TR-IT-0317

Approximate Pattern-Matching Agrep: a UNIX Command

Yves Lepage

December 1999

Abstract

We present a program for approximate pattern-matching that has been shown to be asymptotically faster on average than a similar program by Wu and Manber, agrep. We sketch the method used and show comparative results. This program is accessible under Unix and can be used as C functions. An option not found in agrep allows this program to handle two-byte Japanese characters.

Keywords

Approximate pattern-matching, edit distances, Japanese characters.

Contents

Introduction **1**

1 Approximate pattern-matching **3**

1.1 Edit distances 3

1.2 Exact pattern-matching 3

1.3 Approximate pattern-matching 3

1.4 `Agrep`: a sketch of the technique 4

2 Comparison of `Agrep` with `agrep` **7**

2.1 Protocol 7

2.2 Results 7

2.3 Analysis 10

3 `Agrep` as a Unix command line **11**

3.1 Functionality 12

3.2 Options 12

3.3 Examples 13

3.4 Use 14

4 `Agrep` as C functions **15**

Conclusion **19**

List of Figures

2.1 Average runtimes by lengths and thresholds. 8
2.2 Average runtimes by threshold / length of the pattern ratio. . 9

Introduction

Approximate matching is similar to pattern-matching, *i.e.* the retrieval of a pattern from a text, but is different in that "errors" are tolerated. Formally, these errors are seen as edit operations performed on the pattern: character insertions, deletions or substitutions. The minimum number of edit operations or errors necessary to transform a string into another string is called the edit distance between the strings. To perform approximate matching, usually, the maximum number of such errors is given as a threshold.

Thus, a program of approximate pattern-matching searches for those lines in the text (of length n) that contain a substring at a distance less than or equal to a given threshold k from a given pattern (of length m).

For example, when looking for the pattern `analogy` with a threshold of 2, from the following text:

```
analogous
explanation
neuroanatomy
```

only the first and third lines are output:

`analogous`	dist(`analog`, `analogy`) = 1	(insert `y`)
`neuroanatomy`	dist(`anatomy`, `analogy`) = 2	(`t` $\leftrightarrow$ `l` and `m` $\leftrightarrow$ `g`)

Recently, Wu and Manber [Wu & Manber 92] proposed a practical implementation of the Baeza-Yates and Gonnet method [Baeza-Yates and Gonnet 92], which exhibits a behaviour of $O(nk\lceil \frac{m}{w} \rceil)$, where w is in fact some constant. *agrep* is considered the fastest practical algorithm for approximate pattern-matching. `agrep` has been incorporated in a tool for searching local Web sites (`webglimpse`, see http://glimpse.cs.arizona.edu/webglimpse/index.html).

The present technical report describes a Unix command called `Agrep` (with an uppercase `A`) which appears to be be a good competitor of `agrep`.

Chapter 1

Approximate pattern-matching

1.1 Edit distances

Edit distances have been first proposed by the Russian computer scientist Levenshtein [Levenshtein 65]. Algorithms for distance computation have been first proposed by Wagner and Fischer [Wagner & Fischer 74]. They rely on the computation of a matrix, by dynamic programming.

1.2 Exact pattern-matching

Pattern-matching has been thoroughly studied. Some very good documentation exist on the Web. For instance, see `http://www.dir.univ-rouen.fr/~charras/string/` or `http://www-lsi.upc.es/~rbaeza/handbook/text_a.html`.

1.3 Approximate pattern-matching

The algorithms proposed for approximate pattern-matching are usually based on the distance-matrix computation. For example: [Landau & Vishkin 88] and [Baeza-Yates and Gonnet 92]. All these methods basically try to minimize the number of cells computed in the distance-matrix.

On the contrary to these algorithms, our algorithm bases on representing factored string subsequences [Lepage 97]. This method is the object of a pending Japanese patent (Application no. 9-232902, day of application: August 28th, 1997). An international patent is also pending.

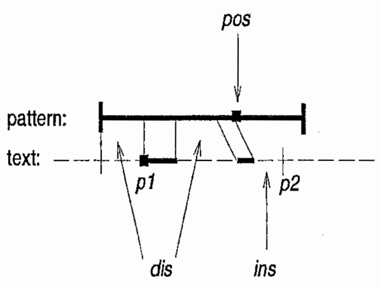
The implementation of this method as a UNIX command is called `Agrep` (with an uppercase `A`), to mimic the name of `agrep` (with a lowercase `a`) by Wu and Manber.

