Hierarchy Builder:
Organizing Textual Spans into a Hierarchy to Facilitate Navigation

Itay Yair[†] and **Hillel Taub-Tabib**[‡] and **Yoav Goldberg**[†,‡]
[†]Computer Science Department, Bar Ilan University
[‡]Allen Institute for Artificial Intelligence

Abstract

Information extraction systems often produce hundreds to thousands of strings on a specific topic. We present a method that facilitates better consumption of these strings, in an exploratory setting in which a user wants to both get a broad overview of what's available, and a chance to dive deeper on some aspects. The system works by grouping similar items together, and arranging the remaining items into a hierarchical navigable DAG structure. We apply the method to medical information extraction.

1 Introduction

We are dealing with the question of organising and displaying a large collection of related textual strings. The need arises, for example, in information extraction or text mining applications, that extract strings from text. Consider a system that scans the scientific literature and extracts possible causes for a given medical condition. Such a system may extract thousands of different strings, some of them relate to each other in various ways,[1] and some are distinct. Users consume the list in an exploratory mode (Agarwal and Sahu, 2021)(White and Roth, 2008), in which they do not have a clear picture of what they are looking for, and would like to get an overview of the different facets in the results, as well as to dig deeper into some of them.

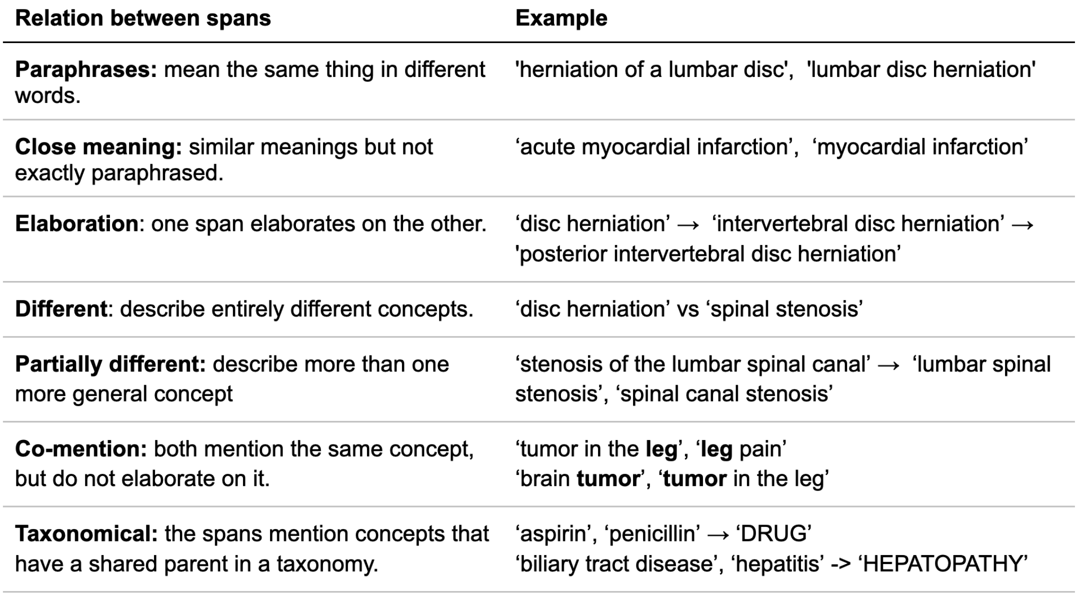
For example, distinct strings extracted as causes for sciatica include "*herniated disc*", "*herniated disk*", "*lumbar disk herniation*" , "*posterior interverbal disc herniation*" and "*endometriosis*", among hundreds of others. The user of this system likes to go over the returned list to learn about possible causes, but going over hundreds to thousands of results is mentally taxing, and we would like to reduce this effort. In the current case, we would certainly like to treat the first two items (*herniated disc* and *herniated disk*) as equivalent and show them as one unified entry. But we would also like to induce an additional hierarchy. For example, it could be useful to separate all the *herniated disc* related items (or even all the *disc* related items) in one branch, and the *endometriosis* case in another. This will allow the user to more efficiently get a high level overview of the high-level represented topics (*disc herniation* and *endometriosis*) and to navigate the results and focus on the cases that interest them in the context of the query (for example, they may feel they know a lot about disc-related causes, and choose to ignore this branch).

