A Robust Method for Searching Across Mixed Transliterated Tamil Texts

K. Rajaraman
21 Heng Mui Keng Terrace
Laboratories for Information Technology
Email: kanagasa@lit.org.sg

V.S. Senthilkumar
Array Networks
San Jose CA, USA
Email: vssenthilkumar@yahoo.com

1. Introduction

We have seen a tremendous growth of electronic text documents during the last few years, powered primarily by the Internet. This is expected to grow further during the coming years and hence there is growing need for tools to ease the problem of locating desired information. Search engines are one such class of tools that have been found to be extremely useful. Famous search engines such as Google, Yahoo and Altavista are getting 10's of millions of queries daily. While most of these engines are targeted for the English language, support for native languages is being provided on some.

A pioneering effort for tamil search was the work on *i*Agent [1]. *i*Agent is a system that could accept search queries in tamil, crawl the web and retrieve documents on a specific document encoding, say TSCII. Since then several websites have started supporting such encoding specific tamil search, e.g. Dinakaran (http://www.dinakaran.com) and WebUlagam (http://www.webulagam.com).

In contrast to documents in specific encodings, there is another class of `tamil' documents created by transliterating tamil words with roman alphabets. These so-called transliterated or romanized texts have been in vogue even before tamil fonts came into existence. The class of such documents on internet is potentially much bigger than that of non-romanized (i.e. document encoded) tamil documents, at least presently. The primary reason being, publishing is easier in romanized form as the user can choose a transliteration scheme convenient to him/her and need not have to go through unfamiliar tools for word processing in tamil. The number of such transliterated documents is expected to grow further until tamil word processing becomes much more widespread and layman friendly. Besides, support for simultaneously creating transliterated texts along with tamil native script texts is also being adopted, e.g. “Project Madurai” [2]. Thus, enabling easier access to transliterated documents is important.

In this paper we consider the problem of searching tamil transliterated documents. At first, the problem looks deceptively simple since any standard web search engine can be used. It is not really so since there are several transliteration schemes in vogue and hence there may not be any retrieved documents if the scheme used in the search query is different from the one used in the documents being searched. For example, if you are searching for *Project Madurai* homepage on Google (http://www.google.com) and suppose you used the queries ‘project madhurai’ or ‘project mathurai’, Google will not return the correct link. Simply because neither ‘madhurai’ nor ‘mathurai’ matches exactly with 'madurai'. This has been a frustrating problem faced by many people searching with transliterated tamil keywords. To minimize this problem, INFITT (http://www.infitt.org) has taken an initiative with a working group on standardization of transliteration schemes [3].

Google provides a partial solution to the problem though a spelling correction system [4]. Given a search query, it can suggest a new query that can possibly return more relevant results. For the 'project madhurai' query example, it actually suggests the right query 'project madurai'. However, it fails when the query is 'project mathurai', for which it suggests 'project mathura'. This is to be expected because the Google's spelling correction is not tailored for tamil. Also, it is not guaranteed to return any spelling corrected query. Sometimes it may not return any correction at all, e.g. for 'silappadhigaaram'. Another approach to the problem is to make use of a dictionary of tamil words together with different transliterated variations for each word. When the user enters a query, the right tamil word may be substituted by scanning this dictionary. This approach would work nicely, provided the dictionary is exhaustive enough. However, to exhaust even the common tamil words and list their most prevalent transliteration variations would involve huge amount of human effort and hence is not an attractive approach. In this paper we propose a new approach to address this problem using Artificial Intelligence (AI) and Information Retrieval (IR) principles.

We seek to develop a tamil search system that has the following features: (1) It is robust enough to tackle the transliteration variations in the user query, and (2) does not require lot of human resources to capture the language specifics in a knowledge base that can be used for intelligent search.

2. Our Tamil Search System

Our search system is based on a rule-based approach. The system consists of three components: (1) Rule Base, (2) Query Refiner, and (3) a Generic Search Engine.

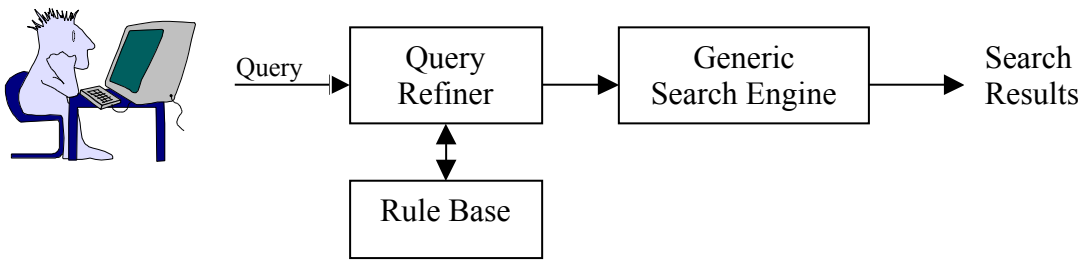

Figure 1: Our Tamil Search System.

