Masters Thesis

Part 3 of the document.

Post Meeting 4 Notes

Tasks:

- Find open subtitles, SRT files (we should have some, check Leverage and other Movies and Shows in our library)
- Run similarity algorithm on sample traces of labeled data
- Consider automated process to expedite our labeling

Next meeting is 19 Oct (Mon) @ 13:00

Labeling Process Notes

We took the SRT files from the Graham Norton Show, which is a sort of late-night host talk show where celebrity guests are invited. It resembles casual conversation with a direction so it's actually very good to use. However, the challenge is that multiple people are always talking, and the subtitles also contain audience reaction, though the guests do interact with audience members as a third collective.
We immediately notice that our labels are a little vague again, but this is to be expected. Here we should note how we're condensing labels, though we should honestly add new ones to be more accurate.
Additionally, in the talkshow, there are places where the closed questions immediately are answered and grouped with the question, so we should be careful to just list those as respond.agree, unless there is a separate question before, then to which we leave it as a question tag… but we should maybe also think about a better solution for this.

Labels that we may need:

- misc - to indicate applause and other things like actions/gestures, instead of using use.social.convention or x.
- give.statement - instead of using give.opinion, since there's a big differential here between a value neutral statement rather than a statement that has an opinion.

- recall - we should have something to indicate that a speech contains quoted speech, which is unique. Or maybe put it under a category of "give.recall"
- relax.atmosphere - we can expand this definition to include statements that are meant as a joke/banter (since the goal is to impact the mood, not necessarily involved with using social conventions) and maybe expand to just exclamations too?

We may also need to consider sub-conversations and how we can make them generic enough to catch multiple speakers in a conversation, i.e: catch trends without specific labeling on who the speaker is. This may be difficult, and perhaps more inaccurate, but something we should think about.

We definitely have to alter the labels and create a more accurate labeling process. Expanding the genre makes it more apparent. Perhaps we need to think about creating a good set of labels that can fit most conversation in different mediums, and from labeling and process determination, we can discover some sub-conversations, and those can help us determine the archetype of conversation (interview show, extended documentary, language exchange, etc) it becomes more apparent.

Post Meeting 5 Notes:

Tasks:

- ~~Finish sub cost~~
- ~~Finish indel cost~~
- Finish clustering
- Have examples confirming the validation of the functions (breaking each step into a separate part might help with this)

Next meeting is Monday, 26 Oct

Post Meeting 6 Notes:

Tasks:

- More formal explanation of the functions
- Move beyond the paper

Next Meeting is Monday, 2 Nov

We've fixed some of the issues with the substitution and indel scores, and they should be for the most part correct (maybe with exception to certain edge cases). Additionally, we've also been able to replicate the table-algorithm for calculating similarity (it's the same algorithm for edit distance, just the opposite). Additionally, we have to account for the fact that traces can have different lengths, so we enforce that the shorter trace be the base comparison.
In the paper, they suggest that insertion is the complement of deletion, but the implementation for our case gets affected when we implement this because of the many cases where there are cases of labels having a severely low insertion score, causing deletion to increase the similarity score too much. For our current purposes, we have listed deletion as a unit 1 similarity cost.

Now, we're moving onwards first to see how we can further add on to the concept of similarity by adding a notion of "sub-conversations." Which are larger trends in the conversation trace than just co-occurrence and context, which seem like intuitively should be tighter and shorter distance similarity measures, whereas a larger presence of similar "archetypes" of conversation should still contribute to similarity. (i.e: for us, our graham norton show traces shouldn't be that similar to the blackpink documentary, but in a sense, they are still interview based traces, and there should be some attention given to that).

So, for our current situation, our similarity metric is displaying that trace1 and trace2 (the two Graham Norton traces) are similar (positive score) whereas trace1 and trace3 (the blackpink documentary) are dissimilar (negative score). However, as we mentioned, there should be some notion where these two are similar, so perhaps we can look to define other metrics to display similarity. (i.e: WHERE are these two traces similar? For us, it would be the interview format, as well as the

idea that speakers display a high amount of recall). We can start building by first defining these trends in the abstract?

Let's maybe start with brainstorming labels that we would choose to define what we intuitively would call a sub-conversation?

> Monologue - maybe something like a long string of give.statements, with some give.opinions inside? (marked by a respond.agree/respond.deny, since a monologue is broken up by the interjection of another person, in this case of speakers not being labeled, we can assume that a respond label is essentially interjection. Things like questions can still be rhetorical)
>
> Making a deeper connection - this would be something along the lines of recall until question or recall until give.statement/opinion or recall until respond.agree/deny for a chain sequence of recalls. The idea is that one person asks a question that is followed by a longer sequence of recalling, which usually indicates sharing a memory, which in polite conversation is how people tend to form a deeper social connection.