An additional complication is that the hierarchy we are considering is often not a tree: a single item may have two different parents, resulting in a direct acyclic graph (DAG). For example, arguably a condition like *leg pain* should be indexed both under *leg* (together with other leg related items) and under *pain* (together with pain related items). The hierarchy structure is contextual, and depends on the data: if there are not many other leg related items, it may not be beneficial to introduce this category into the hierarchy.

Additionally, note that some items in the hierarchy may not directly correspond to input strings: first, for the "*leg pain*" example above, if the input list does not include stand-alone *leg* or *pain* items, we may still introduce them in our hierarchy. We may also introduce additional abstraction, for example we may want to group "*heart disease*", "*ischemia*", "*hypotension*", and "*bleeding*" under "*cardiovascular disease*".

In this work we introduce a system that takes such a flat list of related strings, and arranges them in a navigable DAG structure, allowing users to get a high level overview as well as to navigate from general topics or concepts to more specific content by drilling down through the graph. Ideally, the graph would allow the user to:
(1) get a comprehensive overview of the the various facets reflected in the results;

[1] Figure 1 lists the kinds of relations between strings.



Relation between spans	Example
Paraphrases: mean the same thing in different words.	'herniation of a lumbar disc', 'lumbar disc herniation'
Close meaning: similar meanings but not exactly paraphrased.	‘acute myocardial infarction’, ‘myocardial infarction’
Elaboration: one span elaborates on the other.	‘disc herniation’ → ‘intervertebral disc herniation’ → 'posterior intervertebral disc herniation’
Different: describe entirely different concepts.	‘disc herniation’ vs ‘spinal stenosis’
Partially different: describe more than one more general concept	‘stenosis of the lumbar spinal canal’ → ‘lumbar spinal stenosis’, ‘spinal canal stenosis’
Co-mention: both mention the same concept, but do not elaborate on it.	‘tumor in the **leg**’, ‘**leg** pain’ ‘brain **tumor**’, ‘**tumor** in the leg’
Taxonomical: the spans mention concepts that have a shared parent in a taxonomy.	‘aspirin’, ‘penicillin’ → ‘DRUG’ ‘biliary tract disease’, ‘hepatitis’ -> ‘HEPATOPATHY’

Figure 1: Kinds of possible relations between input strings

(2) quickly get an overview of main aspects in the results;
(3) efficiently navigate the results, finding items in the sub-graph in which they expect to find them.

At a high level, the system works by finding lexically equivalent terms, arranging them in a DAG structure reflecting the specificity relation between terms, further merging equivalent nodes based on a neural similarity model, add additional potential intermediary hierarchy nodes based on taxonomic information and other heuristics, and then pruning it back into a smaller sub-DAG that contains all the initial nodes (input strings) but only a subset of the additional hierarchy nodes. Finally, we select the top-k “entry points” to this graph: high level nodes that span as many of the input nodes as possible. This process is described in section §3. While the DAG extended with potential hierarchies is very permissive and contains a lot of potentially redundant information, the DAG pruning stage aims to ensure the final graph is as compact and informative as possible.

We focus on causes-for-medical-conditions queries, and provide a demo in which a user can select a medical condition, and browse its causes in a compact DAG structure.

To evaluate the resulting DAGs, we perform automatic and manual evaluation. The automatic evaluation is based on measuring various graph metrics. The human evaluation is performed by human domain experts. Our results show that the DAG structure is significantly more informative and effective than a frequency-ranked flat list of results.

