APPROXIMATE PROPER NAME MATCHING

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Computer Science Engineering

By

ANANT VIJAY ANEJA
AKASH PATKI
ROHIT SHIVSAGAR KUMBHALWAR

Department of Computer Science Engineering

National Institute of Technology

Rourkela

2007

APPROXIMATE PROPER NAME MATCHING

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Computer Science Engineering

By

ANANT VIJAY ANEJA
AKASH PATKI
ROHIT SHIVSAGAR KUMBHALWAR

Under the Guidance of

Dr. S. K. Jena
Dr. D P. Mahapatra

Department of Computer Science Engineering

National Institute of Technology

Rourkela

2007

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled, **"APPROXIMATE PROPER NAME MATCHING"** submitted by Mr. Anant Aneja, Mr. Akash Ramesh Patki, Mr. Rohit Kumbhalwar in partial fulfillment of the requirements of the award of Bachelor of Technology Degree in Computer Science Engineering at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by him under our supervision and guidance.

To the best of our knowledge, the matter embodied in the thesis has not been submitted to any other university / institute for the award of any Degree or Diploma.

Date:

Dr. S. K. Jena
Prof. & Head
Dept. of Computer Science Engg
National Institute of Technology,
Rourkela – 769008

Dr. D. P. Mahapatra
Asst. Professor
Dept. of Computer Science Engg
National Institute of Technology,
Rourkela - 769008

A C K N O W L E D G E M E N T

We wish to express my profound sense of deepest gratitude to our guides and motivators Dr. S. K. Jena, Prof. & Head, and Dr. D. P. Mahapatra, Asst Professor, Computer Science Engineering Department, National Institute of Technology, Rourkela for their valuable guidance, sympathy and co-operation and finally help for providing necessary facilities and sources during the entire period of this project.

We wish to convey my sincere gratitude to the faculty of Computer Science Engineering Department who have enlightened me during my studies. The facilities and co-operation received from the technical staff of Computer Science Engg. Dept. is thankfully acknowledged.

We express our thanks to all those who helped me in one way or other.

Last, but not least, we would like to thank the authors of various research articles and book that referred to.

Anant Aneja
Roll No: 10306028
National Institute of Technology
Rourkela

Akash Patki
Roll No: 10306013
National Institute of Technology
Rourkela

Rohit Kumbhalwar
Roll No: 10306026
National Institute of Technology
Rourkela

C O N T E N T S

A B S T R A C T

Approximate proper-name matching uses concepts of approximate string matching and applies them to special case of finding ‘close’ or ‘similar’ names, to an input name, from a large database of names. Such Proper-Name-Approximate matching finds applications in situations where a user is unsure of how a person’s name is spelled, such as in a telephone directory search system or a library search system where a user wishes to search books on an author’s name.

In this report we examine this problem in two main aspects: How to organize data efficiently, so as to obtain relevant results quickly, and how to develop suitable search techniques which would rank results suitably. We suggest four new data organization techniques to replace the current standard technique, Soundex, and we suggest refinements to the currently available search techniques.

We then assess the performance of the developed techniques and compare them against the currently available ones. We also show that the developed techniques provide us with better a result faster that is they take lesser time per query than the current methods. In the course of evaluation we also suggest a new assessment technique (weighted recall) which gives a better measurement of system performance than the standard assessment techniques.

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

INTRODUCTION

Finding the occurrence of given input string from a very large dataset is a fundamental problem in computer science. Simple string matching is the process of identifying a string or substring in a dataset (such as text) which is same as the input. It finds applications in various fields such as text processing & bioinformatics.

Approximate string matching, however, involves finding strings (and/or substrings) which may not be exactly same as the input, but be 'similar' to the input string. A very frequently used application of approximate string matching is an automatic spelling suggestion program where a user is presented with 'closely similar' words to the erroneous word. Other such applications include studying gene mutations, identifying subsequences in data, virus & intrusion detection, file comparison, optical character recognition, etc.

Approximate matching, when applied to proper nouns (i.e. names), however, generally involves finding similar sounding names to the given name. This is because, although spelling errors may exist in a name (either in the input or in the dataset itself), it is more likely that the name itself is spelled differently (the pronunciation remaining the same) by the person in question. Such Proper-Name-Approximate matching finds applications in situations where a user is unsure of how a person's name is spelled, such as in a telephone directory search system or a library search system where a user wishes to search books or an author's name.

There are two important issues, which are to be considered while developing such system: Speed of result retrieval and Precision of results.

The question regarding speed is largely one of data organization. If the dataset is suitably organized it would be easier to eliminate totally irrelevant results and retrieve only those results which are 'good' matches. Of course data organization also influences the 'recall' of the result set. This means that depending upon the selection scheme used to eliminate irrelevant results, some of the 'relevant' results might be lost. Since 'recall' is defined as the ratio of (a) number of relevant results retrieved to (b) the total number of relevant results; results missed out due to the selection scheme adversely affect 'recall'. We present here six different data organization schemes which were explored.

After such a limited set is considered we need to identify among this set, which are the results, the user would 'approve' as a 'similar sounding match'. The 'precision' of the system could then be defined as the ratio between (a) common number of matches obtained between to sets: the one which the user deems as 'approved' and the one produced by the system and (b) the total number of results retrieved. Thus, we need a system which has a high 'precision'

value. We present here seven such search techniques which would provide fairly precise results.

It is fairly obvious that since approximate name matching is a user-criteria based system, it is nearly impossible to develop a system which is 100% precise. It is this nature of 'inexactness' which makes approximate name matching similar to **information retrieval**. In fact, the definition of recall & precision are those taken from this very field.

The information given to us consisted of a text file of names (each name on a new line). The names were to be arranged into certain groups for reducing the search time. The groups were to be formed on the basis of certain characteristics-these characteristics are method dependent i.e. the method used to create the database and is unique to that particular method.

Once the database is formed then either the same method can be applied to search the name or different method could be applied.

Following are the different methods by which the database could be formed:

1. No Rep and order
2. No rep no order
3. Rep with order
4. Rep no order
5. Soundex
6. Q-grams

The first four methods are somewhat similar-since the all have the same basic concept of removing the vowels in a given name unless the name starts with a vowel.

The point where they differ is, while forming groups in some methods either repetitions or the order in which the consonants occur or both are considered.

Once the database is formed one of the following search methods is used to retrieve the set of possible answers

1. Edit Distance
2. Edit Distance Tapered
3. Editex
4. Editex Tapered
5. Ipadist
6. Ipadist Tapered
7. Q-gram

DATA STRUCTURE

- Hash table organization is used to store the names.

- Hash table has the following organization

  Key1 => Value 1

  Key2 => Value 2a,Value 2b, …

- So, for each name in the text file some processing is done and for a key generated the corresponding name is stored as one of the value.

DESIGN

The following diagram represents the flow of control adopted in the retrieval of the answers required

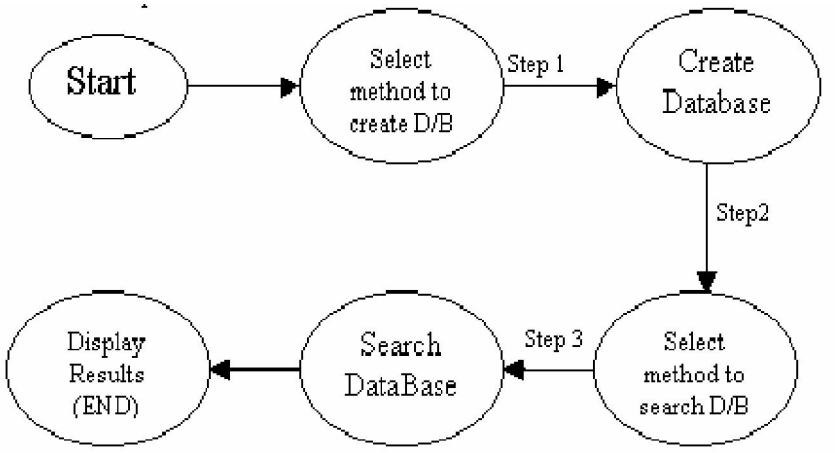

Fig 1.1. System Flowchart: flow of program control

Step 1

- Decides the method to create the database.
- The option for selecting the method is taken through the command line as the first argument.
- Once the method is selected and the database is created-from the names file, the entire database is stored in a ".txt" file for back up purposes. This is also very important for creating the database once again by reading through the previously created file-in the 1^{st} run.
- So in all for six methods there will be six ".txt" files.