1.4 Agrep: a sketch of the technique

`Agrep` bases on representing factored string subsequences with only three integers: past distance, last position in the pattern and potential insertion length.

Our method tries to detect the longest common subsequence between the pattern and the text, rather than to compute the edit distance (the two problems are“ dual”). Any candidate longest subsequence may be characterised by just three integers: the last position matched in the pattern (*pos*), the past distance up to that point (sum of the maximums of the lengths of the corresponding non-matching parts of the pattern and the text, *dis*), and the length of a gap ahead (*ins*).

Hence, the core data of our algorithm is a table containing arrays of three integers, each array representing a possible candidate match being built.



pos	*dis*	*ins*
4	1	0
3	0	1
...	...	...

At a given position in the text, if the next character read from the text belongs to the pattern, a new candidate may be computed over all compatible candidates, by taking the minimal past distance possible.

In any case, the next character may be considered as a gap added ahead. Hence, for all candidates, this gap, represented by *ins*, is incremented.

From this sketch of the algorithm, one could conclude that the set of candidates will always grow. Fortunately, four constraints may be applied so as to reduce it, so that it always remain reasonable during execution. In our experiments (see Chapter 2) it practically remains at 0.5, which is that in average, there is only one candidate every second position in the text searched, and no candidate elsewhere.

The technique also crucially relies on the order in which the triples are examined at each step, *i.e.*, the sort used for the arrays is the key to the efficiency of `Agrep`. An enumeration of the candidates in decreasing order of *pos* and *dis* values has proved to accelerate the execution of the algorithm. For each character from the searched text, scanning the candidate array can start exactly on the right candidate until the end of the set, thus considering only the candidates with a relevant position, and thus avoiding checking the constraint for all candidates in the set.

Chapter 2

Comparison of `Agrep` with `agrep`

2.1 Protocol

We searched for every word of an English dictionary, in the dictionary itself, with all possible thresholds, from 1 to the length of the word minus 1. The size of the dictionary was 200 Kb. As the number of words was 25 143 and the average length of a word was 7.22, this represented approximately 181 500 runs of the search.

2.2 Results

By comparing the outputs of `agrep` and of our algorithm, we verified that both algorithms yield exactly the same results in all the cases.

However, we found that our algorithm is more robust than `agrep`, and could not perform extensive comparisons with patterns of indefinite length with large thresholds, as `agrep` seems to have inherent limitations: `agrep` is shipped with a limited threshold of 8. We extended this limit to 64 by changing some constants in the source code, so that the experiments could be run. Even by doing that, we obtained wrong results for thresholds larger than 12 with different edit operation weights. These errors did not appear for equal weights.

Also, we found a bug in `agrep` which is that some solutions overriding the end of the input buffer are not output.

We drew the execution times by lengths of patterns, by thresholds, and by ratios of threshold over length of pattern in Figures 2.1 and 2.2.