2 Requirements

As discussed in the introduction, our input is a list of strings that reflect answers to a particular question, as extracted for a large text collection (we focus in this paper on the biomedical domain, and more specifically in causes for medical conditions). This list can be the output of an Open-IE system (Fader et al., 2011; Stanovsky et al., 2015; Kolluru et al., 2020), the results of running extractive QA (Rajpurkar et al., 2016) with the same question over many paragraphs, or extracted using an extractive query in a system like SPIKE (Shlain et al., 2020; Taub Tabib et al., 2020; Ravfogel et al., 2021). The lists we consider typically contain from hundreds to thousands of unique items. We identified a set of relations that can hold between strings in our inputs, which are summarized in Table 1. We would like to arrange these items in a hierarchical structure to facilitate exploration of the result list by a user, and allow them to effectively consume the results. Concretely, the user needs to:

a. not see redundant information.

b. be able to get a high-level overview of the various answers that reflected from the results.

c. be able to get a quick access to the main answers.

d. be able to dig-in into a specific phenomenon or concept that is of interest to them.

e. be able to locate concepts they suspect that exist.

This suggests a hierarchy that respects the following conditions:
Paraphrased spans should be combined into a single group, and *close-meaning* spans should be combined into the same group; *Elaboration* relations should be expressed hierarchically; *Co-mention* spans should be both descendants of the shared concept; *Taxonomic relations* should (in some cases) be descendants of the taxonomical parent.

Additionally, we would like each node in the hierarchy to have relatively few children (to reduce the need to scan irrelevant items), yet keep the hierarchy relatively shallow (to save expansion clicks if possible). The hierarchical structure should also be informative: we should be able to guess from a given node which kinds of items to expect to find under it, and which kinds of items *not* to expect to find under it. This means a single item should be lockable in different ways, in case it can be categorized under different keys (we would sometimes like "*brain tumor*" to be listed under *brain* and sometimes under *tumors*).[2]

3 Method

Expanding the initial list. We assume that the strings in the initial list are *maximal*, meaning that the string captures the extracted noun-phrase including all of its possible modifiers. We further expand the list by considering also potential substrings of each maximal string, reflecting different granularities. For example, from the string "severe pain in the lower right leg" we would extract "pain", "severe pain" , "severe pain in the leg", "severe pain in the lower right leg", among others.[3] We then consider the union of the initial set of input strings and the set of additional sub-strings. Different users would be interested in different granularities depending on their information need. We rely on the DAG-pruning stage to properly organize these strings and prune away non-informative ones in the context of the entire set.

Initial grouping into equivalence sets. The input of this stage is a set of strings (the union of the input set and the extended set), and the output is a list of sets, such that the sets are distinct, and their union covers the initial set. For example, after this stage, the items "*herniated disk*", "*herniated disc*", "*disc herniation*", "*herniation of the disc*" will be in the same equivalence set.

The grouping in this stage is inspired by (Gashteovski et al., 2017) and is based on considering each string as a bag of lemmas, discarding stop words, modal words, and quantity words, and considering items as equivalent if their bags are equivalent. The lemma matching is relaxed, and allows, beyond exact string match, also matches with small edit distance and matches based on UMLS (Bodenreider, 2004) and WordNet (Miller, 1992) spelling variants and synonyms.

Initial DAG construction. We now take the list of sets from the previous stage, and arrange them into a DAG, where each set is a DAG node. We add a directed edge between two nodes A and B if B *is more specific than* A, and no other node C is more specific than A and less specific than B.

The *specificity relation* at this stage is determined based on the bags of lemmas that were used to create the equivalence sets: a set B is more specific than a set A if A and B are not equivalent and the bag of B contains the bag of A.

Adding heads as nodes For all spans, we take their head-word (either a single adjective or a single noun) and add them as roots of the DAG. We then add an additional root node above them, so that the DAG has a single root. This handles the co-mention relation.

Merging semantically equivalent graph nodes. We now take the DAG and merge equivalent nodes, as determined by a trained statistical model (we use SAP-BERT (Liu et al., 2020))[4]. For example, this stage will merge "*administration of streptozotocin*" and "*streptozotocin injection*". When merging two graph nodes, we handle the corresponding edges in the expected way (the children of the two individual nodes become children of the merged node, and the parents of the individual nodes become the parents of the merged node).[5]

[2] Arranging information as graphs to facilitate navigation and exploration is, of course, not a novel concept. A notable examples is entailment graphs (Kotlerman et al., 2015; Adler et al., 2012).

[3] This is done using a rules-based algorithm that operated on the parse tree, which extracted all the distinct modification spans derived from the head token.