Step 2

- Once the DataBase is created any one of the seven search methods is used to search the input string.
- The input for the search is now taken from the user at the prompt.
- The same method-used for creating the database is applied on the user input to create the key and further search process is carried depending on the method opted by the user.
- The search method option is taken from the user at the command-line as the second argument.

Step 3

- This step of searching through the DataBase is unique to each of the seven methods.
- But the one thing common to each of the methods is the two-level searching adopted.
- At Level One- Only the keys that satisfy a given constraint-1 are selected along with the corresponding set of values.
- At Level Two-Only those names satisfying constraint-2 are selected and separated out.
- Finally the separated out results are sorted according to relevance and printed out.

Chapter 2

DATABASE ORGANISATION

DATABASE CREATION

The section describes the each of the six DataBase creation methods

- The first four methods have almost similar functioning except when either repetition or/and ordering is considered.
- Vowels except for the first letter are removed in the first four methods.

2.1. No repetition and order important:

- Step 1: All the vowels are removed except if at occurs at position 1 in the word.
- Step 2: All the double occurrences in the word so formed are removed.

This method takes care of spelling mistakes.

Following flow chart shows how a key is formed for the inputs: “Indrajeet”, Shrivastava” and “Srivastava”.

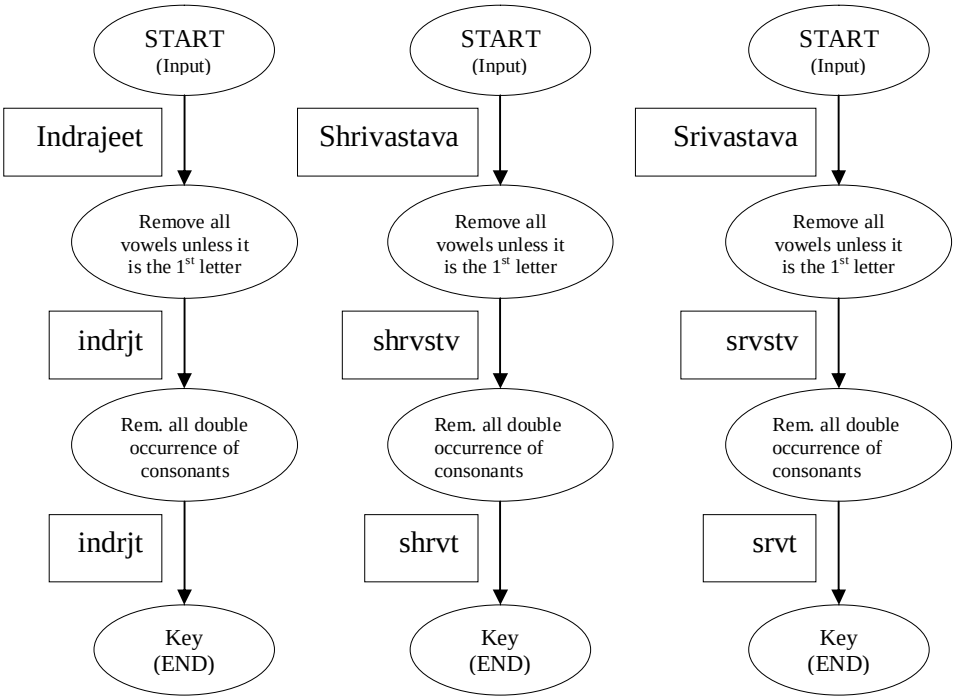

Fig 2.1 Flow of program to obtain key for sample input

indrjt: indrajeet, indrajit, indrajiet, …
…
shrvt: shreevastava, shrivastava, shreevastav, shreevastav, …
srvt: sreevastava, srivastava, sreevastav, srivastav….

Fig 2.2 Part of the hash table with key & corresponding values

2.2 No repetition and order not important

- Step 1: All the vowels are removed except if at occurs at position 1 in the word.
- Step 2: All the double occurrences in the word so formed are removed.
- Step 3: The ordering of the intermediate key, formed from step 2 is removed by arranging the letters in ASCII order

This method takes care of spelling mistakes and increases the scope of search

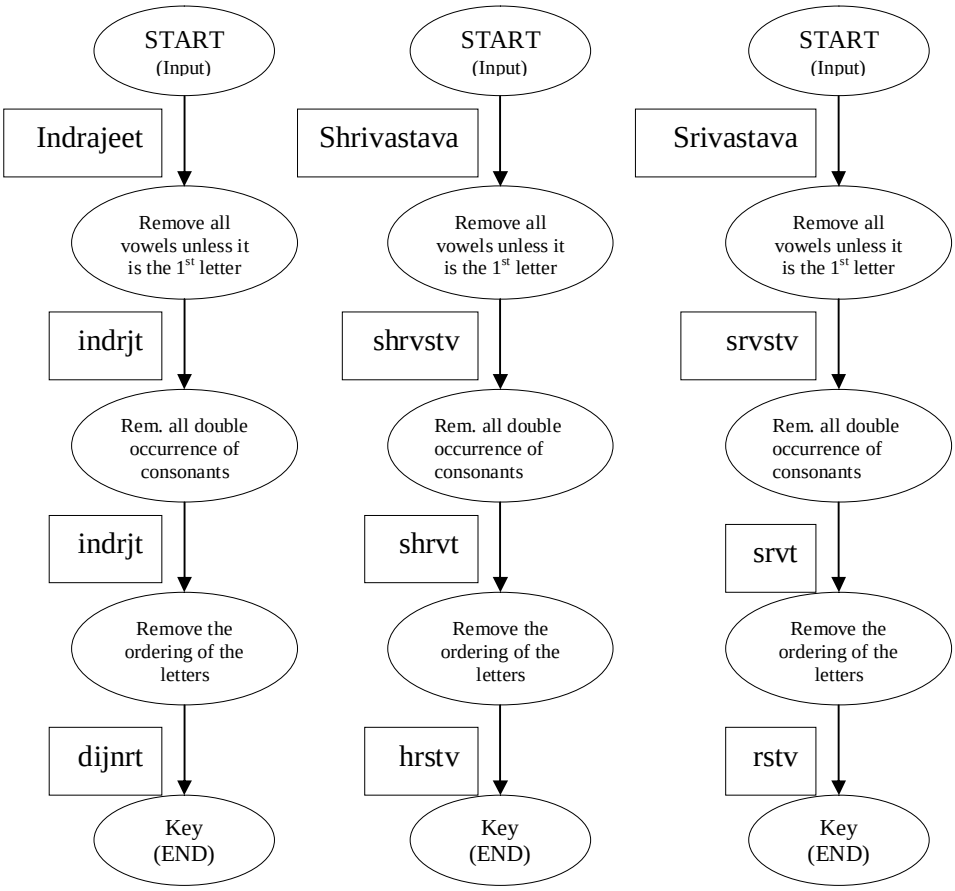

Fig 2.3 Flow of program to obtain key for sample input

…
dijnrt: indrajeet, indrajit, indrajiet, …
…
hrstv: srivathsa, srivasthava, srivathsa, savithri, …
…
…
rstv: sreevastava, srivastava, sreevastav, srivastav, ….

Fig 2.4 Part of the hash table with key & corresponding values

2.3. Repetition allowed and order important

- Step 1: All the vowels are removed except if at occurs at position 1 in the word

This is the only step required to obtain the key.