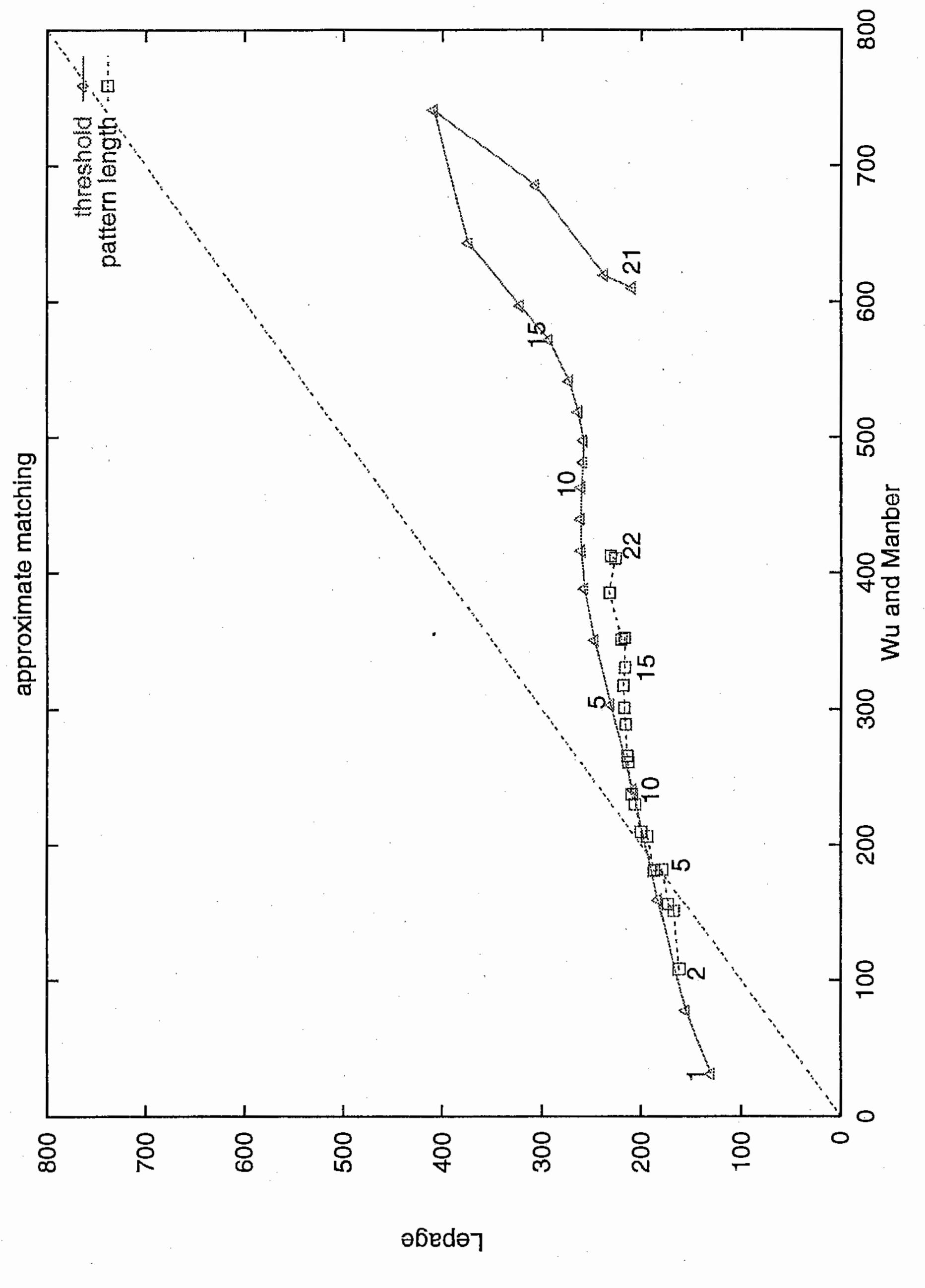


Figure 2.1: Average runtimes by lengths and thresholds.