The Rule Base and the Query Refiner form the core of the system. The components interact as shown in Figure 1. The input to the system is the user query. The Query Refiner parses it and uses the Rule Base (that captures the language specifics) to generate a new query. The new query is in turn passed to the Generic Search Engine to perform a traditional search. The returned results are shown to the user. In the actual implementation, the whole process is transparent to the user. The user will see only a search box and get search results as in a typical search engine. Each of the components is discussed in more detail below.

2.1 Rule Base (RB)

This is the most important component of the system. It is meant to capture the transliteration variations commonly found in tamil. The Rule Base is organized as an ordered list of IF-THEN type rules. Each rule has an antecedent and a precedent. The antecedent is a regular expression pattern and the precedent a list of text strings. Each rule is meant to

represent one specific transliteration variation. For example, the rule to represent the 'ò' variation (as in madurai/madhurai/mathurai) would be:

$$(d \mid t)h? \Rightarrow dh,th,d,t \quad (1)$$

The left hand side (LHS) of the rule (antecedent) is a regular expression that is designed to match the most common variations of 'ò'. (Actually, we use a more complex rule since the above is not effective for, say 'siththar'.). The right hand side (RHS) of the rule (precedent) is a list of substitution strings that are possible candidates to consider while generating the refined query. The way the rules are used will be explained below under Query Refiner.

An important issue is the order of the rules in the Rule Base. When there are several rules to consider, the order helps in selecting the most important rules. In our current implementation, we have used simple heuristics such as the occurrence frequency of transliteration variation and the number of RHS substitution strings to order the rules.

2.2 Query Refiner (QR)

This component works together with the Rule Base. Given a query, it treats each word in the query as follows. The word is matched sequentially with every rule in the rule base. A match occurs when the word and LHS pattern of a rule match. For each rule match, the RHS strings are substituted for the matched part of the word and new words generated. For example, the rule in Equation (1) would give rise to 4 words. Usually there will be several rule matches and hence several new words will be generated. These words are added to a candidate word list in the same order they are generated. After all the rules have been exhausted, the candidate word list is examined to pick the top N words. This list will be used to form the refinement for one word. Now, repeating the procedure for all the words in the query, the target refined query is generated as follows. Consider the query **q**,

$$\mathbf{q} = qw_1 \text{ AND } qw_2 \text{ AND } \ldots. qw_M \quad (2)$$

where qw_i,i=1,…,M, are the words in the query (assuming the query has M words) and 'AND' is the boolean AND operator. This is the most common form of search query (known as 'Match All Words') used in practice. We do not imply that our algorithm is limited to such queries only. We consider this first, mainly for convenience sake. Later on, we show how the method can be adapted for other forms such as 'Match Any Word'.

For the query word qw_i,i=1≤ i ≤ M, suppose the matching with the rulebase generates the candidate word list C_i. After picking the top N words,

$$Ci = \{rw_{i1}, rw_{i2,}\ldots., rw_{iNi}\}, \qquad N_i \le N \quad (3)$$

where rw_{ij} are the refined words.

Then the target refined query **q'** is,

$$\mathbf{q}' = \text{AND}^{M}_{i=1} \left(\text{OR}^{Ni}_{j=1} rw_{ij}\right) \quad (4)$$

where OR denotes the the boolean OR operator.

The algorithm is formally presented in **Figure 2**.

1. Let **R** be the ordered list of rules in the rulebase. Let N be a positive integer.
2. Let the search query **q** be given by Equation (2).
3. For i=1 to M do
   Begin
   - Let C_i be an empty list.
   - For each rule *r* in **R** do
     Begin
     - Let *r* be of the form *p* => **S**, where *p* is the pattern and **S** is the list of substitution strings.
     - If qw_i matches *p*, replace the matched parts with each string in **S**
     - Append the resulting words to C_i.

     End
   - Let N_i be the number of words in C_i. If $N_i > N$, retain only the top N words and delete others.

   End
4. The refined query is **q**' given by Equation (4).

Figure 2. Query Refining Algorithm

Extensions:
(1) If the input query is in 'Match Any Word' form, then the refined query can be generated following the same procedure as above except that **q'** will now be

$$\mathbf{q}' = \mathrm{OR}^{M}_{i=1} \left(\mathrm{OR}^{Ni}_{j=1} rw_{ij}\right) \quad (5)$$

(2) Though we mention as 'word' throughout, the algorithm can also be used if the input query contains phrases. In such a situation, the words in the phrase will not be split (as in Equation (2)) but will be treated as a single "query word".

