. The statistical optimization problem can be solved with many optimum results using the Genetic Algorithms. They are less susceptible to getting stuck at local optima than any other methods of optimizing keyboard layouts.

Solving such kind of problems with the GA is an art of representing a solutions. There may be many optimum layout satisfying the given layout. To use a genetic algorithm, the first and the most important part is to represent a solution to the problem as a genome (or chromosome). The genetic algorithm then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s). Since the algorithms selects only the best out of the population for the reproduction, the process ultimately lead to the best possible solutions. It may not be the best or the optimum but it by nature leads to the solutions near to the best ones. Further more it may not necessarily lead to the best ones. This report outlines some processes of the basics of genetic algorithms. The three most important aspects of using genetic algorithms are:

1. definition of the objective function,
2. definition and implementation of the genetic representation, and
3. definition and implementation of the genetic operators.

Once these three have been defined, the generic genetic algorithm should work fairly well. Beyond that many different variations can be tried to improve performance, find multiple optima (species - if they exist), or parallelize the algorithms.

### 5.1 Keyboard Representation

The representation of keyboard along with pre-occupied keys are shown in Appendix A.3

## Chapter 6

# Genetic Algorithm

Genetic Algorithm is adopted from theory of natural selection. Theory says that the best of the population only survive termed as ‘Survival of the fittest’ rest are selected by the nature. Offspring not necessarily but generally possess better quality than their parents. Hence the improvements are made from generation to generation.

Genetic algorithm is an iterative process. It is not an exact mathematical method to solve a well defined problem. It is a heuristic approach to solve for an optimality in a huge domain of possible solutions. As it is not an exact mathematical method it does not guarantee the best solution but according to the theory of the natural selection it ensures the solution near to the best ones. The solution approaches best as number of generation increases. It also depends on other factors like the crossover probability, population size and mutation probability. Generally mutation probability is very low compared to the crossover probability. And the solution approaches its best faster as the population size is increased.

Most difficult and prominent part of the implementation of genetic algorithm is the representation of the problem or the design of the chromosome(genome). Fitness function is the one which ensures the improvements in the chromosomes as the generation evolves. This is only the function which ensures the survivability of the genomes. So it must ensure all the constrains that must be took care of while optimizing the solution. Most visible and prominent qualities that should be considered at first place are:-

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

The problem is not completely solvable by the computer algorithms since it includes human psychology and the movement capabilities of the fingers while typing. And it depends on the language pattern too. The algorithm implemented and the fitness of the genome is computed by the weighted contribution of the individual six criteria (coefficients). And the final score of the keyboard is taken as the weighted sum of the six scores.

The Basic Algorithm for this approach based on the criterias above is as follows:-

start

1. generate a random population
2. calculate the fitness function based on something is here
3. new population
  - (a) select best population according to the fitness coefficient
  - (b) crossover the chromosomes of the parents with probability say ' $p_c$ '
  - (c) mutate the child chromosome with the probability say ' $p_m$ '
4. generate new population until the number of offspring is equal to the number of the parent population
5. Population produced in each one cycle is called a generation
6. Above process continued for a no. of generation, until a satisfactory condition is achieved

## 6.1 Optimization Characteristics

At a very macro level, an optimal keyboard layout 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. A concrete definition of the optimality of a keyboard is provided by [2]. 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:

1. Load Distribution
2. Modifier Overhead
3. Hand Alteration
4. Consecutive Usage of Same Finger
5. Big steps by fingers of same hand
6. Hit direction

## 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  $f_{m_i}^{desired}$  let  $m_i$  be the set of all monographs. So on this index the performance of the keyboard is measured as:

$$v_1 = \sum_{m_i \in m_1} (f_{m_i} - f_{m_i}^{desired})^2 \quad (6.1)$$

The values used for  $f_{m_i}^{desired}$  given in Table 6.1. To interpret the values the value for  $f_{m_i}^{desired}$  can be obtained by multiplying the respective fractions given in table 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. The computed value for the overall keyboard can be found at Appendix A.3