Then we can move onto considering how we would discover this longer trend of labels, maybe something like LTL? Or a variation of LTL to check a longer sequence?
Additionally, there is the issue of determining the speaker. Is there a need for us to determine the speaker currently? Because if we separate speakers as labels, then we'll only drive the alignment for similarity of two traces further apart, but if we don't it might be harder to apply notions of longer sequences in an intuitive way? I guess we can say statistically, long sequences of give.statements/give.opinions can be considered a monologue because it's unlikely two speakers interchange give.statements or give.opinions without some sort of interjection. Additionally, the same goes with long sequences of recalls. I guess you could call it human nature. Can we think about something like, recall until question or recall until give.statement/give.opinion/respond.agree/respond.deny. This very linear notion

would fit the bill for something like a question about someone's background, which is a likely candidate to pop up in LE and conversation as a format.

I feel like even looking for sequences for just these two to begin with would already be a difficult task. We can probably just start from there. Finding a long sequence using LTL or some variation of logic is algorithmic, isn't it?

Post Meeting 7 Notes:

Tasks:

- Look at Cost-LTL, which is a LTL variation that works on counters and allows us to have bounds on the number of occurrences in a sequence (useful for things like, "recall should not last more than X utterances).
- Longer term dependencies in sub-conversation discovery that can be used in frequency analysis to further define a similarity metric for our traces!

Our current format (Check Meeting 7 slides) is preferable and good work. We should keep on with this level of explanation and setup for our presentations.

Next meeting is Monday, 9 November

Notes on Cost-LTL related papers

Dimitri mentioned the concept of a LTL variation that is bounded by a cost function. This is something that would be helpful for us in terms of expressing longer-dependency sequences in our system, and have a notion of bounding the number of times a label can occur (via a cost or some value).

From a cursory search, I can't seem to find much beyond the paper, "Linear Temporal Logic for Regular Cost Functions" But it seems relatively recent, and is probably where I should start.

"Linear Temporal Logic for Regular Cost Functions"

- The crux of the argument here is that LTL has good properties, and maintaining those properties while allowing for the extension to counting capabilities is very useful.
- They introduce a new operator $U^{\leq N}$ which implies x $U^{\leq N}$ y means that x holds until y, except at most N times.
- Intuitive and makes sense. We'll proceed and see how they set up this new operator, and how they apply it, and maybe get some insights on how we can approach using this notion in our algorithms to maintain the same notion of counters.
- They suggest their motivation is that if we can bound the number of occurrences of certain outcomes, then we can guarantee the global bounding of the number of "mistakes" (defined events) that occur. This also makes sense when talking about in the context of cost functions.
- For us, our usage of it would seem more on the lines of, we define certain sequences using this bounded metrics for similarity (i.e: long monologues and short monologues are built different). But thinking about it like this, perhaps the bounding in this manner is less important for us. Maybe there is a different bounding that we should be thinking about. Regardless, on face value, this is what we would use it for.
- There are a lot of technical definitions, but the start to the part that we perhaps care more about starts at Definition 4.1, where we outline the extended LTL to describe cost functions. The grammar is listed and contains conjunctions, disjunctions, next (X), until (U), until-N ($U^{\leq N}$). Which tells us that the concepts expressed here can be built using just these notions of operators (which is important for me to know when concerns of implementations are important).
- I get the distinct feeling that this comment will be useful during implementation somewhere, "Remark that we do not need a dual operator for U, because we can use Ω to negate it: $\neg(\phi U \psi) \equiv \neg\psi U(\neg\phi \vee \Omega)$."
- They also have ways to define Eventually (F) and Globally (G) through using untils with TOP or BOTTOM and the Ω alphabet, which stands for the end of the word.

- The goal here for them is to be able to extend LTL such that we can associate a function with the LTL formula.
- Also note that they define that the $U^{\leq N}$ operator always appears positively in the formula, meaning that if (u,n) models φ, then for all k <= n, (u,k) models φ as well. (Which makes sense since if you're bounded by a value, if we increase that value, then you should still be bounded)
- They describe further on the proofs for showing the complexity of the extended LTL formula, but we're not too concerned with that. What we've noted above is sufficient for applying the extension to define subsequences that are of interest to us.

Taking a step back, thinking big-picture

We've done the preliminary implementation of until and until-N, but of course we'd have to modify it for a similarity measurement purpose. However, before that, we would like to take a moment to think and consider the bigger picture, about how what we're currently doing fits into our larger goal.

Of course, our goal is to develop something that would be useful for conversation modeling. We're doing conversation modeling from the perspective of process analysis rather than the linguistic aspects (like NLP). If we can think about it as two sides of the same coin, the coin (goal) of both fields is to create tools that aid in language learning/processing. The NLP side aims to address notions of language from a linguistic sense, and often take ML models to turn words into numbers and values, and attempt to optimize and solve the problem through that method. Our approach is from the field of process mining and model verifications. We see the problem and aim to optimize and solve it through turning words into collection of processes to be analyzed.