[4] We chose SAP-BERT for its entity-linking specialization, and since it outperformed other models we tried, such as SciBert(Beltagy et al., 2019), in detecting semantic similarity for our specific case.

[5] We perform this stage after the DAG construction and not prior to it, as it makes the specificity relation between nodes significantly harder to define. In the current order, we first define specificity based on lexical containment, and then add further merge the groups.

For a pair of graph nodes A and B, we encode each string in A and in B into a vector using SAP-BERT, and represent each node as the average vector of the strings within it. We go over the nodes in the DAG in DFS order starting from the root nodes, and for each node consider all of its children for potential merging among them. We merge two nodes if the cosine similarity score between their vectors passes the threshold $t_1 = 0.9$ and their merging does not create a cycle. We then do another pass and merge nodes to direct child nodes if their similarity score is above $t_2 = 0.95$, again avoiding creating circles.

After this stage, we attempt to further merge nodes based on the UMLS ontology (Bodenreider, 2004). Two nodes A and B are considered UMLS-equivalent, if there is at least one string in node A that is listed in UMLS as a synonym of at least one string in node B. Such cases are merged.[6]

Adding taxonomic nodes. So far the relationships between nodes in the DAG were solely based on lexical relations. In order to enrich the graph, we introduce additional nodes based on taxonomical relations, which are not reliant on lexical information. For instance, "heart disease", "ischemia", "hypotension", and "bleeding" are under the broader term "cardiovascular disease". We add many nodes here, relying on many of them to be pruned in the next stage.

We map each node to the UMLS hierarchy, and look for UMLS concepts that govern at least two DAG nodes ("descendent DAG nodes"). These are potential abstractions over graph nodes. For each such UMLS concepts that is already part of the DAG, it is connected by an edge to all its descendant DAG nodes that do not already have a path to them, if adding such an edge does not create a cycle. For UMLS concepts that are not already in the DAG, they are added as new nodes governing the descendant graph nodes. UMLS concepts have multiple synonyms. When adding them as nodes, we choose the synonym with the highest SAP-BERT cosine similarity to the descendent DAG nodes this concept governs.

DAG Pruning. The DAG at this stage is quite large and messy, containing both nodes containing input strings, as well as additional hierarchy nodes based on linguistically motivated substrings of the input strings, and on taxonomic relations. We prune it to create a smaller graph which is more amenable to navigation. The smaller DAG should contain all the nodes corresponding to input strings, and an effective set of additional hierarchy nodes. Some of the hierarchy nodes are more important than others, as they provide a better differential diagnosis among the answers. Our goal is to highlight these and filter out the less important ones. Operatively, we would like for each node in the graph to have the minimal number of children, such that all the input strings that were reachable from it, remain reachable from it. This focuses on hierarchy nodes that are shared among many input concepts. We first prune graph edges according to this criteria. This process result in nodes that have a single child. Such nodes are removed, and their children are attached to their parent.[7] Selecting the minimal number of children according to this criteria is NP-hard. As an alternative, we use an approximation algorithm called the greedy set cover algorithm (Johnson, 1973), which works by selecting in each step the node with the highest number of non-covered answers, covering them, and proceeding. This helps in choosing the most important concepts and with the highest differential diagnosis.

Entry-point selection. Finally, we seek k nodes that will serve as the "entry nodes" to the graph. These should be k nodes that fulfill the following criteria:
a. allow reaching as many input strings as possible.
b. the semantic affinity between a node and the input string reachable by it, is high.

The users will initially see these nodes as well as an additional "other" node, from which all the other input strings can be reached. The entry node labels provide an overview of the k main concepts in the list, and allow the user to both get an overview of the result as well as to drill down into parts that interest them. Criteria (b) is important to ensure that the user not only can reach the input string by navigating from an entry point, but also that it will *expect* to find this input string there.

This selection is done by a heuristic algorithm which we adapted from the Greedy+ DAG-node-selection algorithm in (Zhu et al., 2020). It first assigns each node C with a score that combines the

[6] If this merging creates a cycle, this cycle is removed.

[7] Selecting the smallest group of concepts at each hierarchy level is important for user navigation, who quickly become overwhelmed by too many nodes, making it difficult to orient themselves within the DAG.