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
		4.17

Table 6.1: Ideal Load Distribution

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

## 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 keys 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:

$$v_3 = \sum_{d_i \in d_3} f_{d_i} dist(f_{d_i}) \quad (6.2)$$

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

## 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 digraphs which require both keys to be hit by the same finger. However, instead of simply summing up the frequencies of these digraphs, these frequencies are first weighted by a distance coefficient. The greater the distance between the two keys of a digraph, the more penalizing a consecutive usage. The relevant set  $d_4$  therefore the set of digraphs which are typed using the same finger. However instead of simply summing up the frequencies of these digraphs, these frequencies are weighted first by a distance co-efficient. The greater the distance between the two keys of a digraph, the more penalizing a consecutive usage. The relevant set  $d_4$  therefore the sets of a digraphs which are typed using the same finger. Therefore,

$$v_4 = \sum_{d_i \in d_4} f_{d_i} dist(f_{d_i}) \quad (6.3)$$

The distance co-efficient is given by the Manhattan distribution function  $dist(d_i) = |c_2 - c_1| + |r_2 - r_1|$

## 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 digraphs whose component keys have a vertical distance greater or equal to one. The relevant set  $d_5$  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:

$$v_5 = \sum_{d_i \in d_5} \Phi_i(d_i) f_{d_i}$$

The value for the co-efficients  $\Phi(i, j)$  were taken from the Table 6.1, which were derived from [2].

	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

Table 6.2: The values of  $\Phi$

## 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.  $d_6$  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:

$$v_6 = \sum_{d_i \in d_6} f_{d_i} \quad (6.4)$$

## 6.2 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  $v_{j,ref} (1 \leq j \leq 6)$ . Once the indicators are turned dimensionless they can be multiplied by a relative weight coefficient  $\Psi(j)$  and added. The values of weights  $\Psi(j)$  are represented in the Table 6.3.

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

$$V = \sum_{j=1}^6 \Psi(j) \frac{v_j}{v_{j,ref}} \quad (6.5)$$

## 6.3 Optimization Details

With the keyboard abstraction presented in the next section, optimization is essentially reduced to search to find the best keyboard based on the optimality factor. A Steady-State Genetic Algorithm is used for this search with many variations for parameters like mutation probability, population size etc. But a typical combination of the parameters is provided below:

- Population Size - 500
- Number of generations - 2000

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

Table 6.3: The values for  $\Psi(j)$  used for the computing the overall grade for a keyboard from  $v_j (1 \leq j \leq 6)$

- 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

## 6.4 Learnability

Learnability was the main issue when we designed the algorithm. In Nepali language, we have pairs of letters that sound similar. eg, *ka* and *kha*, *ga* and *gha*, etc. While computing the layout, we have made sure these pairs fall on the same key.

However, we were skeptic whether placing the complement would ruin the load distribution as the complement has modifier overhead (see Section 6.1). But statistics showed us that the ratio of frequency of main key vs. complement was usually greater than 5, which assured us that placing the pair on the same key has no noticeable effect on load distribution, but rather enhances learnability.

The complement pairs and the frequency ratio of main key vs. complement key is shown in Appendix A.4.

## 6.5 Genetic Operators

We have use the following selection, mutation and crossover operators for the optimization.

### 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. So we employed the Roulette Wheel Selection.

## 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  $c_2$ .

## 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  $(f_{p1}, f_{p2})$  and a random partition of  $C$  (the set of Nepali characters)  $C = C1 \cup C2$ , a child mapping  $f_c$  is defined as:

$$f_c(c) = \begin{cases} f_{p1} & , \forall c \in C_1 \\ nearest(f_{p2}(c)) & , \forall c \in C_2 \end{cases}$$

Here  $nearest(m)$  for a macro  $m$  is defined as a macro  $m_0$  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  $f_{p2}(c)$  may already be assigned to some other character while copying part of the mapping from  $f_{p1}$ .

## Chapter 7

# Results

The optimized keyboard layout obtained from above mentioned criteria can be found at appendix A.1. The entire code to compute the optimized keyboard layout was written in C++. The code can be obtained at website mentioned at Appendix A.2.

From the keyboard obtained and tests performed over it, we can be sure that this layout is better than any existing Nepali keyboard layouts. Such a claim can be made because the keyboard has been based on formal mathematical calculations and is modelled upon Nepali corpus statistics.

However, this layout can be more optimized if we go on to more generations of computations. Currently, due to lack of resources, we have optimized the keyboard only upto 2000 generations, which is a very small number as far as Genetic Algorithm's capability is concerned. Optimizing the keyboard to even more genetic generations will definitely lead to even more optimized and perfect layout.