2.3 Generic Search Engine (GSE)

This is any search engine that can provide basic search and retrieval operations for the documents under consideration. If internet search is of interest, then GSE can represent Google, Yahoo or any standard web search engine. It can also stand for others, e.g. Online Tamil Lexicon (OTL) Search [5], if the interest is to find the English equivalent of a tamil word. The only requirement is that the search engine support basic boolean queries, in particular the AND and OR operators with Nesting allowed. This is a feature commonly found in modern day search engines and virtually all internet search engines support it [6].

Now it is easy to see the functioning of the search system in Figure 1. The user inputs a search query. The QR component parses the query and uses the Rule Base to generate a refined query following the algorithm in Figure 2. This new query is passed to the GSE and the returned search results presented to the user.

3. Evaluation Studies

The evaluation studies of our search system has been done with a prototype [7] built with Google as the Generic Search Engine. We have chosen Google as it has one of the largest collection of web pages. It is also one of the most widely used.

For the search task, we selected 7 queries each of which in the authors' opinion had significant scope for transliteration ambiguity. Ten volunteers (from different parts of the world) were individually shown these queries in tamil script and asked to transliterate them in the way they most prefer. With their input, we selected 4 queries that had been spelt most differently. The queries and their variations are presented in different tables below.

For the rulebase, we coded about 25 rules to capture the most common variations and ordered them using our limited tamil knowledge. Without a linguist's involvement, this is admittedly an approximate one. We only intend it to be a starting point towards building a more accurate and complete rulebase.

The query refiner and the interface to Google were coded in PERL on a SUN Solaris platform. The ceiling parameter 'N' (see Figure 2) was set to 10 when the query contained only one word. When the query had M words, the parameter was set to the largest integer less than or equal to 10/M. This setting is needed so as not to exceed Google's limit of 10 words/query.

To measure the performance, ideally we would have liked to study the relevance of each search result and use the standard precision and recall measures [8]. Since Google search results are usually in 1000's even for tamil queries alone, it is very tedious to assign the relevance judgements. Hence, as a compromise, we use an approximate form of recall as our performance measure. It is defined as

$$\mathrm{Re}call' = \frac{\mathrm{Re}l\,\mathrm{Re}t}{\mathrm{Re}l} \tag{6}$$

where *RelRet* is the number of Google results returned for the given query, say q, and *Rel* is computed as follows:

Let $\mathbf{q}_{\text{user}}$ be the list of different variations of the query q according to the 10 users. Let $\mathbf{q}_{\text{spell}}$ be the list of queries obtained by Google's spelling correction on $\mathbf{q}_{\text{user}}$. Let $\mathbf{q}_{\text{trans}}$ be the list of the top 10 candidate refinements returned by our QR algorithm for each query in $\mathbf{q}_{\text{user}}$. Let $RelRet_{max}$ be the maximum *RelRet* over all queries in q_{user}, q_{spell} and q_{trans} combined. Then,

$$\mathrm{Rel} = \mathrm{RelRet}_{\max} \tag{7}$$

Rel is defined to compute an estimate on the total number of Google documents on query q. It is most likely an underestimate because of Google's size. However, our interest is mainly in a normalized measure to quantify the relative differences in recall (as defined in Equation (6)) and so this will serve the purpose.

Below we present the detailed results for 2 search tasks. (For the other 2 search tasks, we are only able to provide brief average results, due to space constraints.) Each table has five

columns. The first column lists the transliteration variations for the target tamil query. The second column contains the *RelRet* and *Recall'* results for the raw Google search. Columns 3 and 4 list the corresponding results using Google's spelling correction and Our Method respectively. If a column 3 entry is empty, it means that Google did not return any spelling correction. In column 5, we mention how much recall improvement our algorithm has been able to achieve compared to Google's best performance either with raw search or spelling correction.