Going from that top-level goal, we move down to what we're currently doing. We're determining similarity measures in order to compare traces of conversation to see if we can differentiate them using some metric. We do this because this allows us to attribute some properties onto these traces, such as similarity in purpose (in the conversation) or effect (on the conversation), as well as properties such as containing sub-conversations (that we've defined) which can also be defining factors to these processes. This is all for the express purpose of suggesting that

by applying these notions onto traces of conversations, that we can derive some useful information. Most process analysis work has been largely applied to business mechanics, or in our CS field to robotics (extremely loose interps) and gene-sequencing (extremely strict interps). We believe language conversations are somewhere in the middle (in terms of strictness/looseness). This allows us to tack on top some extra properties/analysis (i.e: sub-conversations, maybe look at probabilities of occurrence of sub-conversations based on trends of sequences, etc). All of this could be then applied to be useful in a larger picture of language processing. (i.e: think about using these models of analysis to detect developed speech vs simple speech of children, has educational purposes).
We're very happy with this direction so far, so we'll keep pushing sub-conversation discovery (which is for what we've seen, relatively new) and all that it could entail (probabilities of occurrence, similarity/differences in resulting models, etc). EXCITING!!!

Preliminary sub-conversation similarity exploration
Wrote the function for Until-N, hopefully to be able to describe sub-conversations of the form: Give.Monologue, and Recall.From.Memory
We get some potential sequences back, we'd have to determine what the N cutoff should be, as well as how accurate we think the returned sequences really match the concept of the sub-conversation we're thinking about.
Additionally, we should have a function to look at relative position of defined sub-conversations to determine their weight when we do comparisons of similarity via sub-conversation frequency contribution.
There's also a consideration to omit/remove short sequences that are only length 2 or 3, since they hardly seem like they would qualify as the long-term dependencies that we're really looking for.

Some properties that could potentially imply good representation?

- Low N violation value, High length?
-

However, seems like from our sample, there isn't many that fit this description. (Meaning there aren't a lot of long runs of "give.statement") but there are some that may fit the bill for recall (which we expect, since we had this idea in mind when looking at the Blackpink documentary trace).

There's the distinct possibility that we can create sets that fit the best properties/representation, and then those results are the "discovered" sub-conversations that we get. (i.e: from the given trace, try different combinations of labels for the sets of expressions for until-N that best fit the properties we want from above, such as low N violation value, with high length)

Starting with a simple strict one-label -> one-label definition, we generate possible sub-sequences for consideration (though we need a function to filter out qualities that we want to look for, such as minimum length and number of N violations).
We can probably expect the strict one-labels to generally not yield results in most cases. (Intuition suggests that this expression would match cases that simply yielded lengths 1, which are essentially X (Next) function matches).
As a general rule, we'll probably only want to expand the combinations for the second set of labels, as that is the one putting the bound on the number of N violations, as well as the length.

Post Meeting 8 Notes:

- There's a concern with the number of variables and combinations involving the different combinations of labels on top of looking for values of c and l. We should find a way to reduce this. (We talked about how we can use similarity measure from before, maybe something like using the sub scores to determine which pairs of labels to try or put in the same set).
- We want to start correlating the idea of similarity with the occurrences of LTL satisfactions. Not just in the cost LTL sense, but also in the classical LTL senses. We want to get to the point where we can say if two traces are similar (based on our similarity metric), then they will exhibit similar LTL sequences.

- There was the term to look up "Full Abstraction" in temporal logic.
- More samples/visuals next time for the slides.

Next meeting is Monday, 16 Nov

Some sequences from one->one label set, bolded the ones that seem like they can be considered:

use.social.convention -> (closed.question, deflection, relax.atmosphere) x1 each
give.opinion -> use.social.convention x7
give.opinion -> closed.question x4
give.opinion -> open.question x5
give.opinion -> respond.agree x4
give.opinion -> deflection x1
give.opinion -> relax.atmosphere x10
give.opinion -> give.statement x12
give.opinion -> recall x14
give.statement -> give.opinion x1
give.statement -> open.question x1
recall -> use.social.convention x1
recall -> give.opinion x3
recall -> closed.question x2
recall -> respond.agree x1
recall -> relax.atmosphere x4
recall -> give.statement x4

Average length: 23.51948051948052
So 50 as a max length is probably too high, we'll probably want something around max length 30? If we try the same process with max length 30, these are the generated results:

give.statement -> give.opinion x1
give.statement -> open.question x1

recall -> give.statement x4
recall -> closed.question x1
recall -> give.opinion x3
recall -> respond.agree x1
recall -> use.social.convention x1
recall -> relax.atmosphere x3
give.opinion -> give.statement x9
give.opinion -> closed.question x4
give.opinion -> recall x10
give.opinion -> respond.agree x2
give.opinion -> use.social.convention x6
give.opinion -> open.question x4
give.opinion -> relax.atmosphere x9
use.social.convention -> closed.question x1
use.social.convention -> relax.atmosphere x1
use.social.convention -> deflection x1

Average length: 19.629032258064516
We can see that although we're limiting the length, and thus reducing the number of lines, we're still seeing that certain label combinations yield a lot of entries, which is good to see. Some of them remain consistent (the lower valued ones). This could potentially mean that they're either consistent because the line itself was very strong, or consistent because the drop of line yields weren't too bad. Either way, a good sign for lowering the length of the sequences.