# Bibliography

- [1] Priyendra S. Deshwal, Kalyanmoy Deb. *Design of an Optimal Hindi Keyboard for Convenient and Efficient Use*
- [2] Marc Oliver Wagner, Bernard Yannou, Steffen Kehl, Dominique Feillet, and Jan Eggers. *Ergonomic modelling and optimization of keyboard arrangement with an ant colony optimization algorithm*, Technical report, Laboratoire Gnie Industriel, cole Centrale Paris, France, 2001.
- [3] B.J. Oommen, J.S. Valveti, J.R. Zgierski. *Application of genetic algorithms to the keyboard optimization problem*, Technical Report, Carleton University, Ottawa, Canada 1989
- [4] Chad R. Brewbaker. *Optimizing stylus keyboard layouts with a genetic algorithm: customization and internationalization*, Dept. of Computer Science, Iowa State University
- [5] Masahito Yasui, Takumi Yamaguchi and Kazunori Shimamura. *The proposal about the software key input method for the operation device for the information weak*, Kochi University of Technology, Japan

# Appendix A

## Appendix

### A.1 Nepali Unicode Keyboard optimized using Genetic Algorithm

7	6	5	4	3	2	1	0		0	1	2	3	4	5	6	7
		1	2	3	4	5	SPC	0	SPC	6	7	8	9	0	-	=
		Q	W	E	R	T	SPC	1	SPC	Y	U	I	O	P	[	]
		अ ङ	उ ण	ँ ं	ए ऌ	ट ठ			य ज	ग घ	ओ औ	त थ	ब भ	द ध	ड ढ	
		A	S	D	F	G	SPC	2	SPC	H	J	K	L	;	"	\
		ह आ	ज झ	ट ण	यो	ति			न ल	म :	क ख	र ॠ			प फ	
		Z	X	C	V	B	SPC	3	SPC	N	M	,	.	/		
		व ष	ए ऐ	ँ ॠ	च छ	उ ऊ			न ऋ	स श						

 Doesn't exist  
 Existing Key

NOTE: For more optimized layout, please visit <http://www.jwalanta.com.np/nepalikeyboard>

### A.2 Website

The code, report and corpus used in this research can be found at <http://www.jwalanta.com.np/nepalikeyboard>

### A.3 Overall Ideal Load Distribution

7	6	5	4	3	2	1	0		0	1	2	3	4	5	6	7
0.0073	0.0095	0.0117	0.0318	0.0407	0.0273	0.0184	0.0273	0	SPC	6	7	8	9	0	-	=
0.0082	0.0107	0.0132	0.0358	0.0458	0.0307	0.0207	0.0307	1	SPC	Y	U	I	O	P	[	]
0.0199	0.0261	0.0322	0.0874	0.1119	0.0751	0.0506	0.0751	2	SPC	H	J	K	L	;	"	\
0.0063	0.0083	0.0102	0.0278	0.0356	0.0239	0.0161	0.0239	3	SPC	N	M	,	.	/		

