aligner's Algorithm

kaegi

September 3, 2017

1 Notation

Let n be the number of lines in the incorrect subtitle file. Let n_r be the number of lines in the reference subtitle file. $s[0]$ to $s[n-1]$ denote the timespans for the subtitle lines in the incorrect subtitle file, $s_r[0]$ to $s_r[n_r - 1]$ denote the timespans for subtitle lines in the reference subtitle file.

The integer values $s[i].start$, $s[i].end$ and $s[i].length = s[i].end - s[i].start$ denote the start and end timestamps, as well as the length of every line (simlar for $s_r[i]$). The algorithm is based on discrete integer timestamps, meaning that real numbers as timestamps have to be converted into intergers in an arbitrary resolution (aligner uses milliseconds as resolution by default).

The lines have to be sorted, so that $s[i].start \leq s[i+1].start$ is true for all $0 \leq i < n-1$ (the same for $s_r[i]$).

After using the algorithm, the new line timestamps are called $s_c[0]$ to $s_c[n-1]$. Because the lines will only be shifted, but not transformed otherwise, the equality $s[i].length = s_c[i].length$ holds for all $0 \leq i < n$ (start and end timestamps will probably differ however).

Let $\langle s, l \rangle$ denote a timespan that starts at timestamp s and has the length l, meaning $\langle s, l \rangle.start = s$, $\langle s, l \rangle.end = s + l$ $\langle s, l \rangle.length = l$.

$s_c[i] - s[i]$ is called delta for the line i. This is the time by which the line got moved.

Let $m = s[n-1].end - s[0].start$ and $m_r = s_r[n-1]end - s_r[0].start$. This is the length of shortest time span which contains all incorrect lines/reference lines.

2 Preparing the subtitles

After choosing a resolution and converting the timestamps into integers and ensuring that $s[i].start \leq s[i].end$, the lines have to be non-empty, non-overlapping and sorted by their starting times. This can be easily done by sorting by $s[i].start$, filtering out all lines with $s[i].length = 0$ (which get the deltas of their successors) and then grouping lines with $s[i].end > s[i+1].start$.

For simplicity in the formulas later on, all lines (incorrect and reference lines) are shifted by the same time so that $s_r[0].start = m$.

3 Metric

At the core of the algorithm is the *rating* of an alignment. For each pair of subtitles/timespans $a = s_c[i]$ and $b = s_r[j]$ the rating is

$$r(a, b) = \frac{\text{overlapping\_time}(a, b)}{\max(a.length, b.length)}$$

where overlapping_time is defined as following:

$$\text{overlapping\_time}(a, b) = \begin{cases} \max(0, a.end - b.start) & \text{if } a.start \leq b.start \\ \max(0, b.end - a.start) & \text{if } b.start > a.start \end{cases}$$

The maximum of this rating $r(s_c[i], s_r[j])$ is 1, if and only if $s_c[i].start = s_r[j].start \land s_c[i].end = s_r[j].end$, because the overlapping time can not be greater than the length of the individiual lines. In case two lines do not overlap, the rating is 0.

A first attempt for the *total rating of an alignment* can be:

$$\text{total\_rating} = \sum_{i=0}^{n-1} \sum_{j=0}^{n_r-1} r(s_c[i], s_r[j])$$

This formula will give the highest rating to an alignment, where every line $s[i]$ is moved to the most similar reference line $s_r[j]$ (in terms of the length). There are two problems with it:

- The order of the incorrect lines can be changed. A basic assumption should be that the lines might have wrong timings, but not the wrong order.
- Every line probably gets shifted by a different delta. We would like to ”group” many lines which get the same delta (so only long breaks get filtered out).

The first point can be addressed by specifying a simple constraint. All valid alignments have to have $s_c[i].start \leq s_c[i+1].start$ for all $0 \leq i < n - 1$.