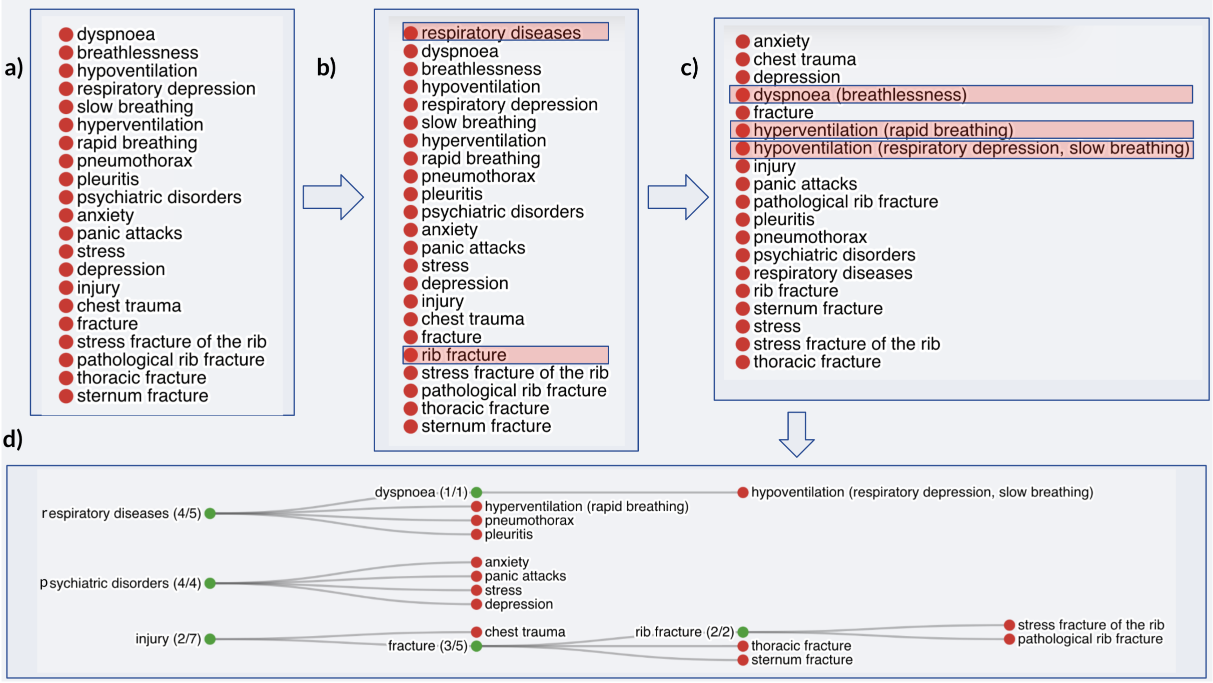


Figure 2: Input-Output Example. See section §4.

number of the input nodes reachable from it, and the semantic affinity (based on SAP-BERT cosine similarity) of C to each of these reachable nodes. It then iteratively adds the highest scoring candidate C to the set of entry points, and adjusts the scores of each remaining node N by subtracting from the score of N the affinity scores between C and the input nodes reachable from N. We do this until we reach k entry points.

Visualization. We are now finally ready to show the DAG to the user. For nodes that correspond to multiple (semantic equivalent but lexically different) input strings, we choose one of them as the representative for display purposes.

4 Input-output Example

We demonstrate with a minified example. Given the set of spans in Figure (2a), representing causes of chest pain, Hierarchy Builder expands the set by adding the spans "rib fracture" (this is a substring of two existing spans) and "respiratory diseases" (a new taxonomic node). Based on the expanded set of spans in Figure (2b), Hierarchy builder identifies synonymous spans and merges them into the concepts. In Figure (2c) we see these concepts, where each concept includes aliases in parenthesis where applicable. Hierarchy Builder then places the entries in a DAG based on a hierarchy of specificity, as depicted in Figure (2d).

5 Experiments and Evaluation

Scope We focus on the medical domain and evaluate our system on etiologies (causes) of two medical symptoms (“*jaundice*” and “*chest pain*”). These symptoms were chosen because their are common and each contain many different etiologies mentioned in the literature.