This method produces specific results.

Following flow chart shows how a key is formed for the inputs: “Indrajeet”, Shrivastava” and “Srivastava”

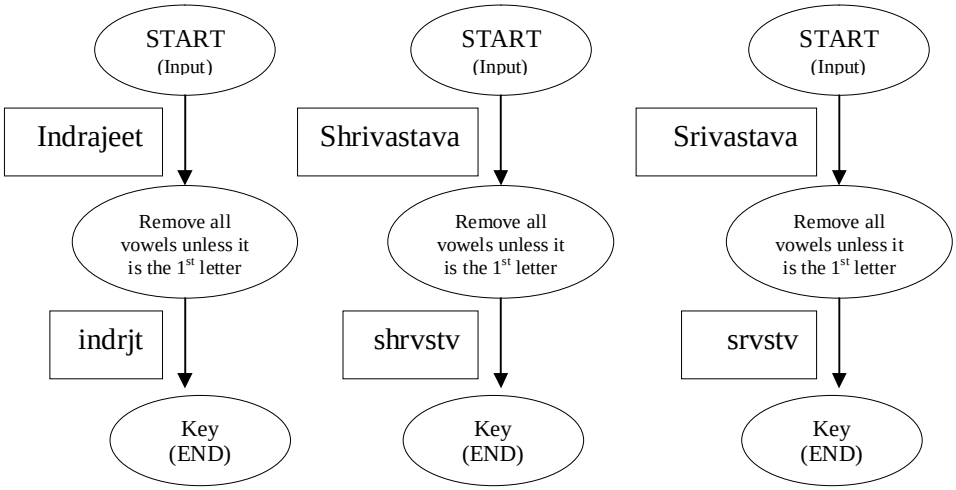

Fig 2.5 Flow of program to obtain key for sample input

….
indrjt: indrajeet, indrajit, indrajiet, …
…
…
shrvstv: shreevastava, shrivastava, shreevastav, shreevastav, …
srvstv: sreevastava, srivastava, sreevastav, srivastav….
…

Fig 2.6 Part of the hash table with key & corresponding values

2.4 Repetition allowed but order not important

- Step 1: All the vowels are removed except if at occurs at position 1 in the word.
- Step 2: The ordering of the intermediate key so formed from step 1 is removed by arranging the letters in ASCII order

This method is slightly more scope of search than the previous method since the ordering is not considered

Following flow chart shows how a key is formed for the inputs: “Indrajeet”, “Shrivastava” and “Srivastava”.

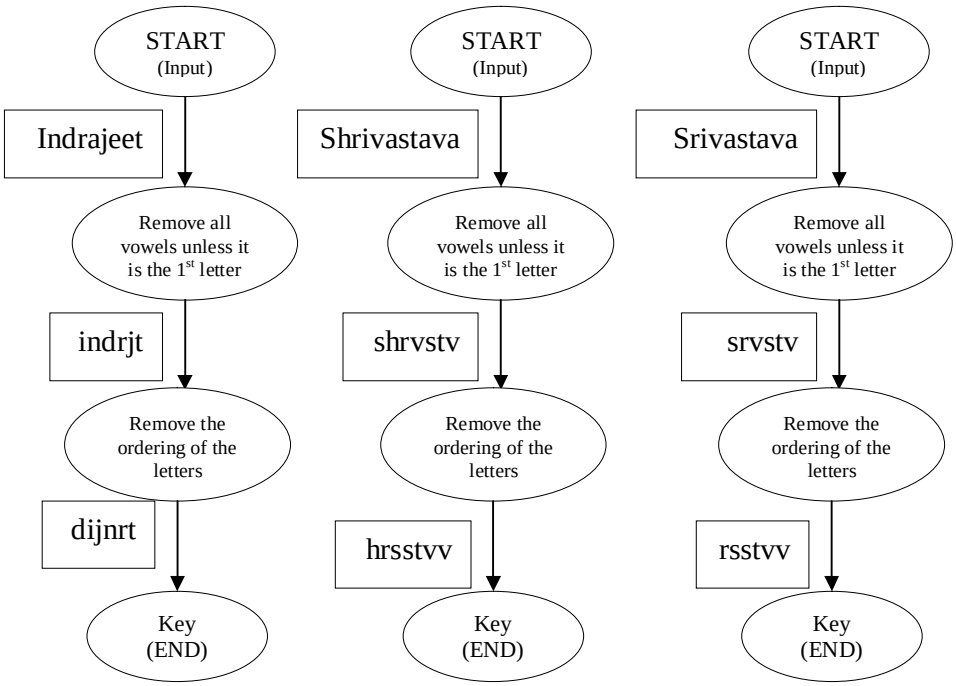

Fig 2.7 Flow of program to obtain key for sample input

….
dijnrt: indrajeet, indrajit, indrajiet, …
…
hrsstvv: shreevastava, shrivastava, shreevastav, srivasthava, …
…
rsstvv: sreevastava, srivastava, sreevastav, srivastav, ….
…

Fig 2.8 Part of the hash table with key & corresponding values

2.5 Soundex

Soundex is a phonetic algorithm for indexing names by their sound

The basic aim is that the names with the same pronunciation to be processed to a same string so that matching can occur despite minor differences in spelling.

- The method relies on generating a code for each word

- The Soundex code for a name consists of a letter followed by numbers: the letter is the first letter of the name, and the numbers contain information about the remaining consonants. Similar sounding consonants share the same number

The exact algorithm is as follows:

1. Retain the first letter of the string
2. Remove all occurrences of the following letters, unless it is the first letter: a, e, i, o, y, w, h, y.
3. Assign numbers to the remaining letters (after the first) as follows:

Soundex Code:	0	1	2	3	4	5	6
Letters:	a e i o u h w	b p	c g j k	d t	l	m n	r
		f v	q s x z				

Fig 2.9 Soundex codes

4. If two or more letters with the same number were adjacent in the original name (before step 1), or adjacent except for any intervening h and w then omit all but the first.

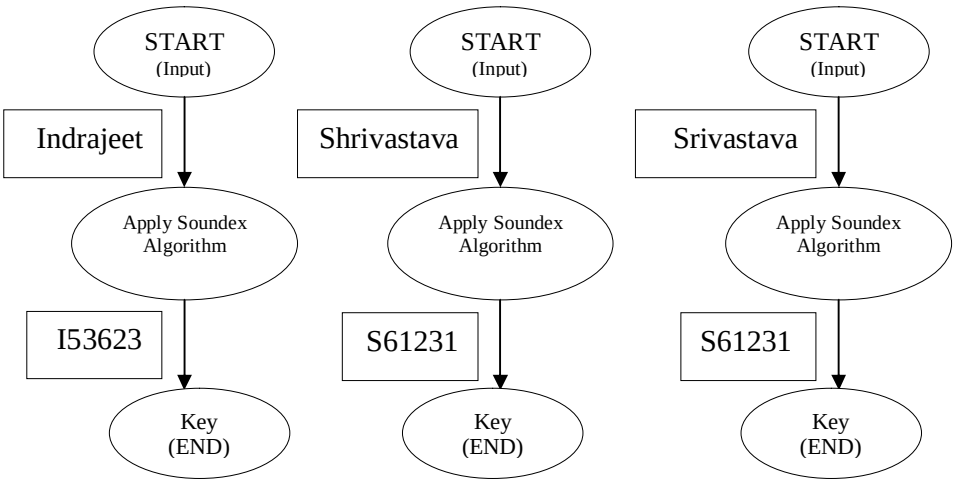

Fig 2.10 Flow of program to obtain key for sample input

….
I53623: indrajeet, indrajit, indrajiet, …
…
S61231: shreevastava, shrivastava, shreevastav, sreevastava, srivastava, sreevastav, srivastav srivasthava, …

Fig 2.11 Part of the hash table with key & corresponding values

2.6 Q-Grams

- This method aims at creating multiple keys each of length "3" from a given single name and then placing the name in each of the keys created.
- The method's underlying principle is that all the letters are important.
- Here, for given name, three consecutive letters are taken at a time (also called **tri-gram**) – starting from the first letter till the last letter is a part of a tri-gram.