The second point can be addressed by a new value called *split-penalty*. The semantics of this value is how hard the algorithm tries to avoid ”splitting” the original subtitle file into independent blocks with different deltas. For every two consecutive lines which have a different delta, the rating is subtracted by the *split-penalty* d (which can be choosen by the user). The algorithm uses a slightly different total rating (for implementation reasons) by *adding* the *split-penalty* to all consecutive lines, which have the *same* delta. The lines with different deltas get ”left” out, so the best alignment for both rating variations is the same.

In mathmatical terms:

$$\delta(i) = \begin{cases} 1 & \text{if } s_c[i].start - s[i].start = s_c[i+1].start - s[i+1].start \\ 0 & \text{otherwise} \end{cases}$$

Then the *total rating of an alignment* in aligner is:

$$\text{total\_rating} = \sum_{i=0}^{n-1} \sum_{j=0}^{n_r-1} r(s_c[i], s_r[j]) + \sum_{i=0}^{n-2} d\delta(i)$$

under the constraint $s_c[i].start \leq s_c[i+1].start$.

4 General overview

This algorithm computes the alignment which yields the maximum of rating of all possible alignments with the given contraint in $O(n \cdot n_r \cdot \max(m, m_r))$ time and space. The algorithm is powered by the principle of dynamic programming.

It is obvious that $s_c[0].start \geq s_r[0].start - m \overset{\text{see } 2}{=} 0$ and $s_c[n-1].end \leq s_r[0].end + m = 2m + m_r$, because these are the extremes if we take the whole incorrect subtitle file and shift it so that each incorrect line ends before any line of the reference subtitle file begins (or every incorrect line begins after the end of all reference lines). This way the search space is limited by $2m + m_r$ (starting with $s_r[0].start - m$ and ending with $s_r[n_r - 1].end + m$).

The algorithm has n phases and each phase $2m + m_r$ sub-steps: in phase i and step t we calculate the rating of the best alignment which only includes the incorrect subtitle lines $s[0]$ to $s[i-1]$ where $0 \leq s[i-1].start \leq t$ (and because of the constraint $0 \leq s[0].start \leq \ldots \leq s[i-1].start \leq t$ is true).

We need the $o : n \times (2m + m_r) \to \mathbb{Q}$ helper-function which is the *overlapping rating for the line n with the start timestamp t* :

$$o(n,t) = \sum_{j=0}^{n_r-1} r(\langle t, s[n].length \rangle, s_r[j])$$

This value takes the incorrect line n, shifts it so that it starts at t and then computes the value it contributes to the rating through overlapping with reference lines.

A table with $n \times (2m + m_r)$ entries will be created by the following function $p : n \times (2m + m_r) \to \mathbb{Q}$:

$$p(i,t) = \begin{cases} 0 & \text{if } i = 0 \\ p(i-1, 0) + o(i, 0) & \text{if } i > 0 \text{ and } t = 0 \\ \max\{p(i, t-1), p(i-1, t) + o(i,t)\} & \text{if } i > 0 \end{cases}$$

The value $p(n-1, 2m + m_r - 1)$ is the total rating where the constraint is satisfied, but still *without the split-penalty.*

It is easy to see that in phase 0 (aligning no incorrect lines to the reference file), the rating of that alignment is 0.

To simplify the problem, we assume ‘start(sub)‘ is the start timestamp in milliseconds of a subtitle line ‘sub‘ and ‘0 <= start(sub)‘ is true. Let’s say ‘get_rating(t, n)‘ computes the best rating/alignment for the first ‘n‘ incorrect subtitle lines with the additional constraint ‘0 <= start(sub) <= t‘ for each of these ‘n‘ subtitles.

Of course we can now simply set ‘get_rating(t, 0) = 0‘ because if we have no incorrect subtitles to align, we have a rating of zero (independent of ‘t‘).