We can also start to see which label sets yield lines after reducing the c value to being less than 20% of the length, which seems quite strict.

give.statement -> give.opinion x1
recall -> give.opinion x1
give.opinion -> give.statement x4
give.opinion -> recall x5
give.opinion -> open.question x1

give.opinion -> relax.atmosphere x4

Average length: 16.5625

But we can still see really consistently, that there are label sets that yield good lines. This probably means that we should go with the c values between 20%-40% (We used <50% for our numbers for the first two, but the feeling was that 20-40 is probably a good range)

*it is evident from the way we set up the function (and also the way that the concept is itself) that when we expand the second set restriction to be more than one label, we will have many categories of the same sequence, since it is an OR statement. This means we probably will dismiss looking into this avenue in favor of a different set up that makes more sense.

For comparing two traces' sub-conversations, we can take the bag of activities approach, since we're less concerned with the drawbacks of lack of context and order of appearance (since we're talking longer dependency sequences, we mind less what order they appear, just that they appear).

But maybe we can put a weight on it? Looking back on the old paper on TraceSim to see if we can get some ideas. They use frequency and relative position. I actually think I like this, since in our context, this also makes sense, although slightly different in position (we want our sequences to be near the middle of the trace). Frequency adds more weight (makes sense intuitively). We just have to use the start and end indexes (s,e) and length (l) to determine where positionally the sequence is, and assign a weight accordingly. Their metric uses TF-IDF for frequency weight placement (which makes sense, and we will most likely replicate). It's notable that they use a $1/i^a$ function to determine the local weight (positionality). We might have to take some time to consider how we can do a function that normalizes in the middle (like a normal distribution).
We know that the halfway mark should be the max, highest weight. $\frac{1}{2}$*l is considered the highest weight.

There's also the consideration of whether or not sequence length itself should affect the weight. Since we looked for specific values, probably not, but then again, we could just weigh it instead of establishing specific values?
Let's say we don't, we just use start and end (s,e) based on length (l) to determine positionality. So we count positionality by finding the midpoint of the sequence, and using that value to establish distance to the midpoint. That's one way. Then we can take that positionality value and put it under a fraction 1/x, which will have lower values generate more weight, and higher values generate less weight.

We revisited the TraceSim paper, but we should pay special attention to the TF-IDF part, since they do a variation. We should think about how the original theory works, why they vary it for their purpose, and how we can vary it to tailor it to our purpose.
For instance, we might find that every trace really contains some form of the sub-convo we're looking for, so we might have to modify IDF to take into account? Or modify something else? Because the fear is that IDF will just have every document have the sub-convo show up at least once.
Maybe this isn't an issue? Perhaps IDF as normal will scale properly for accounting for sub-convos that appear in most if not all traces in the event log?

For clustering, the paper does minimum variance between the clusters, but does say that there can be different metrics used (as long as a metric is used). There's no reason we can't go with our cases of clustering based on similarity scores, weighting, individually, and then both.
For implementation purposes, we'll see if we can use sklearn's agglomerative clustering function. We just have to find a way to modify it such that we can get it to use our metrics for comparison (i.e: similarity, weighting, etc).
Seems like we'll have to use "precomputed", which requires a distance matrix, which should just be a matrix of distance (we'll have to convert our values to represent this) between each and every trace.

We'll probably need more data soon.

Need to reconsider the weighting for clustering, as is, the function is returning total weights for each sub-conversation sequence, meaning that we'd end up clustering those rather than the traces themselves, which is what we were clustering in the previous similarity measure part.
However, there is a point in maybe finding a way to correlate these different sub-conversations to the similarity values besides clustering, that ends up improving the clustering, so there needs to be some thought given to that.
For the code currently, the values are being mixed up somehow, so we'll have to revisit and see where the values are being overwritten.

For us right now, the problem is that the total weights we're doing are individual weights on the sub-conversation that are specific to each trace (tied to TF), while IDF is shared amongst the traces of the event log. The organization might have made it difficult, but we can always fix it granting that we know what the overall goal is, i.e: how we're clustering, if we're clustering traces or the sub-conversations inside the traces. To be honest, clustering the sub-conversations probably has no real impact/use, since we're picking out the useful ones anyways and limiting with the key values on c violations and length.
We need to determine a new way to reasonably determine how different weights of a trace will compare (a metric) with another trace.
I guess it ends up being the question of how to correlate the notion of similarity/distance to the notion of weighting and sub-conversations.
The trace sim paper just modified the levenshtein distance with the weights by insertion cost as the weights of the next, and sub costs as weights of both (sum).
I mean there is a bigger idea of if we swapped out or insert/deleted values it could technically change the composition of the sub-conversation, but i dont think a singular value change will really alter the sub-conversation sequence in the general case, although further changes could alter, it might be too difficult/costly to keep track of that.
The current consideration is just, if we added weights to the levenshtein distance, does that really improve what we're looking at/does that count as forming a metric/correlation to the metric that we want?
I mean I guess I don't see why not?