For example,

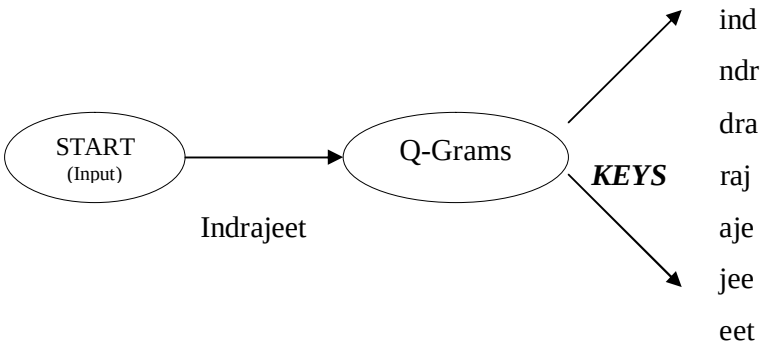

Fig 2.12 Flow of program to obtain key for sample input

- This means that the name "Indrajeet " will be placed in the 7 keys formed above
- So, for the above formed keys the hash table will look some thing like

…
ind: indrajeet, govind, indu, indira, …
ndr: indrajeet, chandra, nagendra, indra, …
dra: indrajeet, chandra, mahendra, nagendra, …
raj: indrajeet, raju, rajdeep, rajesh, …
aje: indrajeet, rajesh, …
jee: indrajeet, mukherjee, sajeev, chatterjee, …
eet: indrajeet, neeti, preeti, sabarjeet, …
…

Fig 2.13 Part of the hash table with key & corresponding values

- The method assumes no spelling mistakes are committed.

2.7 Summary

Following points are to be kept in mind for adopting one of the methods for creating the database

- It is desirable to have less no. of keys. This implies that on an average there should be more no. of values per key.
- Having less no. of keys and more values per key reduces the time to search through the database in Stage-I and it increases the scope of search, as more values are considered.
- But, reducing the no. of keys means the values associated with each key increases, this in turn increases the overhead of comparison and sorting in Stage - II of searching.
- Also, for a large no. of keys, the searching through the hash table becomes a linear search.
- A perfect balance has to exist between the no. keys and the average no. of values for each key.

The following table does a comparison of the important attributes for each of the six database creation methods

Table 2.1 Comparison of the database organization methods

	No rep and order imp	No rep and order not imp	Rep and order important	Rep and order not important	Soundex	Q-Grams
Total keys	15308	7902	19131	13688	8611	5664
Average key size (Char/key)	4.754	5.035	5.291	5.666	5.262	3.000
Average bin size (Values/key)	6.159	12.870	4.729	7.007	11.728	93.699
Max key length	11	11	15	15	12	3
Max bin size	999	1122	965	1083	1145	5517

Chapter 3

SEARCHING TECHNIQUES

SEARCHING METHODS

The next step in system design is implementation of searching methods. Once the database is created with any of the six data organization methods detailed above, one of the following search methods is used to search the user input against the database created

1. Edit Distance
2. Edit Distance Tapered
3. Editex
4. Editex Tapered
5. Ipadist
6. Ipadist tapered
7. Q-grams

- Although one of the above methods is applied to search through the hash table, the method applied for creating the hash is used to obtain the key for input- the user intends to search through the database.
- This means that a key is created for the name entered by the user by one of the database creation method.
- The further search operations are carried, on the key generated.
- These search operations are two levels, i.e the comparisons are done twice.
- Initially the comparisons are between the keys of the database and the key generated for the user input. This comparison is subjected to some constraint, say constraint no. 1.

This constraint selects only those keys that are close enough to the key generated for the user input i.e. the values corresponding values get selected for each of the key.

- In the second stage the comparisons are done between the actual user input and the selected values, again subjected to a constraint, say constraint no. 2. This second stage of comparisons ensures selection of only those values that are a close enough match for the user input.
- Once the second stage is completed the names short-listed are sorted according to the score calculated by the search method.
- The first six methods give the relevance in terms of numbers (integers) - lesser the number better the match.

The last method Q-Gram calculates the relevance in percentage – greater the percentage better the match.

The following section describes each of the search methods as to how each method works the algorithm behind each method and the pros and cons.

3.1. Edit Distance

Edit Distance also known as Levenshtein distance (LD) is a measure of the similarity between two strings, which we will refer to as the source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform source string into target string.

For example,

- If s is "test" and t is "tent", then LD(s,t) = 1, because one substitution (change "s" to "n") is sufficient to transform s into t.
- If s is "test" and t is "test", then LD(s,t) = 0, because no transformations are needed. The strings are already identical.

The greater the Levenshtein distance, the more different the strings are.

THE ALGORITHM

```
edit (0,0)=0
edit (i,0)=i
edit (0,j)=j
edit (i,j)=min[edit(i-1,j) +1, edit(i,j-1) + 1, edit(i-
1,j-1) + r (si,tj)]
```

Fig 3.1 Recurrence relation for minimal edit distance

EXAMPLE

The following matrix shows as to how edit distance is calculated and it represents the complete matrix for calculating edit distance between “GUMBO” & “GAMBOL”.

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	4
A	2	1	1	2	3	4
M	3	2	2	1	2	3
B	4	3	3	2	1	2
O	5	4	4	3	2	1
L	6	5	5	4	3	**2**

Fig 3.2 Matrix calculating the edit distance between “GUMBO” and “GUMBOL”

The bottom right corner represents the edit distance. In this case it is **2**.

3.2. Editex

Editex is a phonetic distance measure that combines the properties of edit distances with the letter-grouping strategy used by Soundex and Phonix.

- Editex was developed after running experiments with Soundex, Phonix, and edit distances, and observing the matches found by the phonetic methods and not the string methods: although Soundex and Phonix are not very effective, they do find good matches that standard edit distances cannot.
- Soundex and Phonix require letter groups with distinct codes to determine a canonical representation for strings; it follows that these groups must partition the set of letters.
- Editex also groups letters that can result in similar pronunciations, but doesn't require that the groups be disjoint and can thus reflect the correspondences between letters and possible similar pronunciation.
- Editex is defined by the edit distance recurrence relation of Fig 3.3 with a redefined function r(a, b) and an additional function d(a, b).

$$edit(0,0)=0$$
$$edit(i,0)=edit(i-1,0) + d(s_{i-1},s_i)$$
$$edit(0,j)=edit(0,j-1) + d(t_{j-1},t_j)$$
$$edit(i,j)=\min[edit(i-1,j) + d(s_{i-1},s_i),\ edit(i,j-1) + d(t_{j-1},t_j),\ edit(i-1,j-1) + r(s_i,t_j)]$$

Fig 3.3 Recurrence relation for minimal editex

- The function r(a, b) returns 0 if a and b are identical, 1 if a and b are both occur n the same group, and 2 otherwise. The groups are listed below,

Editex Code:	0	1	2	3	4	5	6	7	8	9
Letters:	a e i o u	b p	c k q	d t	l r	m n	g j	f p v	s x z	c s z

Fig 3.4 Editex code groupings

- The function d(a, b) is identical to r(a, b)—thus allowing pairs of the same letter to correspond to single occurrences of that letter--except that if a is h or w (letters that are often silent) and a = b then d(a, b) is 1.
- There is explicit similarity between the Editex and Phonix letter groupings; but while Phonix groups the letter h and w with the vowels, Editex handles these as deletions and Phonix does not group c and s

3.3. Q-Grams

Q-Gram method for searching is almost similar to the edit distance method except that the comparisons in q-grams are made in groups rather than one letter at a time as in the case of edit distance.