Task I: Target Query = இளையராஜா

Query	Raw Google		Google's Spelling correction		Our Method		Recall Improvement
	RelRet	Recall	RelRet	Recall	RelRet	Recall	
Ilaiyaraaja	2100	0.231	3790	0.416	7540	0.830	0.414
Ilaiyaraajaa	162	0.178	3790	0.416	5300	0.583	0.167
Ilaiyarajah	27	0.003	-	-	5000	0.550	0.827
Ilayaraja	3790	0.416	3790	0.416	4950	0.544	0.128
Ilaiyaraja	1390	0.152	3790	0.416	5230	0.575	0.159
Ilayaraaja	763	0.084	3790	0.416	9090	1.000	0.584

Table I: Results for the search task where the target query is "இளையராஜா"

Task II: Target Query = கண்ணதாசன் கவிதைகள்

Query	Raw Google		Google's Spelling Correction		Our Method		Recall Improvement
	RelRet	Recall	RelRet	Recall	RelRet	Recall	
Kannadhasan Kavithaigal	10	0.061	32	0.195	144	0.878	0.683
KaNNadhaasan kavidhaigaL	24	0.146	56	0.341	129	0.786	0.445
KaNNadhasan kavithaikaL	9	0.054	32	0.195	23	0.140	-0.055
KannadAsan kavidaigaL	0	0.000	32	0.195	144	0.878	0.683
KaNNadaasan kavidhaigaL	17	0.103	32	0.195	154	0.939	0.744
Kannadasan kavidhaigal	78	0.475	32	0.195	144	0.878	0.403
KaNNadhaasan kavithaigaL	6	0.036	32	0.195	164	1.000	0.805

Table II: Results for the search task where the target query is கண்ணதாசன் கவிதைகள்

It is observed that our method achieved significant improvements in recall on all the four tasks. In particular on Tasks II & IV, tremendous improvement is noted (see Table III). In these tasks, the query had relatively more transliteration ambiguity. Having been customized for tamil, our algorithm was able to refine the query more accurately and outperform Google. On average, our algorithm is able to achieve about 250% improvement in recall. We also observed that our algorithm retrieved the maximum number of documents (i.e. $RelRet_{max}$) in all the tasks.

Summary of Results:

User Query	No. of Variations (Out of 10)	Avg Recall (Best of Google)	Avg Recall (Our Method)	% Improvement
இளையராஜா	6	0.347	0.680	96.0
கண்ணதாசன் கவிதைகள்	7	0.256	0.785	206.6
அபிராமி அந்தாதி	6	0.275	0.800	190.9
அபிராமி அந்தாதி	7	0.092	0.557	505.4
			Avg Total	249.7

Table III: Summary of Results over the four search tasks

4. Discussion and Conclusion

This paper has proposed a new method for robustly searching tamil transliterated collections. This work was borne out by the authors' personal frustration with internet search engines while searching for tamil documents. Though Google spelling correction algorithm can help improve the searching to some extent, it is far from being ideal simply because it does not make use of any tamil language knowledge. Efforts to incorporate language specifics in the form of tamil dictionaries can demand quite an amount of human resources and hence unattractive. We have proposed a query refinement approach in which the language specifics are captured through a rulebase. This approach is attractive because of a relatively simpler rule base. We were able to build to a rulebase without any linguist's involvement and obtained about 250% improvement in recall over Google's best performance. We believe these results are only preliminary as the number of queries considered was rather small. Also it is not clear to what extent query refinement affects the precision. Currently we are doing more detailed experiments to address these issues.

The design of Query Refiner and Rulebase follows the AI approach whereby the rulebase can be enhanced or amended independent of the Query Refiner. This implies that an informed user can add or modify the rules to suit specific needs. For example, suppose a website, say Online Tamil Lexicon, wants to implement robust Tamil search on its dictionary. Since the transliteration scheme used is somewhat different from common usage, our current search system may not give best results. The search performance will be much better if the system can be customized. This is readily possible because the rulebase can be independently modified. The only knowledge needed is the format of the rules which we believe should be relatively easy to follow.

An important consequence is: because the tamil knowledge is embedded only in the rulebase, it is possible to plug-in a suitable rulebase for another language, say Hindi, and create a robust search algorithm for Hindi transliterated documents. Thus, our method can be extended (theoretically) for other native languages too. We are currently exploring this possibility for searching indian languages other than tamil.

5. References

1. K. Rajaraman and Kok F. Lai, "*i*Agent: A System for Managing Networked Tamil and Multilingual Information Resources", *TamilNet'97 International Symposium*, May 1997.

2. "Project Madurai", URL: http://www.tamil.net/projectmadurai/.
3. "Working Group 5: Transliteration schemes for Tamil", URL: http://www.infitt.org/about/wgrep2k1.php.
4. "Google: Technical Highlights", URL: http://www.google.com/press/highlights.html.
5. "Online Tamil Lexicon", URL: http://www.uni-koeln.de/phil-fak/indologie/tamil/otl_search.html
6. "Search Engine Facts", URL: http://searchenginewatch.com/facts/ataglance.html.
7. "Google: Tamil Search", URL: http://textmining.lit.org.sg/people/kanagasa/tamil/.
8. G. Salton and McGill, "Introduction to Modern Information Retrieval", McGraw-Hill, 1983.