There's also the consideration that we can just organize all sub-conversation lines in terms of starting index as an order, and do the levenshtein distance between two traces based on sub-conversations as the unit instead of labels, but i'm not exactly sure that really gives us a long term dependency... The idea of the weights is that they represent for a singular unit, how important it is. So I guess in that case, maybe we need to change the way we think about calculating weight.
Let's isolate to just a singular trace.
Sequence of labels, and sections of the sequence are sub-conversations (perhaps overlapping).
Each label in the sequence can be given a weight, based on how many sub-conversations it's a part of? Impacted by the idea of how important this sub-conversation is based on it's position to the center of the conversation.
So when we do TF-IDF we don't have to consider other traces, we just have to consider each individual trace and it's sub-conversations, since we're comparing each individual action and its weight in that exact position (based on its participation/contribution to that particular sub-conversation)? In that sense IDF here is measuring how many sub-conversations would be affected by this specific label. But this concept breaks the intuitive notion on TF, since there's only one action in that sub-conversation. Or can we make the TF part the number of specific sub-conversations. No, we can't. We could theoretically consider the individual label count for the sub-conversation it is a part of, how many times does the action appear inside that particular sub-convo, but it loses relation to the whole of the IDF and weight notion? Unless we consider that maybe more of the label in the sub-convo sequence is good, since it doesn't break the sub-convo sequence definition, and contributes to the importance of the line. Okay yea, I can get behind that.
So the notion for TF-IDF and weighting is as follows:
~~TF, the term frequency, is looking at how many times this specific label appears in the sub-convo it is in (generic count of label type of the specific label).~~
~~IDF, the inverse document frequency, is looking at how many sub-conversations this specific action is a part of (within the start and end index).~~
Local weight is the distance between this specific action to the center of the entire trace (we mentioned the importance of the middle of the conversation).

This keeps the general idea that each action in the trace can have a weight. Then we can use this weighting as part of a metric to affect levenshtein. We can then do the three cases we mentioned in order to see how it improves/worsens the clustering.

This means we will probably need to change some of the graphics and slides for the clustering section (we've skipped the slides for now).

So now we need to consider our definitions of TF and IDF to see if the variant really is consistent with the original function of TF and IDF. Originally, the idea is that we want to see if the term is frequent, but penalize it for being frequent in other documents. Here, we are penalizing being in too many sub-convos, which may not be the right goal. Don't we want how many sub-conversations does this label contribute to, penalize for every single sub-convo that it violates. That might be the better metric. The idea being, your TF-IDF weight is determined by how many sub-conversations you contribute to, and how many sub-conversations you detract from (violate).

For implementation, we know that the TF and IDF values for each position will be 0 unless there's a sub-convo there. We can thus start from searching the non-zero sub-convo entries in the sub-convo table. We can take each s and e sub-sequence, and search that sub-sequence, and if the label is label1 (or label2) then we increment TF by 1, otherwise we increment IDF by 1. Then at the very end, we should have all of the appropriate TF and IDF values in the tables.
We can then go through in order and extract the values for TF-IDF (TF*IDF) and sum with the local weight (which can be calculated relatively easily) and returned for each trace.

All implementation for now seems done. We'll have to get more traces for the event log if we want to compare and see results! Meaning it might be time to start looking into automatic labeling for traces! (Can you say ML model?!) Regardless, still means we probably need to manually label for more data… But we should also look into methods where we can do unsupervised learning. It's a part of the process!

Post Meeting 9 Notes:

- For the c and l violation counts, we have to ensure that we don't count the start and end label as part of the length, since they are not violations.
- Look into automated labeling, in order to get more trace data, where we want to have a variety of types of traces (i.e: talk shows, interview-based documentaries, movies (as outliers)). Starting with these are good.
- Start thinking about drawing up an outline for the structure of the thesis paper, to get a sense of what needs to be written (so that we can iterate and get everything in before we start physically writing).

Next meeting is Monday, 30 November

It's time to finally dive into the tensorflow stuff and papers to look for ways to automate labeling for the trace data!
Essentially we're looking at a classification task for intent, which should be plenty to find (a very popular task in NLP), and the variations we want to look for is anything that is unsupervised or semi-unsupervised, since we don't have a lot of resources (time).
Though it is also important to note, the solution will also just be leveraging the fact that we can spend a week labeling data manually to then train a classifier. Meaning that our focus isn't to develop a super sick unsupervised labeling model, we just want to expedite the labeling process.
We'll start with the design for a simple classifier based on data that we can just label manually. At the very least this will be a good fundamental start to Tensorflow proper.