- Q-grams are string distance measures based on q-gram counts, where a q-gram of string s is any sub string of s of some fixed length q.
- A simple such measure is to choose q and count the number of q-grams two strings have in common.
- However, simply counting q-grams does not allow for length differences; for example, Fred has exactly as many q- grams in common with itself as it does with Frederick. So to address the problem, an q-gram distance which for strings without repeated q-grams (q-gram repeats are rare in names) can be defined as

  $$|G_s| + |G_t| - 2|G_s \cap G_t|$$

  Where,

  G_s: Set of q-grams in string s

  G_t: Set of q-grams in string t

  $G_s \cap G_t$: Set of q-grams common to G_s & G_t

Although this formula gives us a q-gram distance it does not tell us a percent match between two strings.

For example,

According to this formula the distance between rhodes and rod is 5 for q of 2 or 3

- A minor modification of the above formula gives us a fair idea of similarity between two strings

  $$2|G_s \cap G_t|/|G_s|+|G_t|$$

  Where, the symbols have their usual meanings as detailed above.

3.4. Edit Distance Tapered

Tapering is a refinement to the edit distance technique.

- It is based on the human factors property: “Differences at the start of a pronunciation can be more significant than differences at the end”.
- A tapered edit distance of particular interest is one in which the maximum penalty for replacement or deletion at the start of the string just exceeds the minimum penalty for replacement or deletion at the end of the string.

- Such an edit distance, in effect, breaks two ties : two errors always attract a higher penalty than one, regardless of position, but strings with one error are ranked according to the position at which the error occurs.

Implementation:

- Maximum Penalty (maxp): 2*(length(source)+length(target)), i.e., two times the sum of the length of the two strings.
- If,

  s: source string

  t: target string

  l1: length of source string

  l2: length of target string

  maxl = length of the longer string

  ED_T[][]=matrix representing the tapered edit distance cost,

Then, the following for loop calculates the tapered edit distance:

```
for 1 to l1 {
  for 1 to l2 {
    ED_T[i][j] = min3(ED_T[i-1][j] + maxp – i – j,
                      ED_T[i][j-1] + maxp – i – j,
                      ED_T[i-1][j-1] + (( equal(s[i],t[j]) ? 0 : maxp-i-j)
          //substitution cost + penalty (maximum penalty – positions)
          //if there is no substitution, penalty=0;
      }
    }
```

Where,

min3 (a,b,c) : finds minimum among a,b,c and returns it

equal(a,b) : returns 0 if a,b are equal

- The tapered edit distance is ED_T[l1][l2]

Consider for example,

Source : srivastava

Target : srivastav

l1=10

l2=9

maxl=10

maxp=38

The ED_T is as follows:

		s	r	i	v	a	s	t	a	v	a
	0	37	73	108	142	175	207	238	268	297	325
s	37	0	*	*	*	*	*	*	*	*	*
r	73	*	0	*	*	*	*	*	*	*	*
i	108	*	*	0	*	*	*	*	*	*	*
v	142	*	*	*	0	*	*	*	*	*	*
a	175	*	*	*	*	0	*	*	*	*	*
s	207	*	*	*	*	*	0	*	*	*	*
t	238	*	*	*	*	*	*	0	*	*	*
a	268	*	*	*	*	*	*	*	0	*	*
v	297	*	*	*	*	*	*	*	*	0	**19**

Where, * is a value which is greater than 0 & not needed for this example.

- ED_T[9][10] = 19 is the tapered edit distance

3.5. Editex Tapered:

- The same tapering scheme is applied to the Editex method.
- The values obtained are nearly three times those obtained edit distance tapered algorithm.

3.6. IpaDist:

- IpaDist is a **phonometric** search method developed by Justin Zobel (RMIT, Australia) and Philip Dart (University of Melbourne, Australia)
- IPA is the International Phonetic Algorithm. The strings are converted into phonetic codes as defined by the IPA.
- The codes, called **phonemes** are then compared by assigning distance values between different phoneme pairs. An editex like algorithm is used.

3.7. IpaDist Tapered:

- The tapering scheme when applied to IpaDist gives us this modified method.
- This method, however, seems to give us inaccurate results.

3.8. Hindex:

- Transliteration refers to the conversion of a string from one language to another. (E.g. English to Hindi)
- It is important to capture the pronunciations in the native language of the name.

- To find the Hindex distance between s1 and s2 we first convert the consonant/consonant groups of both the strings into their Unicode Hindi representation based on Harvard-Kyoto transliteration scheme, a standard 'English to Hindi Transliteration scheme'.
- The character groupings are as follows:

k kh g gh G

c ch j jh J

t Th D Dh N

t th d dh n

p ph b bh m

y r l v

z S s h

  - We apply the Editex algorithm by but use the above character groupings rather than the standard ones.
  - We replace the all instances of the 'd' function by the 'r' function.
  - His modification of Editex is termed:Hindex

3.9. Gram analysis:

- The method takes into account the various points at which possible errors occur during pronunciation to representation.
- Variants of the same name can be identified by suitable analysis to find 2-grams or 3-grams which could be possibly misspelled or confused for the same pronunciation.

- e. g

  aa a

  ph f

  sh s

  th t

  ky ki ……..

  ci si

  ava av

  aks ax

  etc,

  - This means that the initially occurring n-gram can be replaced safely without altering the pronunciation, much.
  - Any search method is used after the n-gram analysis function is applied when this particular search method altering technique is chosen.

Chapter 4

METHODS FOR PERFORMANCE ASSESSMENT

METHODS FOR PERFORMANCE ASSESSMENT

Grading results with an approximate measure implies human-factors component in judging them as 'relevant' or 'irrelevant'. In proper-name matching, this translates into grading results into 'similar-sounding' and 'not-similar-sounding' categories. Parallels to information retrieval are therefore not just incidental, in fact, the metrics employed for assessment are from this very field. A test bed of 28 queries was created. A spelling mistake was purposely introduced in these queries. For each such test query the entire database was manually scanned and the results deemed 'relevant' for this query were noted. Performance metrics for each of the 28 queries were obtained, and the average of these for a particular hash organization-search scheme combination was calculated. The metrics used were:

4.1 Recall:

- It is the ratio of the relevant results retrieved to the total number of relevant results (in a pre-defined result set).
- It is a measure of 'false negatives', i.e. it is also an indicator of which results were marked as 'irrelevant' but were supposed to be marked as 'relevant' by the system.

$$\textbf{Recall} = \frac{|\{\textbf{relevant documents}\} \cap |\{\textbf{retrieved documents}\}}{|\{\textbf{relevant documents}\}|}$$

4.2 Precision:

- It is the ratio of the 'relevant results' (from the retrieved result set) to the total number. of results retrieved.
- It is a measure of 'false positives', i.e. it is also an indicator of which results were marked as 'relevant' but were supposed to be marked as 'irrelevant' by the system.

$$\textbf{Precision} = \frac{|\{\textbf{relevant documents}\} \cap |\{\textbf{retrieved documents}\}}{|\{\textbf{retrieved documents}\}|}$$

4.3 Weighted Recall:

Since some results were deemed to be of more importance than others, a weighted recall scheme was considered:

- Each ‘relevant result’ set was divided into subsets. The no. of subsets for each query was decided individually based on the need to grade some results more extremely important than other subsets.
- For 4 subsets, the subset weights were, Set 1: 40%, Set 2: 30%, Set 3: 20%, Set 4: 10%
- For 3 subsets, Set 1: 50%, Set 2: 30%, Set 3: 20%
- For 2 subsets, Set 1: 70%, Set 2: 30%

Such a weighted-scheme of assessment was considered more reliable as a performance metric than simple recall.

4.4 Time analysis:

The time taken to execute a query is of utmost importance to measure the effectiveness of a data organization scheme. For each combination of ‘data-organization’ and ‘search scheme used’ the average time to execute a query was calculated by obtaining the run-times of 28 (different length) queries and then averaging the values obtained.