The input lists for the system were the result of running a set of 33 syntactic patterns over PubMed abstracts, looking for patterns such as “`COND due to ___`” or “`patients with COND after ___`” where COND is either *jaundice* or *chest pain*. The results were extracted using the SPIKE system (Shlain et al., 2020; Taub Tabib et al., 2020) and each matched head-word was expanded to the entire syntactic subgraph below it. This resulted in 3389 overall extracted strings and 2623 unique strings for *jaundice* and 2464 overall and 2037 unique for *chest pain*. After merging strings into synonym sets as described in §3, we remain with 2227 concepts for *jaundice* and 1783 for *chest pain*.

For each of the symptoms there are established and widely accepted lists of common etiologies, which we rely on in our evaluation.[8] We take 38 established etiologies for jaundice and 33 for chest

[8] We take the established etiologies for *jaundice* from https://www.ncbi.nlm.nih.gov/books/NBK544252/ and for *chest pain* from https://www.webmd.com/pain-management/guide/whats-causing-my-chest-pain.

pain, and check their accessability in the flat list of extracted symptoms, as well as in the hierarchical DAG we create.

Coverage and Entry-point Selection For *jaundice*, our input list contains 28 out of the 38 known etiologies, and for *chest pain* 26/33. With $k = 50$, 25 of 28 concepts are reachable from an entry point for *jaundice* and 21/26 for *chest pain*. With $k = 100$ the numbers are 28/28 (*jaundice*) and 24/26 (*chest pain*).

Assessing the contribution of the different components The different components in our algorithm contribute by adding nodes, combining nodes, adding edges, and removing edges. Table 1 describes the kind of contribution of each component and quantifies its impact, for each of the two tested conditions.

We now look at the case where we select 50 entry-point nodes, and focus on the effect on the top-level nodes. We see that for Chest-pain, a total of 20 of the 50 selected entry-points were not in the original input, but were added by the various components (12 from expanding the initial list, 5 from adding head words, and 3 from taxonomic words). Similarly, for Jaundice, these components added a total of 29 root nodes (out of the selected 50) that were not in the original input (17 from expanding initial list, 5 from head words and 6 from taxonomic nodes).

The "Expanding the initial list" component plays a significant role in shaping the DAG structure. In Chest Pain, 161 out of 224 internal nodes originate from the expanded list (146 from Expanding the initial list and 15 from co-mention). In Jaundice, 347 out of 423 internal nodes stem from the expanded list (333 from Expanding the initial list and 14 from co-mention). This highlights the substantial impact of this component on the DAG's structure.

The number of merges performed indicates the usefulness of the employed merging methods.

Furthermore, the set cover pruning algorithm effectively reduces the number of edges in the DAG.

Qualitative Measures For *jaundice*, our final DAG contains 2620 nodes overall and has a maximum depth of 11. With $k = 50$ The average number of leaves per entry point is 22.68 (min 0, max 600), and the average depth is 2.86 (min 0, max 9). Most importantly, each internal node has an average of 9.12 children (min 1, max 56, variance 34.91), making them highly browsable.

For *chest pain*, the trends are overall similar: our final DAG contains 2124 nodes overall and has a maximum depth of 9. With $k = 50$ The average number of leaves per entry point is 14.14 (min 1, max 175), and the average depth is 2.8 (min 0, max 7). Each internal node has an average of 4.94 children (min 1, max 53, variance 27.53).

Human evaluation. Our main evaluation centers around the effort for an expert[9] to locate the known etiologies in the resulting DAG, compared to a flat list sorted by frequency. For each of the etiologies, we ask how many entries need to be considered before finding the etiologies. For the flat list, this means how many items are read when scanning the list in order before reaching the etiology. For the DAG, we count the number of clicks (expansions of a node) starting from $k = 50$ entry points (a quantity that aligns with a reasonable threshold of entry nodes perceivable by a user) , while summing also the number of items before the expanded node in each level. Note that since we look for common etiologies rather than rare ones, we would assume a frequency-ranked list based on literature mentions would compare favorably in these measures. Nonetheless, we see a clear benefit of the DAG. We compare to conditions: an ideal condition where the user knows exactly which nodes to expand (blue in the graph), and a realistic scenario, in which the user searches for the etiologies by expanding nodes (gray in the graph).