The process will take the following form:

1. Pull the list of text (should be the U column in our data)
2. Pull the list of labels (should be the L column in our data)
3. Create a table to show label value (0-11 based on our labelset)
4. Determine basic framework of the model (i.e: layers)
5. Line up the data properly, and compile the model

6. Train the model
7. Evaluate the model
8. Apply the model to new list of text (new U column values)
9. Evaluate the new list of labels (new L column values, based on table)

Alright, let's get labeling!

Post Meeting 10 Notes:

- Work on finding more patterns inside our current conversations to give more separation between traces. See how the additions improve/alter the clustering.
- As a follow up to the first point, make more specific labels to help improve separation for clustering. Meaning have to relabel.
- Fill out the sections we can fill out for the thesis paper so we can start iterating.

Next meeting is Monday, 7 December

In terms of searching for patterns (sequence discovery), we have the deets from a paper that looks at pattern matching in sports sequences, that uses an algorithm called Safe Pattern Pruning, and benchmarks with another algorithm called PrefixSpan. The concept both makes sense in terms of how we can apply it to our use-case.
SPP is described as a tree structure that is defined as every possible pattern in the offered data. Then, pruning the tree gives the guarantee that a node once pruned, implies that the descendants will not be required for the predictive model. Whereas the PrefixSpan starts with one event, and then iteratively adds on more events to the pattern based on some metric.
Both of these algorithms are part of larger based frequency-based analysis, which is nice because we can apply it to our use-case in a very similar fashion.

So with PrefixSpan, we can discover patterns that are highly present over the ENTIRE eventlog. Meaning the patterns that it finds are ones that we would consider general "sub-conversations" and in a sense is something better than the LTL statements we've been looking at before. Well, actually, technically not better, but perhaps more intuitively what we thought about before. LTL-N looks for a different type of pattern than this pattern matching, which looks for specific patterns (exact matches). So this additional pattern matching may still be interesting/useful.
There is a good sense to utilize the library, but it isn't written specifically to our liking, so maybe there's also warrants in at least implementing the part of it we want. Though with efficiency concerns this may not be the best, we'll have to see. Either way, discovery of patterns to look for may change the way we apply weights and we could see if altering those functions perhaps could increase the clustering results.
The idea here is that if we can get the top X patterns that occurs over our eventlog, then we can start using those patterns as part of the TF-IDF variant? (However, we'd have to consider searching complexity etc). Meaning it probably serves our purpose to rewrite the PrefixSpan algorithm to fit our purpose.

The PrefixSpan paper actually gives us a good basis and idea for how to properly identify change of speaker (not necessarily identify speakers, but a grammar to talk about actions can be grouped). We'd have to consider whether or not this could be useful, as well as how we could integrate this in the most efficient manner.
The concept here is that in the PrefixSpan sequence, they group items in an element as such: <a(abc)b(cd)(ef)> where they omit parentheses if an element only contains one item (i.e: (a) becomes a). The concept is that they are grouped. However, it seems that they don't specify that order matters in this grouping (even though for us it will) and the fact that there can only be one of each item type in this grouping (which we know is not the case for us). This means that there may be resulting concerns about how if we changed and adapted this grouping, might impact the algorithm functioning on the sequence. So we might want to first

start without an indicator of speakers (perhaps this would be something good as a further study?)

There are physical notes in our notebook for this algorithm, we'll have to re-explore these concerns when we type up the powerpoint slides, but for now we want to implement the general idea of the algorithm first to adapt it to our needs (in integration with our variant of TF-IDF).

Okay in terms of further considerations, we determine that the PrefixSpan algorithm and their respective families of algorithms are concerned with mining patterns that are common between all sequences (traces). For our purposes, we don't really care about finding common patterns between traces. We care more about finding patterns that exist within each trace, since we're using these patterns as a form of identification/separation between the traces. We're not finding commonality, but rather differentials.

So we need a variant. The concept should theoretically still apply. But essentially, we turn the question into: what's the longest (or X-length) pattern we can find that occurs more than once? What's the shortest (or X-length) pattern we think is acceptable, and how many times does that pattern occur? We can think about it in a couple of phases I guess:

1. Generate the "puzzle pieces" (patterns) to look for.
2. Determine the count of the patterns (at worst size N scan).

So the problem here isn't the count of the patterns, it's determining the patterns to look for (the puzzle pieces). Generating them is non-trivial.