Chapter 5

EXPERIMENTAL RESULTS

EXPERIMENTAL RESULTS

Weighted Recall analysis: A program was written which calculated weighted recall values for every 'data organization method'-'search method' combination. The results were plotted in two graphs: one with the size of the results set=20 and one with the result set size=50.

Time analysis: Average time to execute a query was calculated for each 'data organization method'-'search method' combination. This was also plotted in two graphs: one for 'time taken vs. hash organization' and another for 'time taken vs search method'.

5.1 Weighted recall for 20 results set:

Table 5.1 Weighted recalls for hash scheme-search method combination. Result set size= 20.

Hash scheme	Search method	Recall
No Rep and order	Edit Distance	33.15344774
	Edit Distance Tapered	30.52673418
	Editex	31.65807308
	Editex Tapered	28.46182488
	Ipadist	32.88291074
	Ipadist tapered	30.83468615
	Hindex	25.87022006
Soundex	Edit Distance	31.48993764
	Edit Distance Tapered	30.03566275
	Editex	31.36045403
	Editex Tapered	28.50646774
	Ipadist	32.69583591
	Ipadist tapered	30.83468615
	Hindex	28.83249588
No rep no order	Edit Distance	31.51364409
	Edit Distance Tapered	30.39280561
	Editex	32.09561173
	Editex Tapered	28.50646774
	Ipadist	32.94204803
	Ipadist tapered	30.79004329
	Hindex	32.03595908
Rep with order	Edit Distance	30.34094774
	Edit Distance Tapered	30.16959132
	Editex	31.75547568
	Editex Tapered	28.46182488
	Ipadist	32.98031334
	Ipadist tapered	30.83468615
	Hindex	30.54283911
Rep no order	Edit Distance	30.06961194
	Edit Distance Tapered	30.08030561
	Editex	32.09561173
	Editex Tapered	28.50646774
	Ipadist	32.89740517
	Ipadist tapered	30.79004329
	Hindex	31.61929241

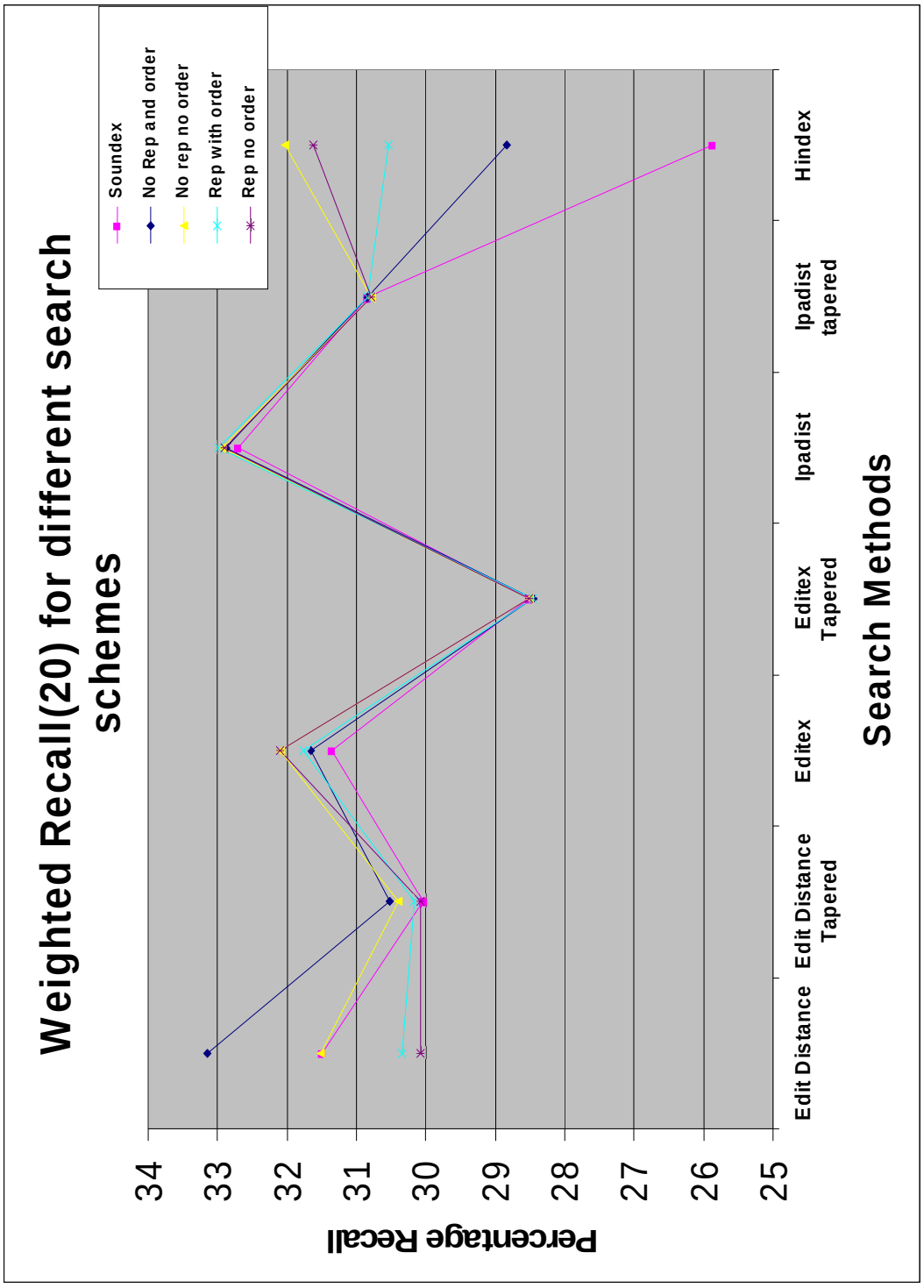

Fig 5.1 Percentage-weighted-recall values for different search methods. Each series of the graph is a particular data-organization scheme.

5.2 Weighted recall for 50 results set:

Table 5.2: Weighted recall values for a hash scheme-search method combination. Result set size is 50.

Hash scheme	Search method	Recall
No Rep and order	Edit Distance	37.39861626
	Edit Distance Tapered	34.5180504
	Editex	40.14882241
	Editex Tapered	37.60018553
	Ipadist	38.33936302
	Ipadist tapered	36.89812667
	Hindex	37.77066584
Soundex	Edit Distance	36.86373944
	Edit Distance Tapered	34.5180504
	Editex	40.59525098
	Editex Tapered	38.18054267
	Ipadist	38.74520717
	Ipadist tapered	36.89812667
	Hindex	34.87744795
No rep no order	Edit Distance	38.76334776
	Edit Distance Tapered	34.5180504
	Editex	39.73891981
	Editex Tapered	38.0466141
	Ipadist	38.6599799
	Ipadist tapered	36.89812667
	Hindex	38.22327871
Rep with order	Edit Distance	34.37643012
	Edit Distance Tapered	34.5180504
	Editex	40.28275098
	Editex Tapered	37.60018553
	Ipadist	38.20543445
	Ipadist tapered	36.89812667
	Hindex	36.3371341
Rep no order	Edit Distance	35.33647702
	Edit Distance Tapered	34.2055504
	Editex	39.73891981
	Editex Tapered	37.91268553
	Ipadist	38.30283704
	Ipadist tapered	36.7641981
	Hindex	37.86117553