Doesn't exist  
 Already assigned

## A.4 Monograph Frequencies

Main					Compliment		
Letter	f	f'	DWVC	DWWOC	Letter	f	fr
।	1419659	1.000000	0.107716	0.120012			
ॡ	1064821	0.750054	0.080793	0.090016			
र	980805	0.690874	0.074418	0.082913			
न	780493	0.549775	0.059220	0.065980			
क	756110	0.532600	0.057370	0.063919	ख	78800	9.5953
।	614843	0.433092	0.046651	0.051976	।	276226	2.2259
ॡ	511722	0.360454	0.038827	0.043259	ॡ	110827	4.6173
स	492183	0.346691	0.037344	0.041607	श	85241	5.7740
म	469168	0.330479	0.035598	0.039662			
ल	462898	0.326063	0.035122	0.039132			
त	449248	0.316448	0.034087	0.037978	थ	93910	4.7838
।	419666	0.295610	0.031842	0.035477	।	31196	13.4526
प	354778	0.249904	0.026919	0.029992	फ	39895	8.8928
य	335091	0.236036	0.025425	0.028327			
ॡ	272343	0.191837	0.020664	0.023023	ॡ	68518	3.9748
व	261419	0.184142	0.019835	0.022099			
ग	258105	0.181808	0.019584	0.021819	घ	31370	8.2278
द	248544	0.175073	0.018858	0.021011	ध	94481	2.6306
ह	248286	0.174891	0.018839	0.020989			
ब	197434	0.139071	0.014980	0.016690	भ	150765	
ज	161414	0.113699	0.012247	0.013645	झ	15334	10.5265
ए	140110	0.098693	0.010631	0.011844	ॡ	1323	105.9033
छ	138876	0.097823	0.010537	0.011740	च	96765	1.4352
ट	124482	0.087684	0.009445	0.010523	ठ	36137	3.4447
उ	104053	0.073294	0.007895	0.008796	ऊ	1041	99.9549
अ	99186	0.069866	0.007526	0.008385	आ	69073	1.4360
ष	83614	0.058897	0.006344	0.007068			
ॡ	76846	0.054130	0.005831	0.006496			
ड	66408	0.046777	0.005039	0.005614	ढ	20475	3.2434
ण	59462	0.041885	0.004512	0.005027			
ॡ	58875	0.041471	0.004467	0.004977	ॡ	46005	1.2798
ॡ	55471	0.039073	0.004209	0.004689			
ॡ	19753	0.013914	0.001499	0.001670			
ॡ	13938	0.009818	0.001058	0.001178			
ॡ	13836	0.009746	0.001050	0.001170			
औ	9683	0.006821	0.000735	0.000819	औ	2971	3.2592
:	4600	0.003240	0.000349	0.000389			
ॡ	1059	0.000746	0.000080	0.000090			

f = frequency  
f' = normalized frequency  
DWVC = Distributed Weight considering Compliment Letter  
DWWOC = Distributed Weight without considering Compliment Letter  
fr = ratio of Main Letter vs. Compliment Letter

## A.5 Nepali corpus analysis code

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

import os

# not value of variables not shown due to
```

```

# unicode incompatibilities in LaTeX
consonants = u"..."
vowels = u"..."
joiners = u"..."
othersymbols = u".."

# file and folder path
textfolder = "./nepaliarticles/" # folder in which files are kept
#resultfile = "frequency.txt" # file in which frequency is updates

filelist = os.listdir(textfolder)

# module definition
# this module calculates the freq of given str in all the files and updates the result to resultfile
def calculateAndUpdate(findString):
    count = 0
    for f in filelist:
        in_file = open(textfolder+f, "r")
        text = in_file.read()
        in_file.close()

    count += text.count(findString.encode('utf-8'))

    print findString, count

    out_file = open(resultfile, "a")
    out_file.write(findString.encode('utf-8')+"\t"+str(count)+"\n")
    out_file.close()

# calculate consonants
resultfile = "frequency-consonants.txt"
for i in range(len(consonants)):
    calculateAndUpdate(consonants[i])

# calculate vowels
resultfile = "frequency-vowels.txt"
for i in range(len(vowels)):
    calculateAndUpdate(vowels[i])

# calculate joiners
resultfile = "frequency-joiners.txt"
for i in range(len(joiners)):
    calculateAndUpdate(joiners[i])

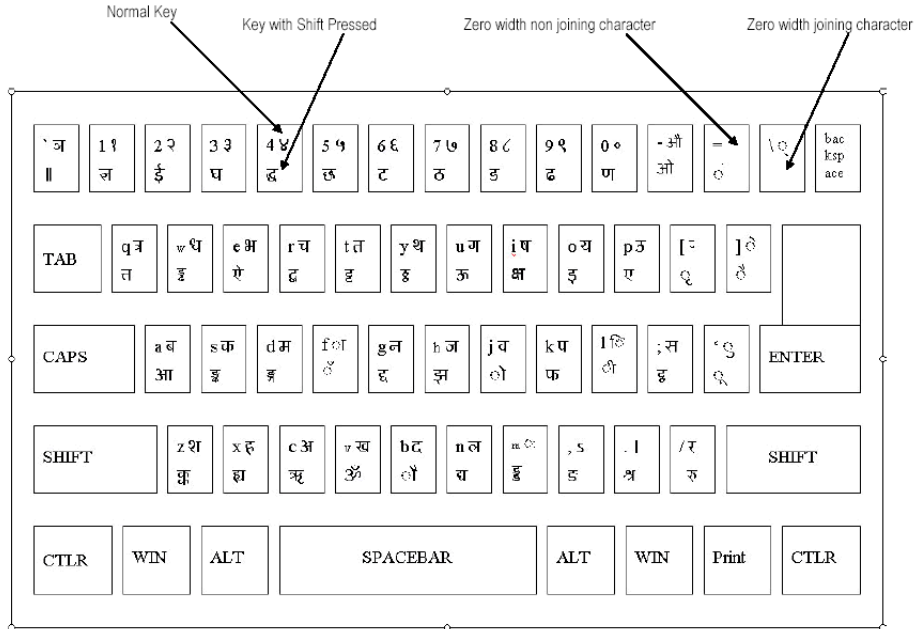
# calculate othersymbols
resultfile = "frequency-othersymbols.txt"
for i in range(len(othersymbols)):
    calculateAndUpdate(othersymbols[i])