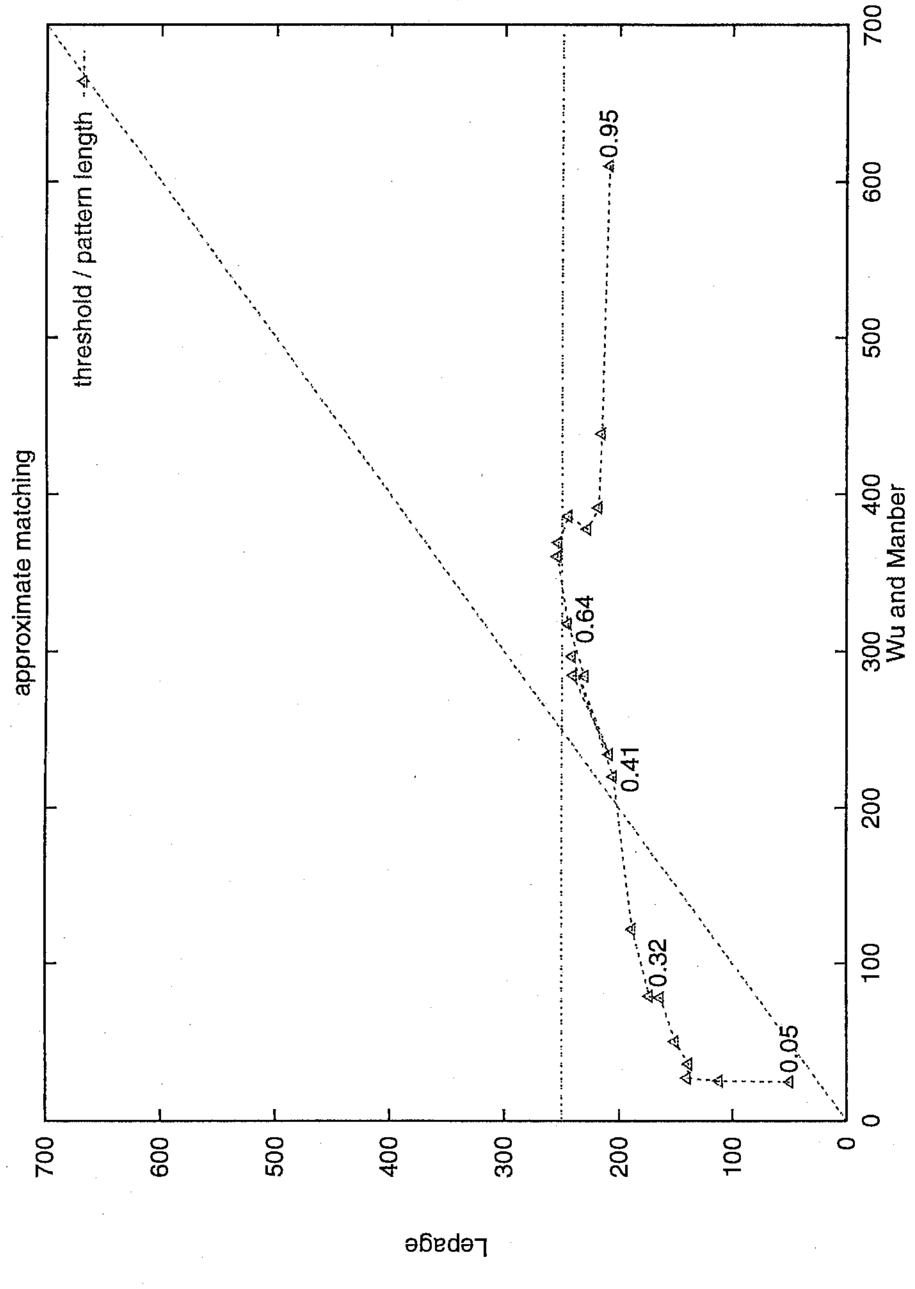


Figure 2.2: Average runtimes by threshold / length of the pattern ratio.

2.3 Analysis

In average, for measures by pattern lengths, although the results for `agrep` extend from 100 to 400 ms, those of `Agrep` are framed between 100 and 200 ms.

For measures by thresholds, the runtimes of `agrep` roughly increase by 50 ms when the threshold increases by 1, and our algorithm is shown to be approximately twice faster in average as `agrep`.

Relatively to the threshold / length of the pattern ratio, the results for `agrep` extend roughly linearly from 20 ms up to 600 ms. On the contrary, for our algorithm, the average execution times are approximately constant: they lay around 200 ms, and always remain under 250 ms . Our algorithm is significantly faster when the ratio threshold / length of the pattern becomes greater than 0.4.

Chapter 3

Agrep as a Unix command line

We present the command line options of Agrep. Hereafter is the result of the command `Agrep -h` which displays on-line help.

Also, in the next sections, one will find the description of Agrep as found in its man page.

```
use:      Agrep [-n][-w][-J][-B]
                [-D<d>][-I<i>][-S<s>]
                [-<n>] <string> [-in=<filename>]
default: Agrep <string> means Agrep -0 -D1 -I1 -S1 <string>
job:      approximate search of <string> with distance threshold <n>
          with costs <d>, <i> and <s> for deletion, insertion and
          substitution respectively.

options:
-D<d>: set deletion cost to <d>
-I<i>: set insertion cost to <i>
-S<s>: set substitution cost to <s>
       <d> and <s> must be positive integers
       <i> must be a non-negative integer
-n: output line prefixed by line number
-w: forces to match words, e.g. 'win' will not match 'wind'
-J: use two-byte characters (useful for Japanese characters)
-B: output closest match
-in=<filename>: search in file <filename>
-T: display execution time in ms (not useful to the user)
```