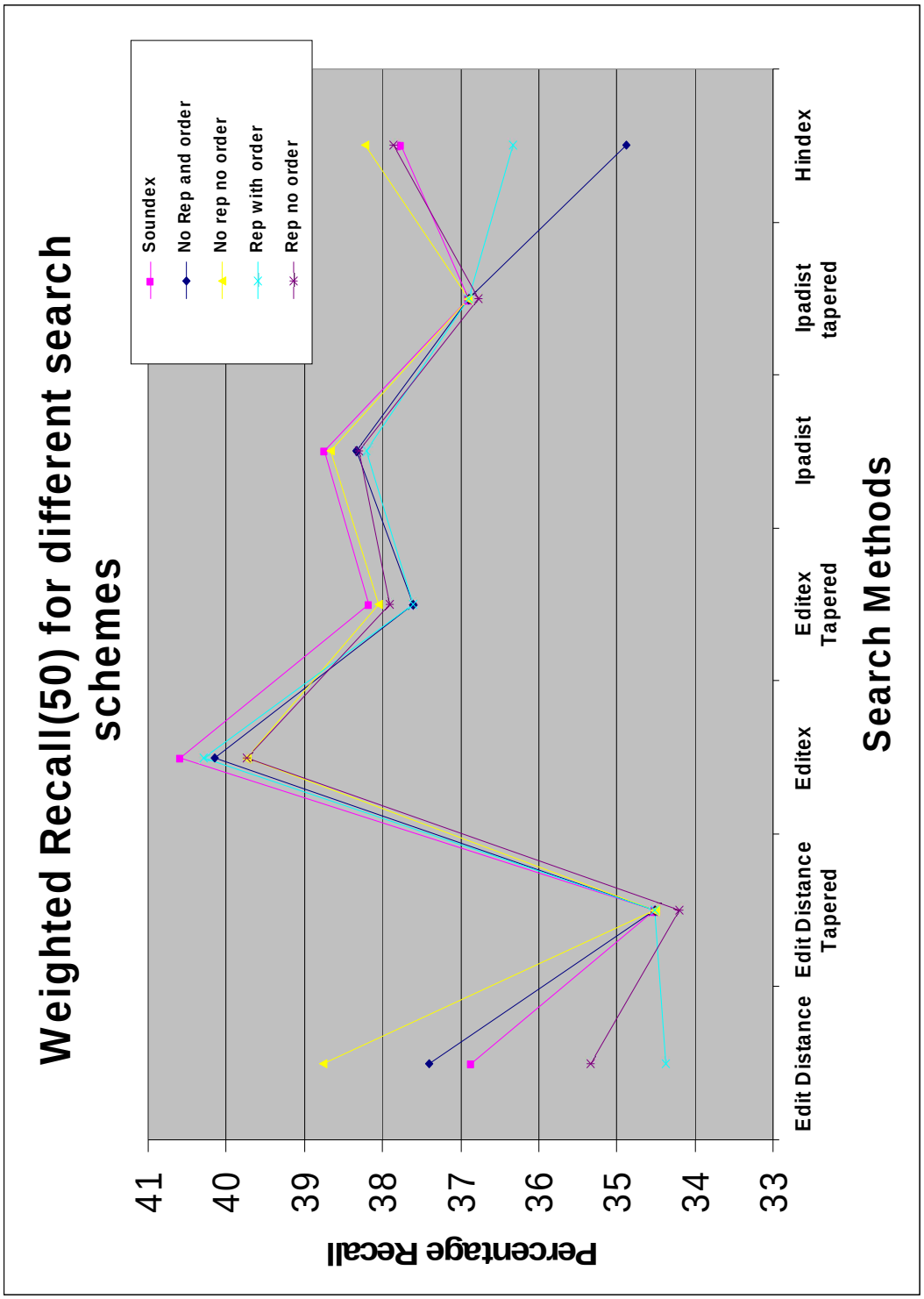

Fig 5.2: Percentage-weighted-recall values for different search methods. Each series of the graph is a particular data-organization scheme.

5.3 Time Analysis:

Table 5.3: Execution time values for a hash scheme-search method combination

Hash Scheme	Search method	Time Taken
No Rep and order	Edit Distance	0.473374239
	Edit Distance Tapered	1.520261066
	Editex	0.740915452
	Editex Tapered	1.159738966
	Ipadist	0.818892172
	Ipadist tapered	1.514890943
Soundex	Edit Distance	0.346088265
	Edit Distance Tapered	1.283133362
	Editex	1.564180255
	Editex Tapered	1.627933392
	Ipadist	1.177978482
	Ipadist tapered	1.528901679
No rep no order	Edit Distance	0.31172439
	Edit Distance Tapered	1.245935559
	Editex	0.642552461
	Editex Tapered	0.980184291
	Ipadist	0.711250041
	Ipadist tapered	1.306379676
Rep with order	Edit Distance	0.568913485
	Edit Distance Tapered	1.301637028
	Editex	0.782405785
	Editex Tapered	1.060655398
	Ipadist	0.848563739
	Ipadist tapered	1.350011451
Rep no order	Edit Distance	0.436129502
	Edit Distance Tapered	1.154374199
	Editex	0.840347895
	Editex Tapered	1.003445549
	Ipadist	0.773821371
	Ipadist tapered	1.233742595

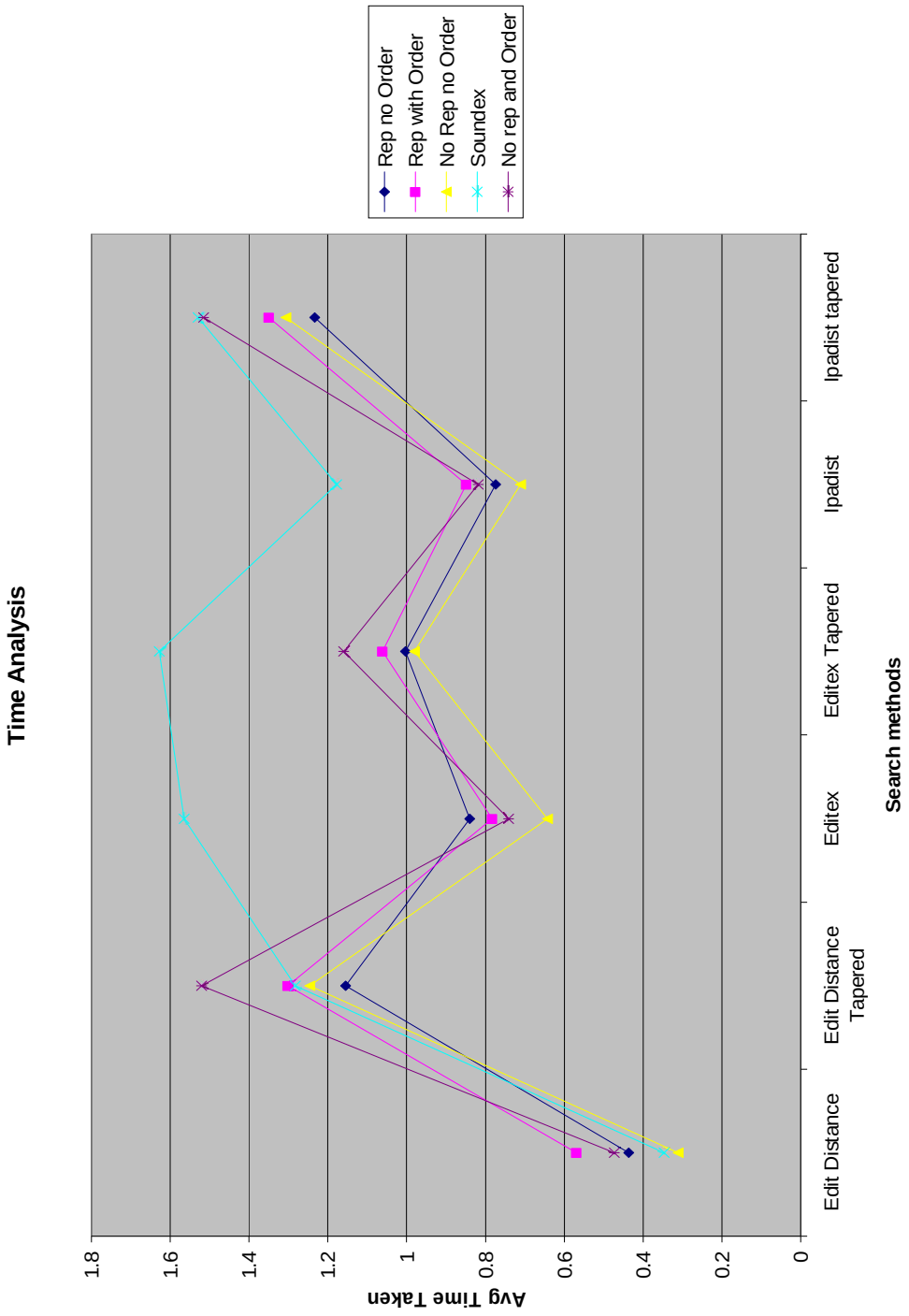

Fig 5.3: Execution time for different search methods for a particular hash organization scheme