# calculate consonant+joiner
resultfile = "frequency-consonant-joiner.txt"
for i in range(len(consonants)):
    for j in range(len(joiners)):
        calculateAndUpdate(consonants[i]+joiners[j])

# calculate consonant+consonant
resultfile = "frequency-consonant-consonant.txt"
for i in range(len(consonants)):
    for j in range(len(consonants)):
        calculateAndUpdate(consonants[i]+consonants[j])

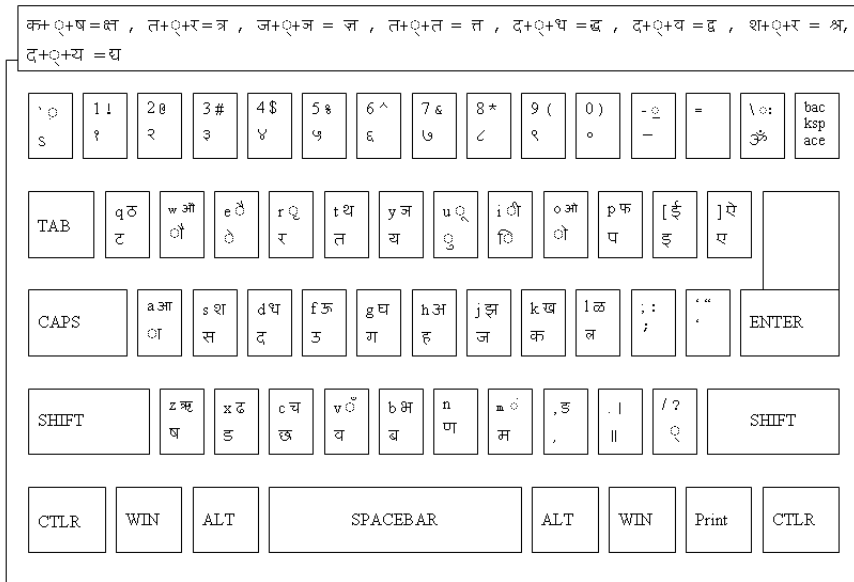
```

## A.6 Traditional Layout



Keyboard Layout for the new Devnagiri Font

## A.7 Romanized Layout



Nepali Unicode Keyboard Layout (Romanized)

## A.8 Devanagari Keyboard (Dp)

Esc	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12		
कृपया, सामान्य र शिफ्ट बन्दा बढी अक्षरहरूको लागि "क्याप्स लक" की थिच्नुहोस्, धन्यवाद!														
~ zwnj ` zwj	१ 1!	२ 2@	३ 3#	४ 4\$	५ 5%	६ 6^	७ 7&	८ 8*	९ 9(	० 0)	- =	+ =	 	←
Tab	Q जघ	W गघ	E यज	R सश	T एऐ	Y छच	U पफ	I ेे	O ोो	P तथ	[ उऊ	] ईइ	Enter ←	
Caps Lock	A मं	S िी	D नण	F ड	G टठ	H दध	J ाँ	K रू	L कख	:	;	"		
Shift	Z षत्र	X बभ	C ुू	V हढ	B ओ	N ओ	M लढ	<	>	? ।	Shift			
Ctrl	Win	Alt	Spacebar							Alt	Win	Print	Ctrl	