We have to figure out how the PrefixSpan algorithm generates its sequential patterns in detail. It seems that my first guess was correct in that the postfix projections are based on everything after the first appearance of the prefix, which for efficiency purposes, can be done via index as a pseudo-projection so we don't have to re-store values, saving memory. All we need is a start and end index (or alternatively start and length). Still having issues on how it is counting the occurrences/support for the sequential patterns, but it may be mostly stemming from how they are defining the grouping. For us, it's entirely based on singular-elements. Theoretically that shouldn't ruin the efficiency of the algorithm.

The second issue to handle is the previous one we mentioned of, we only have one trace to look for comparisons in. We could theoretically arbitrarily break the trace

into equally sized chunks to treat as a sequence database, but the question would then be begged of, how large should we make the chunks? And also, does the way we break into chunks matter? I feel like for both, an arbitrarily selected value should be fine??
Let's say we do that. The question then becomes, does it affect how we generate and mine? I don't think it does because theoretically our space still gets smaller, and the process is the same. Let's run with it.
So the process is as follows:

1. Start with length 1 patterns (freq), drop those below min-supp
2. For each group, do the same, find the freq, drop those below min-supp
3. If we at any time only have one projected in database, terminate

So the first thing we need to do is take the trace and break it into equal chunks.

1. Determine length of trace
2. Run a range with increments (runoff is fine)
3. Append to a new list as "sequence database"** (technically we append as start and end indexes for the pseudo projection optimization)

Then we need to actually run the algorithm. Which we don't want to do recursively since we're working with python. We'd rather make it iterative. Which means we have to define the iteration and the stopping conditions.
Iteration: do the counts for the sequences to get grouping, drop the ones that don't meet min-supp, store the current groupings, project for each grouping.
Stop condition: projected group has only 1 sequence, or no grouping has the required min-supp.

Start with an initial groupings list

- List of symbols
- Master list of patterns

Do the counts of sequences to get groupings

- Have a bucket of your symbols
- For each sequence

        - Check existence of symbol, increment bucket
    - Check your bucket and drop < min-supp
    - Pass bucket as list

Store the current groupings

- Store your current groupings in master list as patterns

Project for each grouping

- For each item in your list (from bucket)
    - Create a list to store projected sequences as projected database
    - For each sequence
        - Find the first occurrence and start from next index as new sequence
    - Add as projected database to your item's list

How will we alter this for the optimization of pseudo projection? The idea is that instead of storing additional sequences, we only store start and end index, then refer to the entire thing which is stored already in memory. For our purposes, we start by chunking the trace, which we technically don't have to "chunk" we can just treat that as the first sequence generation, which is done via start and end indexes. Then for each sub-sequence, we just instead of adding projected sequences, simply store start and end indexes, and pass a reference to that.

PrefixSpan algorithm coded complete (6 Dec), we'll have to integrate to weighting and then evaluate the results in the evening.

*We should scale the values of PF-ITF, since they're much higher than the weights from the trace weight calculations (which uses the cont-det values along with natural local weight values).

*We also need to reconsider and think about how the weights are being used to calculate for similarity, since there seems to be some inconsistencies/incorrectness there.

Post Meeting 11 Notes:

- Finish up new labeling and integration
- Finish up clustering
- More writing for thesis sections

Next meeting is Monday 14, December

We also want to consider if we get reasonable results to start up automatic labeling, and to start formulating a more holistic vision for the project re: developing it as a scaffold for a learning tool.

New Labels Consideration:

- exclamation
- inner.dialogue
- quoted.speech
-

Maybe more labeling/re-labeling will also help develop more ideas? We can revisit some of the old ones with these new labels, and maybe add one or two more? Any additional labeling will still be useful for training the classifier, so it's not wasted work.

Modes of analysis:

- Context, order of appearance and co-locations
- Long-term dependencies, LTL expressions
- Specific patterns, common pattern appearances

Final Push for LTL Sequences

Before we open the time for Jan of formal paper writing, we want to explore the LTL sequences and pattern identification and checking more, which will hopefully tie things together nicely.

As a reminder from the previous paper examinations, we've identified some ways that could be useful in formulating the LTL expressions to check for patterns we may care about. This method breaks down a concept into what we'll call property, and scope. Which is intuitively defined. Scope defines the sub-sequence that you look for the property in, and the property defines something that you're looking for to hold True.

There's a consideration for checking for edges and making sure things are closed under stuttering. For now, let's worry about finding relevant analysis points from what we have before we go to close the entire process.

The first important step would be to consider a type of property to look for within a defined scope that constitutes as a "pattern" in conversation that we would care about.

Originally, we talked about how there could potentially be "subconversations" (essentially patterns) of conversations that we could dial in on to get some insight on the type of conversation this sub-sequence would be (i.e: disagreement, interview, etc). I think there's definitely still some merit in doing so, and this LTL pattern checking method might be really good for that. So we can brainstorm.

Let's say we're looking for characteristics of an interview.

Interviews as a whole will have some sort of introduction, some questions, and some form of closing. So I guess on a large pseudo-global scale, we're looking at whether or not the beginning and closing has use.social.convention.

