Supplemental Material

Corpus Pre-processing

We use two NLP packages to pre-process our corpus, OpenNLP [1] and the Stanford Parser [1]. We use the sentence breaker from OpenNLP, which applies a maximum entropy model. After sentence breaking, we use a home-built rule-based tokenizer that respects domain specific tokens such as "CD4+" or "TdT+" as one token. Following the approach of [2], we use the UMLS Specialist Lexicon (which contains lexical descriptions of over 1.1 million words) to build an extended lexicon by mapping UMLS style part-of-speech tags and linguistic features such as *pleural* and *present singular* to Penn Treebank tags [3]. Unlike [2], we add the extended lexicon into OpenNLP's POS tagger dictionary. This is straightforward because the OpenNLP tagger enumerates possible tags only from its dictionary and then evaluates their likelihood.

Matching Token Subsequences to UMLS Concepts

To group token subsequences that are medical terminology, we perform dictionary look up against the UMLS Metathesaurus [4]. We investigate each of the $n \times (n-1)$ subsequences of tokens in a sentence and look it up in the UMLS Metathesaurus. For the UMLS CUI matching, we experimented with the entire set or subsets of CUIs and chose the following approach that balances the coverage and accuracy on our data. If the token subsequence has only one CUI match, this CUI is used. If the token subsequence has multiple CUI matches, we select the one that is confirmed by the most number of sources. If there is a tie, we prefer the CUI supported by SNOMED CT [5] if there is one, or flip a coin otherwise. We then perform a greedy search to find the longest token subsequences with a matching UMLS concept unique identifier (CUI). Employed heuristics to guide the greedy search include ignoring case in matching, eliminating subsequences that are fully contained in longer sequences, eliminating interpretations of single tokens that fall into function-word grammatical categories, and ignoring punctuations. After that, we look up multiple mapping tables in the UMLS Metathesaurus and obtain medical subject headings (MeSH), semantic type unique identifiers (TUI) from CUIs.

Two-Phase Sentence Parsing

The medical language used in pathology reports is challenging for general domain parsers. Consider the example sentence: “In situ hybridization for kappa and lambda immunoglobulin light chains show the plasma cells to be polytypic.” Figure 1 shows the parse by the Stanford Parser, in which the term “in situ hybridization” is broken and erroneous dependencies such as amod(hybridization-3, situ-2) and prep_in(show-11, hybridization-3) are generated.

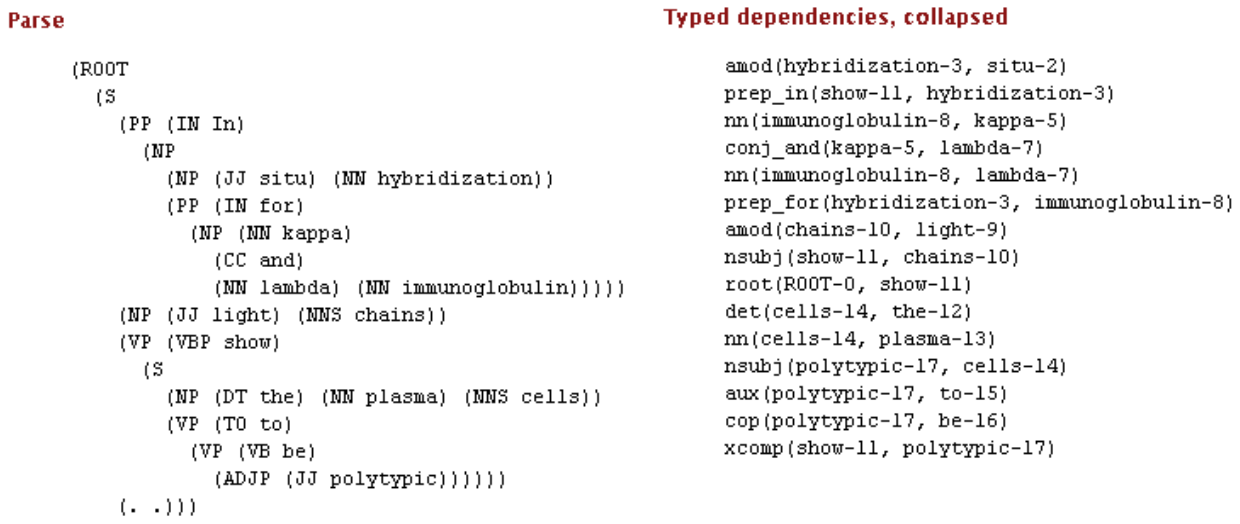

Figure 1 Example sentence parsed directly by the Stanford Parser

Knowing that “in situ hybridization” is one phrase, the parser not only corrects the error with “in situ hybridization”, but also respects the long phrase “kappa and lambda immunoglobulin light chains”, as shown in Figure 2. We therefore parse sentences in two steps: 1) we identify and group together the non-determiner tokens that match to the concept unique identifiers (CUI) in the UMLS Metathesaurus [6], 2) we then apply the Stanford Parser with grouped tokens as one token. We only group token subsequences whose last token is a noun. Finally, we assign POS tags to grouped token subsequences by using the POS tags from their last tokens during a separate run of POS tagger on the original sentence.