3.1 Functionality

`Agrep` searches the standard input for any line containing the pattern with at most *n* errors (distance threshold) according to edit distance. Each line found is copied to the standard output. Approximate matching allows finding lines that contain the pattern with several errors, namely substitutions, insertions, or deletions. For example, *edible* matches *eatable* with 3 errors (one insertion and two substitutions). Running `Agrep -3 edible < foo` outputs all lines in foo containing any string with at most 3 errors from edible.

3.2 Options

`-`<*n*> *n* is a non-negative integer (no limitation) specifying the maximum number of errors permitted in finding the approximate matches. Each insertion, deletion, or substitution normally counts as one error.

`-D`<*d*> Set the cost of a deletion to <*d*> (<*d*> is a positive integer).

`-I`<*i*> Set the cost of an insertion to <*i*> (<*i*> is a positive integer).

`-S`<*s*> Set the cost of a substitution to <*s*> (<*s*> is a positive integer).

`-J` The input file is read as a 2-byte character file (useful for Japanese).

`-B` Best match mode. Search is performed subsequently for all error numbers from 0 up to the length of the pattern minus 1, until some matches are found. The best match mode is not supported for standard input, i.e. pipeline input. To designate the file where to search, use option `-in=`.

`-n` Each line that is printed is prefixed by its record number in the file.

`-w` Search for the pattern as a word (*i.e.*, composed only of alphanumeric characters and surrounded by non-alphanumeric characters. Non-alphanumeric characters *will* surround the match, and *are not* part of it; they do not count as errors. For example, `Agrep -w -1 car` will match *cars*, but not *carpenters*. As another counter-example, `Agrep -w -1 youxhave` will not match *you have*.

`-in=<`*filename*`>` File <*filename*> is the file where to search. This option is mandatory when option `-B` is in use.

3.3 Examples

`Agrep -5 abcdefghij < /usr/dict/words` outputs all words containing at least 5 of the first 10 letters of the alphabet in order:

```
Philadelphia
```

`Agrep -2 -S3 enamel < /usr/dict/words` outputs all words containing the word *enamel* with at most two extra or deleted characters:

```
adrenal
adrenaline
ameliorate
...
tournament
venal
Vietnamese
```

`Agrep -B -w approixmate -in=/usr/dict/words` performs spell checking of the word *approixmate* using the Unix dictionary. Outputs:

```
approximate
```

`Agrep -J -B 二面様 -in=nec.text` performs a best match search from the texts of the ATR-NEC treebank, using two-byte characters. Outputs:

```
お二人様別々の席になられてもよければ空いております。
申し訳ありませんがコースは二名様からとなっています。
二名様ですね。
```

3.4 Use

The program is accessible at ATR as the command `Agrep`

- under the directory `/home/lepage/bin` for Solaris[1];
- under the directory `/home/lepage/bin/alpha` for Alpha machines.

These directories are normally accessible to all users. By the way, anyone is welcome to use any other program from these directories, which would be of interest for him. Descriptions of the commands are found in the file `INFOS`.

[1]We do not support SunOS, as this sytem is supposed to be not Y2K compliant, and Sun Microsystems is said not to support SunOS after 1999.

Chapter 4

Agrep as C functions

The functionalities of `Agrep` are proposed as four C functions. For their use, please mail to `lepage@itl.atr.co.jp`. They have been integrated under the general library we have been building at ATR for years [Lepage 92b].