We also perform another evaluation in which we ask the experts to rank each path to an etiology based on its quality, given the question "to what extent is this a logical path to follow in order to find the etiology", on a scale of 1 (very bad) to 5 (very good).

Results Figure 3 shows the main results for the two conditions. Despite the frequency-based ranking, many of the etiologies appear relatively low in the flat list, making them very hard to come by in this condition (orange). On the other hand, when considering the DAG, the vast majority of items a are significantly easier to locate, requiring scanning significantly fewer items. Only 3 items for jaundice and 2 for chest pain were significantly harder to locate in the DAG than in the flat list. In terms of the quality of the DAG paths associated with each

[9] We use two experts, each evaluating a different condition. The expert evaluating *jaundice* is an expert MD specializing in children's medicine. The expert evaluating *chest pain* is a PhD in biology with 38 years of biomedical research.

Component	Contribution	Chest-pain	Jaundice
Expanding the initial list (for full DAG)	Add nodes	504	893
Expanding the initial list (for DAG with 50 entry nodes)	Add nodes	158 (12 top level)	350 (17 top level)
Adding heads as nodes (Full DAG)	Add nodes	457	379
Adding heads as nodes (50 entry nodes)	Add nodes	20 (5 top level)	19 (6 top level)
Merging semantically equivalent nodes	Merge nodes	93 (out of 2556)	266 (out of 3330)
UMLS merging of synonym nodes	Merge nodes	62 (out of 2504)	99 (out of 3167)
UMLS taxonomic nodes (full DAG)	Add nodes	113	169
UMLS taxonomic nodes (50 entry nodes)	Add nodes	3	6
UMLS taxonomic edges	Add edges	140 (5 top level)	153 (3 top level)
DAG Pruning	Remove edges	2363	3209

Table 1: Quantifying the contribution of the different components.

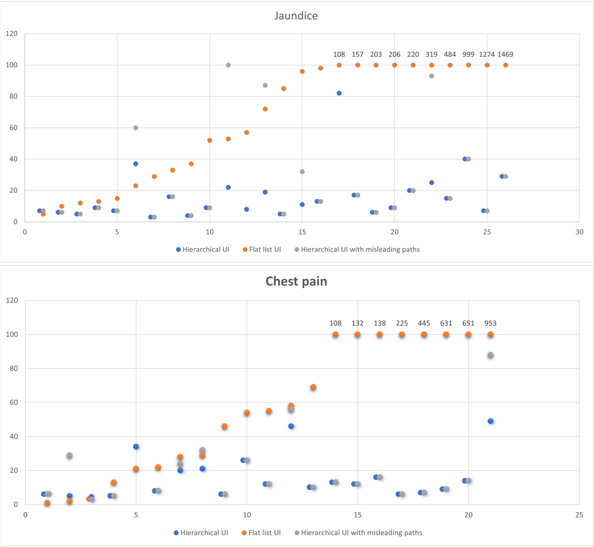


Figure 3: Effort to reach a set of common etiology items using our created DAG vs. a frequency ranked list. X axes coordinates correspond to different etiologies sorted by their frequency in the input list, and Y axes corresponds to effort. Orange: frequency-ranked flat list. Blue: DAG + oracle locating of items. Gray: DAG + human locating of items.

etiology, the jaundice annotator ranked 23 out of 25 as 5, 1 as a 2, and 1 as a 1. For chest pain, the numbers are 19 out of 21 ranked as 5, 1 as 2, and 1 as 1. Overall, our hierarchy building algorithm works well for the vast majority of the cases, and offers significant benefits over the flat list.

6 Conclusions

We presented an automatic method to organize large lists of extracted terms (here, of medical etiologies) into a navigable, DAG-based hierarchy, where the initial layer provides a good overview of the different facets in the data, and each internal node is has relatively few items. The code together with a video and an online demonstration are available at https://github.com/itayair/hierarchybuilder.