Question here is, does this show that we need to think more about the relative positioning of a label (as in relative to center?) Additionally, this also brings up the point of perhaps isolating and identifying different speakers (which at this point definitely looks like a section for future works).

But let's continue. So at a pseudo-global scale, we're looking at whether or not the beginning and closing has use.social.convention.

So that's something along the lines of

scope:Global, existence:W(use.social.convention, use.social.convention).

And we'll probably find the areas that those patterns are true, and then looking at such might be useful (Is there a way we can visualize this using Python in the same way that the paper does? This might actually make it a very simple yet effective analysis tool).

Additionally, we can branch, and check the questions that are asked

scope:W((open.question or closed.question), (respond.agree or respond.deny or deflection))

There's definitely something in the scope of between a given question and an answer that we may want to analyze. For instance, perhaps we can consider how long the length of the subsequence is between the two. Presumably for most of our interview-based stuff, from what I remember, question and response is generally almost instant.

Perhaps there's something interesting in noting in "theory" some of the potential patterns (i.e: how would I define a rhetorical question using scope and property? How would I define an argumentative subsequence, etc) and I guess can I? If there is some sort of way to find areas close to (i.e: we can define the scope, but the property doesn't necessarily fit) can we have a way to flag it as an "area of interest" (which again brings up the interest of having a visual, which might be nice).

Additionally, there's also no reason why as part of the property aspect, we can't add our addendum of Until-N variation, which would additionally be helpful to consider.

Additionally, I think that this method can also be used to check for simple properties you want the sequence to hold (i.e: scope:Global property:(open.question or closed.question)Precedes(respond.agree, respond.deny, deflection)) and if something like that is violated, then we can afford to not use the trace as part of our process analysis/modeling (or have a threshold as necessary).

Maybe this would be a nice table for us to start brainstorming ideas for patterns

Pattern	Scope	Property
interview	Global	W(use.social.convention, use.social.convention)
questioning	W((open.question or closed.question), (respond.agree or	???

	respond.deny or deflection))	
Longest unanswered question?	(open.question or closed.question) U (respond.agree or respond.deny)	???
Answers to non-existent questions	Global	(open.question or closed.question)Precedes(respond.agree, respond.deny)
Unprompted opinions	Global?	(give.opinion)Precedes(open.question or give.statement)

Interruption in Recall?	Global?	(Recall)Until(~Recall)
Good time to ask questions?	W(relax.atmosphere, relax.atmosphere)	existence(open.question or closed.question)

I guess in a sense we can take patterns like these and visualize them, but also analytically count them, which then we can generate conclusions from (i.e: unprompted opinions pattern too high for the length of the conversation, if we had a lot of traces that we made decisions about whether they were "successful" or not, then we can raise a percentage to whether or not this contributes to that. Much similar to the paper on japanese rugby plays).
Additionally, I guess the trend here is that we can go from having a scope, to another scope, to finally the property we're looking for. When we change from looking at a broader scope to a more defined specific scope, we get more potentially uncertain properties to evaluate. This is where analytics and visualizations might help?

Moving away from visualization for the moment, the concept that we can take these patterns and use them for analysis is probably something good to focus on and demonstrate. The end-goal here is to have lots of different traces lets say, and this type of analysis would give us some analytical comparison points if we archetypally classify the traces or deem them "successful" conversations or not, or have any sort of metric as a baseline. These patterns will most likely prove useful in, again, the concept of establishing a "distance" or metric of comparison.
Though I guess in this part, we are not exactly sure how we should compare the patterns, rather that we know we can find them.
So perhaps the next step is to consider, let's say we have a bunch of traces, and some handful of patterns we know we can find analytically, how can we use these to establish a notion of distance, or make a certain conclusion about two traces in comparison??
I guess we can always do the traditional TF-IDF and cosine similarity, but in this case, it seems like the analytics actually tells us a little more, since specific patterns like, "There are a lot of unprompted opinions" may signify something beyond just, "comparing X trace to Y trace" but rather, "these patterns may say something about X trace by itself." Perhaps even in those cases, we're actually still comparing X trace against a theoretical trace in our head? Like, we only assume that certain patterns tell us about X trace because we would make those conclusions about a theoretical trace Z that we have already considered to have a lot of "unprompted opinions". In this case I guess clustering might be interesting to think about.
Additionally, I guess checking traces in this manner in terms of writing "rules" that we think should make sense of traces following the fact that they are conversations, i.e: questions are asked before answers are given, etc… These "rules" can then be used to check automatically labeled traces, and could potentially again increase the quality of traces that we get, and perhaps overall modeling that follows.
I guess in terms of a unified front, we've presented multiple things that can take the notion of thinking about conversations as a process rather than embeddings/numerical representations that could be beneficial to exploration and usage of collected data (and fixing collected data) to improve understandings of

conversation for generation, analysis, checking, modeling, etc, that can improve language learning (human or ML).