We list hereafter the C header file for the corresponding functions.

```
/* Copyright (c) 1999, Yves Lepage */
/*
 * Approximate matching functions
 */

   #include <stddef.h>
   #include <stdio.h>

/*
 * Function approxmatch:
 * All arguments version:
 * input (1st arg and 3rd to last args):
 *    input file                    (1st arg),
 *    searched pattern of type x * (3rd arg),
 *    threshold,
 *    deletion     weight,
 *    insertion    weight,
 *    substitution weight,
 *    display line numbers in output (0/1),
 *    search inside words only (0/1),
 *    size of x, i.e., sizeof(x),
 *        1: 1-byte char,
 *        2: 2-byte char, useful for Japanese character,
 *        4: 4-byte char, pointers (void *) on sun machine,
 *        8: 8-byte char, pointers (void *) on alpha machine,
 * output:
 *     output file (2nd arg)
 *        contains those lines of the input file which
 *        contain at least one substring at a distance
 *                of atmost the threshold
 *
 * At the end of the function approxmatch,
 *     the input and output files are rewinded
 *     so that they are ready for further use
 *
 * Function closestmatch:
 *     same arguments as approxmatch, but
 *     threshold does not exist
 *     (approxmatch is applied repeatedly
 *      from threshold=0 to length(pattern)-1
 *      until something is found)
 * output:
 *     threshold for which something was found
 *     -1 if nothing was found
 */
```

void approxmatch(**FILE** *istream, **FILE** *ostream, **unsigned char** *pattern, **int** threshold, **int** dd, **int** ii, **int** ss, **int** linenbron, **int** wordonlyon, **size_t** nbyte) ;

int closestmatch(**FILE** *istream, **FILE** *ostream, **unsigned char** *pattern, **int** dd, **int** ii, **int** ss, **int** linenbron, **int** wordonlyon, **size_t** nbyte) ;

```
/*
 * Simplified versions:
 *     the pattern is of type unsigned char *,
 * hence:                                       (in approxmatch:)
 *     sizeof(x) = 1                            (10th arg = 1)
 * and
 *     deletion     weight = 1                  ( 5th arg = 1),
*    insertion    weight = 1                 ( 6th arg = 1),
*    substitution weight = 1                 ( 7th arg = 1),
*    do not display line numbers in output ( 8th arg = 0),
*    search inclusive of word frontiers    ( 9th arg = 0),
*
* input:
*    input file,
*    pattern of type unsigned char *,
*    threshold for approxmatch1
* output:
*    output file (2nd arg)
*    threshold for which something was found for closestmatch1
*/
```

void approxmatch1(**FILE** *istream, **FILE** *ostream, **unsigned char** *pattern, **int** threshold) ;

int closestmatch1(**FILE** *istream, **FILE** *ostream, **unsigned char** *pattern) ;

Conclusion

Agrep has been shown to be faster than agrep in average on a one fifth million run experiment of retrieving words with all possible thresholds from a 200 Kb size file. It is definitely faster for larger thresholds ($k > 3$), and for threshold over pattern length ratios greater than 0.4. The algorithm used in Agrep is also more robust, and does not imply any limitation in the number of errors on the contrary to agrep.

Any bug reports or comments will be appreciated. Please mail them to lepage@itl.atr.co.jp

Bibliography

[Baeza-Yates and Gonnet 92] Ricardo Baeza-Yates and Gaston H. Gonnet
A New Approach to Text Searching
Communications of the ACM, Vol. 35, No. 10, October 1992, pp. 74-82.

[Landau & Vishkin 88] Gad M. Landau and Uzi Vishkin
Fast String Matching with *k* Differences
Journal of Computer and System Sciences, Vol. 37, 1988, pp. 63-78.

[Levenshtein 65] V.I. Levenshtein
Binary codes capable of correcting deletions, insertions and reversals
Dokl. Akad. Nauk SSSR, vol. 163, No. 4, August 1965, pp. 845-848. English translation in *Soviet Physics-doklady* vol. 10, No. 8, February 1966, pp. 707-710.

[Lepage 92b] Yves Lepage
Easier C programming
Some useful objects
ATR report TR-I-0294, Kyoto, November 1992.

[Lepage 97] Yves Lepage
String Approximate Pattern-Matching
55th Meeting of the Information Processing Society of Japan, Fukuoka, August 1997, vol. 3, pp. 139-140.

[Wu & Manber 92] Sun Wu & Udi Manber
Fast Text Searching Allowing Errors
Communications of the ACM, Vol. 35, No. 10, October 1992, pp. 83-91.

[Wagner & Fischer 74] Robert A. Wagner and Michael J. Fischer
The String-to-String Correction Problem
Journal for the Association of Computing Machinery, Vol. 21, No. 1, January 1974, pp. 168-173.