7 Limitations

While our method is aimed at organizing any flat-list of extractions, we evaluated it here only on the medical domain, only on a single kind of information need (etiologies), and only for common conditions (jaundice and chest pain). More extensive evaluation over additional conditions is needed in order to establish general-purpose utility. However, we do find the system useful for navigating in automatically-extracted etiology lists, and encourage the readers to experiment with the system also on other conditions, to assess its utility.

There are also some candidates for improving the method also in the biomedical domain, which are not currently handled: (a) abstraction over sub-strings. e.g., for the spans "*administration of penicillin*", "*administration of aspirin*", "*administration of augmentin*", it could be useful to introduce an shared parent level of "*administration of antibiotic/drug*". Our system can currently identify *penicillin*, *augmentin*, *aspirin* as an *antibiiotic/drug*, but cannot handle abstraction over sub-strings. (b) Linking to UMLS currently relies on exact lexical matches, and can be improved.

8 Ethical Considerations

We present a system for organizing large result lists into a browsable hierarchy. In general, consuming a hierarchy is more effective than consuming a very long list. However, hierarchies can hide items, especially if the items are misplaced in an unexpected branch—which our system sometimes does (albeit rarely). In situations where consuming the entire information is crucial and the cost of missing an item is prohibitive or dangerous, a flat list would be the safer choice.

Acknowledgements This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 802774 (iEXTRACT).

References

Meni Adler, Jonathan Berant, and Ido Dagan. 2012. Entailment-based text exploration with application to the health-care domain. In *Proceedings of the ACL 2012 System Demonstrations*, pages 79–84, Jeju Island, Korea. Association for Computational Linguistics.

Manoj K Agarwal and Tezan Sahu. 2021. Lookup or exploratory: What is your search intent? *arXiv preprint arXiv:2110.04640*.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Olivier Bodenreider. 2004. The unified medical language system (umls): integrating biomedical terminology. *Nucleic Acids Research*, pages D267–D270.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Kiril Gashteovski, Rainer Gemulla, and Luciano del Corro. 2017. MinIE: Minimizing facts in open information extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2630–2640, Copenhagen, Denmark. Association for Computational Linguistics.

David S Johnson. 1973. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49.

Keshav Kolluru, Vaibhav Adlakha, Samarth Aggarwal, Mausam, and Soumen Chakrabarti. 2020. OpenIE6: Iterative Grid Labeling and Coordination Analysis for Open Information Extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3748–3761, Online. Association for Computational Linguistics.

Lili Kotlerman, Ido Dagan, Bernardo Magnini, and Luisa Bentivogli. 2015. Textual entailment graphs. *Natural Language Engineering*, 21(5):699–724.

Fangyu Liu, Ehsan Shareghi, Zaiqiao Meng, Marco Basaldella, and Nigel Collier. 2020. Self-alignment pretraining for biomedical entity representations. *arXiv preprint arXiv:2010.11784*.

George A. Miller. 1992. WordNet: A lexical database for English. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Shauli Ravfogel, Hillel Taub-Tabib, and Yoav Goldberg. 2021. Neural extractive search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 210–217, Online. Association for Computational Linguistics.

Micah Shlain, Hillel Taub-Tabib, Shoval Sadde, and Yoav Goldberg. 2020. Syntactic search by example. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 17–23, Online. Association for Computational Linguistics.

Gabriel Stanovsky, Ido Dagan, and Mausam. 2015. Open IE as an intermediate structure for semantic tasks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 303–308, Beijing, China. Association for Computational Linguistics.

Hillel Taub Tabib, Micah Shlain, Shoval Sadde, Dan Lahav, Matan Eyal, Yaara Cohen, and Yoav Goldberg. 2020. Interactive extractive search over biomedical corpora. In *Proceedings of the 19th SIGBioMed Workshop on Biomedical Language Processing*, pages 28–37, Online. Association for Computational Linguistics.

Ryen W White and Resa A Roth. 2008. Evaluation of exploratory search systems. In *Exploratory Search: Beyond the Query—Response Paradigm*, pages 61–69. Springer.

Xuliang Zhu, Xin Huang, Byron Choi, and Jianliang Xu. 2020. Top-k graph summarization on hierarchical dags. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1903–1912.