MAKING THE WORLD'S LARGEST WORD SEARCH PUZZLE

DAVE MILLER* & KATHLEEN MATTSON[1]

May 27, 2020

CONTENTS

ABSTRACT

This article describes how we constructed the world's largest word search puzzle containing the names of 163,563 cities and places around the world. We describe how we obtained the names and converted them into the ASCII character set. We describe our algorithm and C++ program that generated the puzzle. Finally we describe how we converted the computed 948 x 948 character array into graphic files and printouts.

* *dave@millermattson.com*
[1] *kathleen@millermattson.com*

1 PREPROCESSING THE PLACE NAMES

Our list of place names starts with a database of names provided by GeoNames[2]. One of their geographical databases, "cities500.txt," is a list of the names of cities and places around the world with a population of 500 or greater, licensed under Creative Commons Attribution 4.0 License[3].

Each line in their database contains several fields separated by tabs. One of the fields is the city or place name in ASCII, but we extracted the field containing the name in the original character set so that we could manage our own conversion to ASCII.

We did all the pre-processing on a Linux system where the necessary text processing commands are easily available.

To extract the second field from each line in the database, we used the Linux filter:

```
cut -f2
```

Out of 165,126 lines, 8,758 of them contained one or more hyphens, for example:

```
al-Kum
Acheux-en-Vimeu
Castelbello-Ciardes - Kastelbell-Tschars
```

After sampling these cases, we seemed to identify a consistent pattern: hyphens with no space before or after were part of a hyphenated name, and a hyphen with a space before or it separated two different names. We used the filter:

```
sed -e 's/ -/\n/g;s/\- /\n/g'
```

For example, this caused the name "Castelbello-Ciardes - Kastelbell-Tschars" to become two lines:

```
Castelbello-Ciardes
Kastelbell-Tschars
```

Over one thousand lines contained a parenthetical part, for example:

```
Vicente Guerrero (San Javier)
```

The parts in parentheses seemed to be alternative names, so we used the following filter:

```
sed 's/(/\n/'
```

This separated the parenthetical part into lines of their own. The last example became:

```
Vicente Guerrero
San Javier)
```

A few lines contained text in square brackets, for example:

[2] https://geonames.org/
[3] https://creativecommons.org/licenses/by/4.0/

```
Ninguno [Centro de Readaptación Social de Atlacholoaya]
San Sebastián [Fraccionamiento]
```

Some of the bracketed text seemed to be alternative names, but most of them seemed to be explanatory text. In the example above, "Fraccionamiento" means "division." There were only 56 bracketed instances, and instead of examining each one, we simply deleted all bracketed text with the filter:

```
sed 's/\[.*\]//g'
```

There were 372 lines containing a forward slash that separated two variations of a name, for example:

```
Zürich (Kreis 11) / Affoltern
Wetzikon / Ober-Wetzikon
Biel/Bienne
```

We ran the following filter to separate the parts after the slash onto lines of their own:

```
sed 's/\//\n/g'
```

A few lines in the database contained quotation marks which seemed to show alternative names, for example:

```
Poselok Turisticheskogo pansionata "Klyazminskoe vodohranilische"
```

We ran a filter to move the text between pairs of quotes onto lines of their own without changing any lines containing a single double quote embedded inside a word:

```
sed 's/\(^.*\)"\(.*\)"/\1\n\2/'
```

Numerals appeared in 362 of the lines, for example:

```
Zürich (Kreis 7) / Fluntern
Sur 2da. Sección
Poblado C-21 Licenciado Benito Juárez García
Colonia 24 de Febrero
```

We could have constructed a word search puzzle that included numerals, but there's a problem with that. Since the puzzle will be constructed with upper-case ASCII characters and printed in a very small font size, it would be difficult to see the difference between numeral "1" and upper case "I", or between numeral "0" and upper case "O". Removing only the numerals or converting them into text would result in erroneous place names. For simplicity, we used the following filter to simply remove all lines containing any numerals:

```
grep -v [0-9]
```

After all the previous filtering, some lines are explanatory text. For example, a line might have started out as:

```
Geroskípou (quarter)
```

The parenthetical part in this case is explanatory, not an alternative name, and at this point has now become two lines:

```
Geroskípou
quarter
```

We remove lines that are explanatory fragments with the following filter:

```
NOTNAMES="^dorf$|^nord$|^ost$|^west$|^süd$|^city$\
|^dorfkern$|^borders$|^fraccionamiento$\
|^historical$|^village$|^quarter$|^nördl. Teil$|^Barrio$\
|^club$|^granjas familiares$|^unidad habitacional$"
egrep -vi "$NOTNAMES"
```