Now we handle the case ‘get_rating(0, 1)‘. We can simply compute the overlapping rating (where the first incorrect subtitle starts at the ’zero timepoint’) with every reference subtitle and add up these values. With ‘get_rating(1, 1)‘ things get interesting. We can either have ‘start(sub) = 0‘ or ‘start(sub) = 1‘. Fortunaly we already have ‘get_rating(0, 1)‘, so we only need the rating where ‘start(sub) = 1‘. This can be computed by adding up all overlapping ratings. Similarly we can compute ‘get_rating(2, 1)‘ by taking the maximum of ‘get_rating(1, 1)‘ and the rating where ‘start(sub) = 2‘. In this vein we can create ‘get_rating(t+1, 1)‘ from ‘get_rating(t, 1)‘. We can also speed up computing the overlapping rating, because the subtitle line will only be shifted by 1ms from ‘start(sub) = t‘ to ‘start(sub) = t + 1‘. The subtitle will lose the rating for the segment ‘[t, t + 1]‘ and gain the overlapping rating for the segment ‘[t + length(sub), t + 1 + length(sub)]‘ on the other side. By creating a lookup-table for reference subtitles for every ‘t‘ this process has a runtime of ‘O(1)‘.

When do we stop? Well, the rating won’t change anymore if ‘t‘ gets big. At the latest when ‘start(sub)‘ is be greater than any of the reference subtitle lines, because after that the overlapping rating will always be zero. Let’s call ‘max_t‘ the timepoint where all incorrect subtitles have been moved behind the reference subtitles. The best total rating is then ‘get_rating(max_t, number_of_incorrect_subtitles)‘.

Now we have all ‘get_rating(0, 1)‘ to ‘get_rating(max_t, 1)‘. To compute ‘get_rating(0, 2)‘, which means that ‘0 <= start(sub0) <= 0‘ and ‘0 <= start(sub1) <= 0‘. We already have the rating for ‘sub0‘ in form of ‘get_rating(0, 1)‘. We only need to add the overlapping rating for ‘sub1‘. To get ‘get_rating(1, 2)‘ we can either use ‘get_rating(0, 2)‘ (we leave ‘sub1‘ where it is), or move ‘start(sub1)‘ to 1, which allows ‘start(sub0)‘ to be in ‘0 <= start(sub0) <= start(sub1) = 1‘. The best rating for ‘sub0‘ for that range has been computed with ‘get_rating(1, 1)‘, we only need to add the overlapping rating for ‘start(sub1) == 1‘. We proceed similarly to get ‘get_rating(t+1, 2)‘: leave the ‘sub1‘ like it was for ‘get_rating(t,2)‘ or reposition the subtitle to ‘start(sub1) == t+1‘ and use ‘get_rating(t+1,1) + overlapping_rating(sub1,t+1)‘.

With the same principle we proceed with ‘subN‘:

- initialize ‘get_rating(0, n) = get_rating(0, n - 1) + overlapping_rating(subN, 0)‘ - choose for ‘get_rating(t+1, n)‘ the maximum of - ‘get_rating(t, n)‘ which means ”leaving ‘subN‘” and - ‘get_rating(t+1, n-1) + overlapping_rating(t+1, subN)‘ which means repositioning the ‘subN‘

Until now we didn’t use the ‘split_penalty‘. We need to add the split penalty when ‘start(subN) - start(subN+1)‘ is a specific value (the original distance ‘diff(N)‘). The trick here is seeing that we only need to consider the ”repostion choice”. The only time ‘get_rating(_, n-1)‘ is consulted after the inital phase is when ‘subN‘ gets repositioned. ‘subN‘ will then start at ‘t+1‘ and we consult ‘get_rating(t+1, n-1)‘. So if ‘subN-1‘ were positioned at ‘t+1-diff(N-1)‘ for ‘get_rating(t+1-diff(N-1), n-1)‘ we’d be able to get the ‘split_penalty‘. This is exactly the thing we will do when we are in a phase ‘n‘: We will not only have the ”leave choice” or ”reposition choice” but also the ”nosplit choice”. If we compute ‘get_rating(t, n)‘, we can also compare the two other values with ‘get_rating(t-diffN, n-1) + overlapping_rating(t-diffN, subN) + split_penalty‘. The ‘get_rating(t-diffN, n-1) + overlapping_rating(t-diffN, subN)‘ is again the best rating where ‘start(subN) = t-diffN‘. We are allowed to add the ‘split_penalty‘ because in the next phase ‘n+1‘, ‘subN+1‘ will start at ‘t‘ when ‘get_rating(t-diffN, n)‘ is looked up. So the final rating algorithm is:

- initialize ‘get_rating(t, 0)‘ with 0 - initialize ‘get_rating(0, n) = get_rating(0, n - 1) + overlapping_rating(subN, 0)‘ - choose for ‘get_rating(t+1, n)‘ the maximum of - ‘get_rating(t, n)‘ which means ”leaving ‘subN‘” and - ‘get_rating(t+1, n-1) + overlapping_rating(t+1, subN)‘ which means repositioning the ‘subN‘ and - ‘get_rating(t+1-diffN, n-1) + overlapping_rating(t+1-diffN, subN) + split_penalty‘ which means doing a nosplit-repositioning for ‘subN‘

To get the final alignment, we save for each phase ‘n‘ and ‘t+1‘ where ‘subN‘ was positioned (can be ‘t+1‘, ‘t+1-diffN‘ or the previous position). If we look up that value for ‘n = number_of_incorrect_subtitles‘ and ‘t = max_t‘, we know where the last subtitles ‘subN‘ has to be. We then know ‘start(subN)‘. The best alignment of all previous subtitles is then computed with ‘get_rating(start(subN), n-1)‘. So we look up the position for ‘subN-1‘ in that table with ‘n’ = n-1‘ and ‘t’ = start(subN)‘. That way we get all corrected positions of all incorrect subtitles and are done!

Though this algorithm works (and was implemented in one of the early versions of ‘aligner‘) it is neither fast nor space-efficient. Let’s take a ‘45 minutes = 2700000 milliseconds < max_t‘ subtitle file which has about ‘n = 900 subtitles‘ (these are realistic values). We build a table of ‘max_t * n = 2430000000 ‘ entries. We can discard the ratings of the phase ‘n-1‘ after phase ‘n‘, but we always need to store the positions of the subtitle ‘subN‘. Let’s assume we need 4 bytes to store them: we then have a table of ‘2430000000 * 4 bytes = 9720000000 bytes = 9 GB‘ of data in RAM!!! Even filling the table with zeros might take some noticeable time. But as it turns out we can compress that table in under 2 MB (most of the time; probably the best compression I’ve ever seen) with ‘delta encoding‘. The empirical foundation is that the choices almost never change from one ‘t‘ to ‘t+1‘ (about 10 to 1000 times for one phase). If we always take the

- ”leave choice”, the position will always be ‘t+1‘ for every ‘t+1‘ (rise by 1) - ”nosplit-reposition choice”, the position will always be ‘t+1-diffN‘ (rise by 1) - ”reposition choice”, the position won’t change from ‘t - 1‘ (constant)

So if we store values in a ‘(start, delta, length)‘ tuple, where the first uncompressed value is ‘start + 0 * delta‘, the second is ‘start + 1 * delta‘, the third is ‘start + 2 * delta‘, ..., the last is ‘start + length * delta‘, we can compress an entire phase into a few bytes. The same thing is applicable to the ratings. Without going into details: if we take the overlapping rating of a incorrect subtitle to a reference subtitle and ”move” the incorrect subtitle from the far left to the far right we will have five segments of compressed values (first the rating will be zero, then rise linearly, then be constant, then fall linearly, then be zero again). The comparisons/choices can then be done for rating segments instead of single ‘t‘. This yields a speedup of at least one order of magnitude.

- article
- book
- report
- letter

1. article
2. book
3. report
4. letter

article Article is ...

book The book class ...

report Report gives you ...

letter If you want to write a letter.

5 Conclusions

There is no longer LaTeX example which was written by [Doe].

References

[Doe] *First and last LaTeX example.*, John Doe 50 B.C.