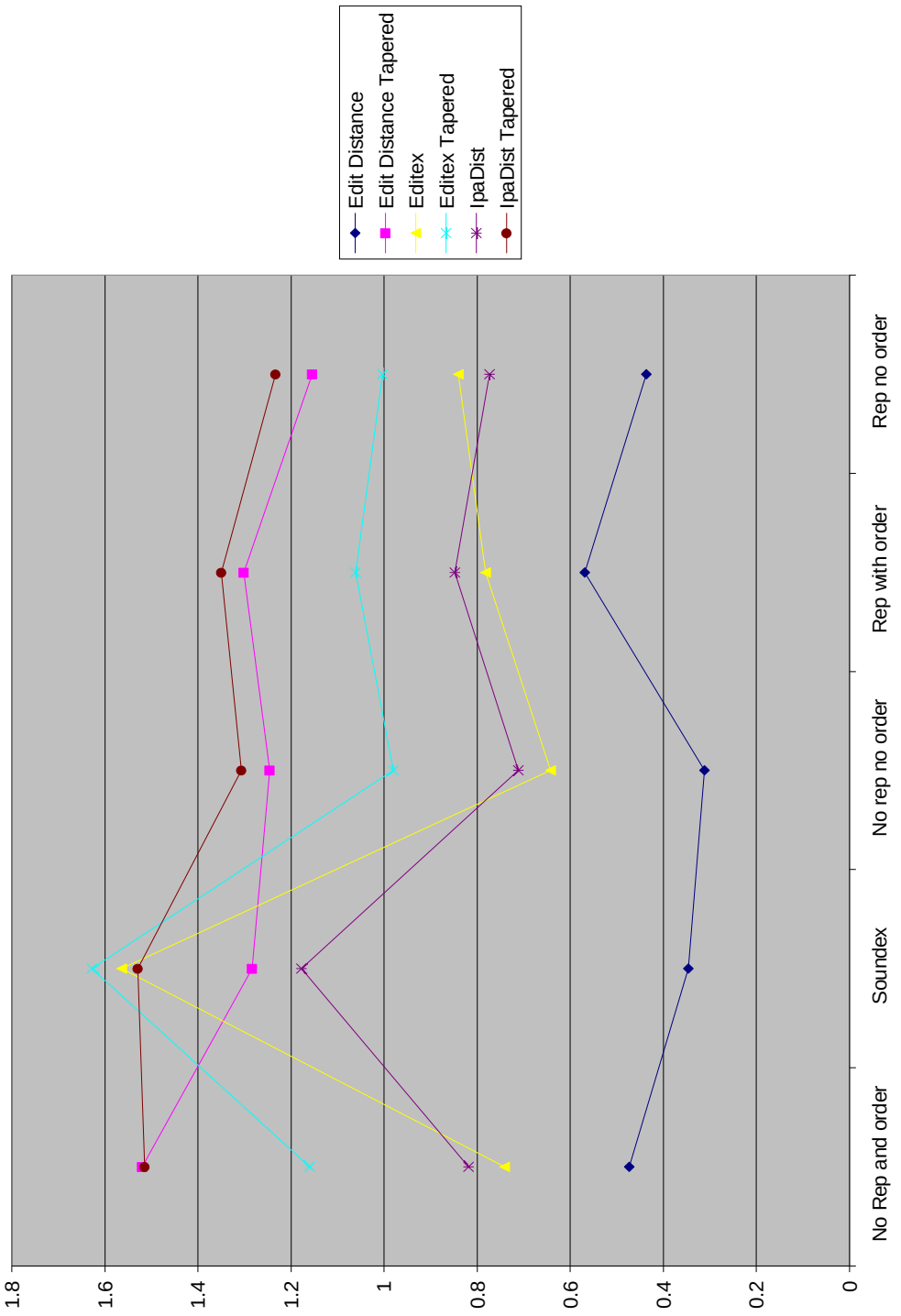

Fig 5.4: Execution time of different hash organization schemes for a constant search technique

Chapter 6

CONCLUSION

6. CONCLUSION

From the calculation of the weighted recall values, it is seen that the data organization techniques have a large role to play in the 'quality' of results obtained. The 'no-rep-order' data organization scheme provides the best recall values for all the search schemes. This implies that the order of the consonants in the names has an important effect on the pronunciation. Also if the same consonant occurs consecutively, the extra occurrence(s) can be safely overlooked. This scheme also has an optimal words/bin size, which is better than Soundex or Phonix, which are the generally used methods for grouping similar sounding names. Thus we suggest the 'no-rep-order' as a newer and better data organization scheme.

The searching techniques, while based on simple approximation-based string matching methods (edit-distance), still provide the best search results. Changes in these methods (editex) to account for pronunciation of the names yield only slightly better results.

Better results may be obtained if the stress on pronunciation is more than subtle. IpaDist which is based on the International Phonetic Algorithm accomplishes this by converting names into phonetic codes. However, this too fails to provide good results. This is perhaps due to the fact the phonetic codes that IPA uses do no properly capture the nature of the sound produced. Transliteration of names from the initial source language to the original language of the names and then application of the distance-based techniques, should give better results. Hindex, a modification of editex applicable to Indian names is a step in this direction. While such a transliteration based algorithm reduces the applicability to only Indian names, studying such an approach provides us with an insight into the native language of the name.

Initial testing has shown that this algorithm is giving slightly inferior results to IpaDist but further refinements (using a modified transliteration scheme, studying Hindi sounds and accounting for them more accurately) should provide us with improved recall values. This is the current focus of our research.

A simple way to improve recall for all the search methods is to accommodate for spelling errors and pronunciation variations before any search method is applied. This was done by manually performing n-gram analysis on the dataset and observing the possible mistakes in spelling and similarities in pronunciation. After accounting for these, the system was tested for a few queries and it was observed that some results had improved 'match values', i.e. they had a better rank than before. Thus better ordering of results was achieved, which also mildly affected system recall. The hash organization efficiency can be improved by implementing the reverse indexing scheme. On the search technique side, Editex-Bloom seems promising.

REFERENCES

1. Justin Zobel and Philip Dart “Phonetic String matching: lessons from Information Retrieval”, SIGIR'96,Zurich ,pp. 105-110, 1996.
2. *Pattern Matching Algorithms,* Alberto Apostolico & Zvi Galil, Oxford University Press, UK, 1997.
3. E. Ukkonen, Algorithms for approximate string matching. Information and Control 64, 100-118. 1985.
4. R. Baeza-Yates and G. Navarro. Fast Approximate String Matching in a Dictionary. Proc. 1998.
5. V. I. Levenshtein, *Binary codes capable of correcting deletions, insertions, and reversals.* Soviet Physics Doklady 10 (1966)
6. Zobel, J. and Dart, P. [1995]. Finding approximate matches in large lexicons. Software-Practice and Experience,25(3):331-345.
7. Zobel, J. and Dart, P. [1996]. Fnetik: An integrated system for phonetic matching. Technical Report 96-6, Department of Computer Science, RMIT.
8. Ukkonen, E. [1992]. Approximate string-matching with q-grams and maximal matches. Theoretical Computer Science, 92:191-211.