Note: It appears that the maximum input that OCRSpell can proces at once is 13887 words in a file.

So a limit there must be considered.

Taking the previous 684,079 candidates from my 173 3-grams phrases (FR940104.0.candidates). I applied Levenshtein distance to reduce the number of candidates per 3gms.(refineCandidates.py)

Here is a summary of my findings:

Out of the 173 3grams:
-76 3gms had 0 candidates given from Google1T. That means we didn't find anything for these in previous steps. Possibly because multiple words found together are misspelled or simply it's not a phrase that existed in Google1T in the first place. So in reality we are working with about 100 3gms with candidates.
-This section of the program. Reducing candidates etc, took ~3 seconds which is pretty good.
-Approximately 20 3gms had all their candidates lost after levenshtein distance limits were placed. This means that all the candidates were not remotely close to the suggestions. This could be due to many things and I will look into it later. That's about 20% of our 3gms lost.

Depending on the maximum Levenshtein Distance I allowed, here are how many of the 173 3gms had candidates remaining. Remember, we are really only working with about 100 3gms that had at least one candidate before reducing/refining them:

1: 30 with largest 5,3,3
(this means, max 1 edit distance allowed, and the top 3 3gms with most candidates remaining after reduction had 5, 3, and 3 candidates respectively)
similarly,
2: 48 with largest 74, 28, 19
3: 58 with largest 2071, 1707, 317
4: 70 with largest 5207, 4106, 3041
5: 77 with largest 19183, 9932, 7412
6: 83 with largest 26364, 16383, 11617
7: 87...

6 out of the 173 have no candidates. From what I saw they are ones that have several words together with mistakes.

As for the
1: 30: 5, 3,3. This means for a dist of 1 levenstein we have 30 3gms with candidates of which our top 3 3gms have 5, 3, 3 candidates. Thats not a lot but when we look at levenshtein distance of 4:

4: 70 with largest 5207, 4106, 3041

This means with a levenshtein of distance 4 we remove any candidates above that leaving 70 3gms with candidates of which the top 3 3gms with most candidates have 3,4,5k usually because they are common 3gms. Those values usually cover over 3/4 of our candidates. If you open the tar.gz and look at folder 4 inside is history.log which shows candidates for each 3gms and we can also use that to see which 3gms dont have candidates.

This goes on to tell us somewhere around 4-5 edit distance is the best spot in terms of having about 70-80% of 3gms with candidates remaining while keeping distance as strict as possible.

This also tells us (along with a list of number of candidates per 3gms, in the history.log of each folder) that about 3/4 of all the candidates are in the 3 largest 3gms candidate pool. This is due to those 3gms being very generic(An example is the 3gms 'both X and') therefore giving a huge amount of candidates as they appear with a lot of different words/context in Google1T.

Running with the Ratcliff/Obershelp algorithm just to see a comparison gives me similar results.

I also need to go back and add a search for upper and lowercase in Google1T to make full use of all the 3gms possible.

FR940104.0.candidates.noCand, this file contains all 76 3gms that had no candidates in the Google1T list. As I suspected, the majority of them is due to having spelling errors in the first word as well. This can be improved upon by having multiple passes through the spelling correction.

What these values means is that when I am reducing the number of candidates based on levenshtein distance, I put a threshold. Suppose I say I want an edit distance of 4 or less in my candidates. What that means is that any potential correct word has to be within 4 edit distance. By doing this out of the 97 3gms that did have candidates, how many of those have potential correct words within that distance? The answer is 70 of them. Suppose I go more lenient and allow edit distance of 7. In that case I have 87 3gms meaning that only 10 3gms have candidates whose suggested words have an edit distance of 7 or more. In the other way, If I only allow an edit distance of 2, then I only have 48 3gms with candidates that have suggestions within that edit distance. These values tells us that this is an S shaped graph where somewhere around 4-5 is the ideal point to get the most candidates without sacrificing accuracy.

With that said the other values I had in there were to show that about 3/4 of the candidates regardless of levenshtein distance we pick, come from 2 or 3 of the 3gms that are very generic for example: 'both X and' and skew the number of total candidates making it seem far bigger than it is. So when we reduced from 682700 candidates to 16123 candidates with a levenshtein distance of 4. What that really means is about 4,000 candidates for 100 3gms.

So now let's look at some data from the 4 folder. FR940104.0.candidates.clean : This file shows us the 3gm and its candidates one by one.

Sample:

```
%%% Federal hegister Vol.
Federal Register Vol    8908
Federal Register Vol.59    41
Federal Register Vol.60    75
Federal Register Vol.61    81
Federal Register Vol.62    83
Federal Register Vol.63    69
Federal Register Vol.65    82
Federal Register Vol.66    57
Federal Register Vol.70    63
Federal Register Volume    1833
&&& Register
%%% Tuesday1 anuary 41
&&&
```

The above shows the candidates for the "Federal hegister Vol" and then on the &&& line shows just the words which in this case are our suggestions. For that case we only have one. "Register" which happens to be the correct word. So although we have 10 candidates for this single 3gms, in reality we only have 1 suggestion for the 3gms, the correct one.

Next, "Tuesday1 anuary 41" we see has no suggestions. This is most likely due to the 1 at the end of the first word screwing up our candidate search. I can either correct that first or clean out the 3gms by removing numbers inside of words.

So that's that file. If we look at FR940104.0.candidates.suggestions that file only has the incorrect word for each 3gms and the suggested correct words. This gives us a true insight onto how many suggestions per incorrect word we have. A look at this shows us that for the majority of them we do have the correct word being suggested but there still needs to be some weighting to pick it as #1 among the choices which I plan to do using levenshtein distance so that words closer to the typo are weighted higher (so if we have a threshold of 4 edit distance, but one of our suggestion is only 1 edit distance away we want to give this priority over one that is 3 edit distances away.... in addition to taking frequency into consideration as well), but ultimately according to google 1t frequency should be a defining factor.

Lastly, history.log has a lot of good information as this is the script output as the python file runs.

Note: feeding OCRSpell the cleaned version of the TREC-5 test file I am using for ground truth. That is I fed it the correct text to see if it would detect any false positives so I can then expect those in my OCR'd version potentially.

Attached is the OCRSpell outfile.

Overall it detected a couple of numbers as spelling errors with the majority having punctuation or other characters around them like parenthesis or $ symbol. Some were just numbers (like years). It also detected a couple of hyphenated words such as 'Caifornia-Arizona' as spelling errors, with some of OCRSpell suggestions being the non hyphenated version as correct, but some without any suggestions. It also detected most Acronymns or special abbreviations such as (c) as spelling errrors

As expected all of these also showed up in my OCR'd version that I fed to OCRSpell. I have a couple of ways I'm thinking of dealing with this but thought I would share.