At this point we have some duplicates. For example, there are multiple places named "La Magdalena" and quite a few called "San Francisco." To remove duplicates, we applied the filter:

```
sort -d | uniq
```

The GeoNames database has a field in each line containing an ASCII equivalent of the place name, but we could not find documentation explaining the source of those names. We latinized the names ourselves using the command:

```
iconv -f utf-8 -t ascii//TRANSLIT
```

We converted all the lines to upper case with:

```
tr '[:lower:]' '[:upper:]
```

Some punctuation remained, such as hyphens, right parentheses, and single quote marks. We removed all non-alpha characters and all spaces with:

```
sed 's/[^A-Z]//g'
```

We again removed duplicates, because the latinization process might have converted different foreign accents or diacritical marks into the same ASCII character:

```
sort -d | uniq
```

Our puzzle generating algorithm works more efficiently if it processes long words first, so we use this command to sort the names by length:

```
awk '{ print length, $0 }' | sort -nrs | cut -d" " -f2-
```

Finally we have a list of 163,563 all-upper-case, ASCII-only names without spaces, sorted by size, ready for our algorithm to arrange into puzzle form.

Here are some examples of the results of all the conversions applied:

Poblado Alfredo V. Bonfil	POBLADOALFREDOVBONFIL
União da Vitória	UNIAODAVITORIA
Untermaßfeld	UNTERMASSFELD
Untsukul'	UNTSUKUL
Pănătău	PANATAU
Poá	POA

After all the preprocessing described, the average length of a place name that will become input to the puzzle generator is 9.55 characters. The longest place name began as

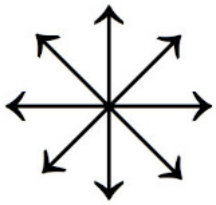
```
Unidad Familiar Confederación de Trabajadores Campesinos
```

and after the preprocessing became

```
UNIDADFAMILIARCONFEDERACIONDETRABAJADORESCAMPESINOS
```

2 THE ALGORITHM

There is a small body of literature on the subject of how to automatically construct a crossword puzzle or word search puzzle. A crossword puzzle is a special case where words may flow in only two directions. In a general word-search puzzle, words may flow in eight directions:



Many of the published algorithms start by placing the first word at a random location and orientation, then adding words that intersect the existing word(s) optimally, where optimally is defined in various ways. An undo stack may be maintained so that when a word fails to fit anywhere, the stack can be unwound and words refitted differently in an attempt to allow more words to fit.

Our algorithm is simple. We start with a randomly placed word in a fixed grid size, then give up when we encounter the first word that doesn't fit. We process the words in decreasing order of length, i.e., the first word is the longest in the set. In a puzzle as large as the ones we're constructing, by the time we get to the short words, many of them already exist by accidental placement of prior words, or else they easily fit into a multitude of cracks that remain.

When we fit a new word into the puzzle, we score how it would fit at all eight orientations at every grid point in the puzzle and choose the location and orientation with the highest score, choosing randomly in case of a tie. That means in a grid of N x N characters, we evaluate all possible $8N^2$ orientations and locations that a word might fit.

Our scoring formula uses a letter frequency table for the specific set of words in our puzzle. We obtain a letter frequency table for our set of input words with this command:

```
sed 's/\(.\)/\1\n/g' | sort | uniq -c | sort -nr
```

We divide the counts by the total number of characters in the corpus. For our list of 163,563 latinized city and place names, we arrived at this table that we included in our C++ program:

```
    const float letterFreq[] = {
        0.132392179631196, // A
        0.0237766474397,
        0.034265231165356,
        0.02983503515127,
        0.092987416223858,
        0.008277903292986,
        0.030076108495627,
        0.029719547135519,
        0.071271882770198,
        0.007303512602709,
        0.018694543664567,
        0.061379040502846,
        0.027342892122834,
        0.079099824559189,
        0.073086243673402,
        0.019114213230004,
        0.002462482171932,
        0.065484860341542,
        0.055515657145742,
        0.046633808327759,
        0.03990016281917,
        0.015404712920774,
        0.008503830668063,
        0.002790013757589,
        0.014317358542958,
        0.010364891643212 // Z
    };
```

Given a location and orientation, if w is the set of letters in the word and g is the corresponding set of letters in the existing grid at the same location and orientation, then the score is negative if any intersecting letters $w_i \neq g_i$ and that location and orientation is discarded from further consideration. The score is zero if all $g_i = empty$. Otherwise there are intersecting letters that are the same and the score is the sum of the inverse letter frequencies f of the intersecting letters using the frequency table described earlier:

$$score = \sum_{i|w_i=g_i} 1 - f_{w_i}$$

If no non-negative score is found, it means there is no place to fit the word and our program aborts.

Here is a simple example. Suppose we start with a set of exactly three words:

```
DAVE
DREW
DAWN
```

The first word gets placed at a random location and orientation:

```
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . D A V E . . .
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
```

The next word, DREW, could intersect DAVE at several orientations at either the D or the E. Any path that intersects an existing letter scores higher than any path that fills only empty locations. Of the possible paths that intersect D or E, the paths that intersect D are preferred to any that intersect E because D is a less common letter than E. All the possible orientations that intersect D have the same score, so the algorithm picks one at random:

```
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . D A V E . . .
. . . R . . . . .
. . . . E . . . .
. . . . . W . . .
```

The third word, DAWN, could fit by sharing the D, A, or W that already exist, but since W occurs less frequently than D or A according to our frequency table, the paths that intersect W are scored the highest and one is chosen at random:

```
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . D A V E . . .
. . . R . . . . .
. . . . E . . . .
. . . D A W N . .
```

If a word intersects with more than one existing letter, the score is the sum of all the inverse frequencies of the intersecting letters.

Our brute-force algorithm required about seven hours of CPU time to fit all 163,563 names into a puzzle. There are 898,704 locations in a 948 x 948 grid, and a total of 1,562,731 letters in our set of names, so there was quite of bit of sharing of letters.

The final puzzle had a density of 95.86%, meaning that 4.14% of the locations were still blank when the puzzle was finished, which we filled with random letters.

3 POST-PROCESSING PRINTOUTS AND IMAGES

With the puzzle grid created, our next step was to create the final puzzle and solution key.

3.1 Layout

First we output the 948 x 948 character array as a flat ASCII file. Next, using Affinity Publisher desktop publishing software, we placed the ASCII file into a document sized for a 72 x 78-inch poster. This size was chosen because it was the smallest printed poster size possible using a readable text size, as described below.

We set the character array to the non-proportional font Consolas at 7pt bold, which, in our experience, is a minimum size for comfortable human reading. Additional settings applied to the text include the following:

- Tracking: -66
- Leading: 7 pt.
- Justification: Full

To make it possible to publish a solution for the word search, we added a base-10 tick mark counter system along each of the four sides of the grid. This system uses a lowercase light blue o for the 5 place, a dark blue bar (|) for the 10 place, and a gray dot (.) for every remaining space. For example:

....o....|....o....|....o....|

We then added a number at each 10 place.

We added a title and minimal explanatory text above the grid, and added a very large, semi-transparent vector icon of a globe behind it.

We also converted the PDF poster to a single JPEG image using the Linux command:

```
pdftoppm -jpeg -rx 150 -ry 150 PlaceNames-WordSearchPoster.pdf \
   PlaceNames-WordSearchPoster-150dpi.jpg
```

At 150 pixels per inch, the resulting image file is 10,800 x 11,700 pixels; at 300 dpi, the image file is 21,600 x 23,400 pixels.

As we continue to search for a print-on-demand service that can accommodate such a large poster, we produced a full-size physical poster by printing it in letter-size tiles using our office laser printer. We used the Linux PosteRazor program to create the tiled image tiles. Printed at 13.3 letters per inch in both axes, the 72 x 78-inch poster required 72 letter-sized sheets which we trimmed and pasted together.

3.2 Solution guide

Reasonably solving this puzzle requires a list of the words to search for. In addition to generating the actual puzzle grid, the original algorithm described above also output the solution. We used this output to create an accompanying 959-page PDF

file that contains one page of explanation followed by an alphabetic list of all the words in the puzzle[4].

Each word is preceded by the column and row number of the first letter of the word and an arrow to indicate the word's direction. For example:

6,141 ↙ POZOS

This entry tells you that the word Pozos begins in the 6th column in the 141st row, moving diagonally downward toward the left, as shown below:

The solutions consisted of 163,563 lines of text. We used LibreOffice Writer to convert those lines into a 958-page, three-column PDF document. We used Affinity Publisher and Adobe Acrobat Pro to insert a page of explanation at the beginning of the document and to add bookmarks in the PDF corresponding to the letters of the alphabet for ease of navigation.

[4] Solution document available at ptolemypress.com.

Places of the World