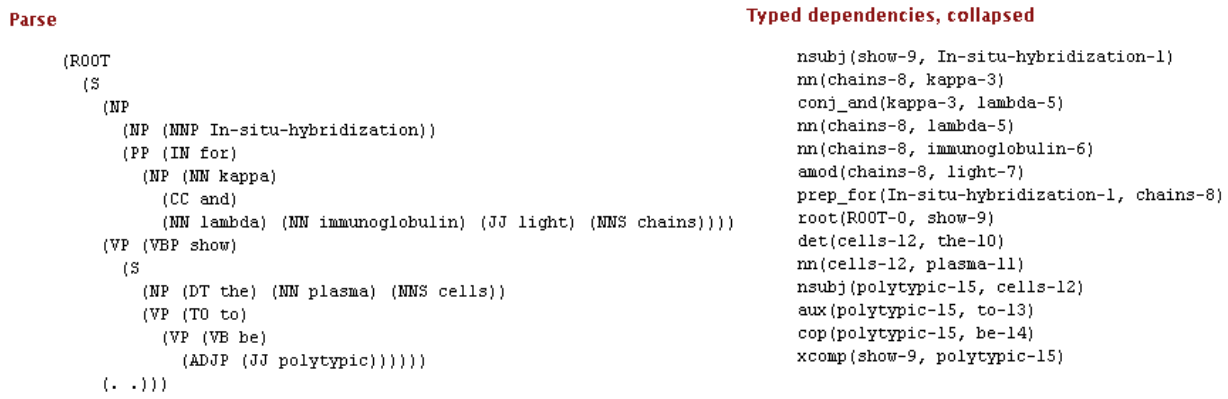

Figure 2 Two-phase sentence parsing on example

Choosing CUI over TUI to Group Token Subsequences

The relative usefulness of various dictionaries from the UMLS Metathesaurus has received mixed reports from the research community [7]. Earlier in our experiments, we initially relied on using the UMLS semantic types to group token subsequences. The UMLS currently defines 133 semantic types that are indexed by type unique identifiers (TUI). Our earlier approach followed a sequence of steps called zoom-in, mine and zoom-out. In the zoom-in step, in addition to grouping token subsequences using CUIs, we further mapped each CUI to a corresponding TUI and identified the semantic types of the grouped token subsequences. In the mining step, we treated token subsequences sharing a semantic type as identical nodes in the sentence graphs, and applied frequent subgraph mining. The rationale was to group concepts of the same semantic types together. This would lead to a coarser granularity of concepts, with the hope for the captured frequent subgraphs to cover more sentences. In the zoom-out step, we took the frequent subgraphs returned by the mining step, mapped them back to the sentences and replaced TUI labels for their nodes with corresponding CUIs extracted from those sentences.

However, we later noticed that UMLS semantic categories in general provided too coarse a granularity for our application. For example, T cells, B cells, neutrophils, and megakaryocytes all mapped to the semantic type of "Cell" at the lowest level of the UMLS semantic types. Moreover, the UMLS semantic types sometimes lead to inconsistencies with our domain knowledge. For example, if one includes all CUIs for "CD10" and maps them to semantic types, one gets the following semantic types: *molecular function*, *enzyme*, and *gene or genome*.

However, pathologists see CD10 primarily as an important *immunologic factor*. In fact, this happens for multiple other CD antigens, including CD79a (mapping to *Amino Acid, Peptide, or Protein* and *Receptor*), CD138 (mapping to *Gene or Genome*, *Amino Acid, Peptide, or Protein* and *Biologically Active Substance*) etc. Note that for CD138, strictly speaking, *Biologically Active Substance* is a semantic type subsuming *immunologic factor*. However, referring only to the semantic type hierarchy, this does not preclude the possibility that CD138 may belong to other subsumed semantic types such as *Neuroreactive Substance or Biogenic Amine*, *Hormone*, *Enzyme*, *Vitamin*, and *Receptor*. A third problem is that the UMLS semantic type hierarchy does not form a strict taxonomy. For example, under the type *chemical*, the subtypes *chemical viewed functionally* and *chemical viewed structurally* largely overlap each other. This leads to the problem that even the same CUI of a chemical can have two semantic types. Due to the above problems, we saw much noise coming from using the UMLS semantic types as node labels for sentence graph, which affected discovery of frequent subgraphs and, in turn, classification performance. We tried multiple heuristics to attempt to resolve such inconsistencies, for example, only looking at upper levels of the semantic hierarchy. However, this aggravated the coarse granularity problem and led to no obvious classification performance gain. We finally resorted to relying on the CUIs to label sentence graph nodes.

Parse Post Processing

In order to increase the accuracy of the sentence graph representations, we perform post processing on the Stanford dependency parsing results. The main observation is that a list of immunologic factors often poses parsing challenge as in the sentence, "Most interstitial lymphocytes are CD3 positive T-cells with fewer CD20 and PAX5 positive B-cells". Even if all POS tags are correctly assigned, the parser still has difficulty in determining that "CD20" and "PAX5" are both connected to "positive". We observed the following list patterns that may interfere with the parsing process and implemented rule-based post-processing systems to systematically correct list related errors. For each pattern, we give an example sentence along with its Stanford Parsing results with and without pre-processing.

1. A list of nominal immunological factors:

   Example sentence 1: "These large cells are positive for the B-cell markers CD20, OCT2, BOB1 and are also MUM1 and BCL6 positive."

Figure 3 shows the raw Stanford parsing result. Figure 4 shows the parsing results after pre-processing on tokenization and POS tags. It is clear that pre-processing helps on correcting the POS tags for "MUM1" and "BCL6". However dependencies involving "OCT2" and "BOB1" are incorrect as highlighted in Figure 4.

```
(ROOT
  (S
    (NP (DT These) (JJ large) (NNS cells))
    (VP
      (VP (VBP are)
        (ADJP (JJ positive)
          (PP (IN for)
            (NP
              (NP (DT the) (JJ B-cell) (NNS markers) (NN CD20))
              (, ,)
              (NP (NNP OCT2) (, ,) (NNP BOB1))))))
      (CC and)
      (VP (VBP are)
        (ADVP (RB also))
        (ADJP
          (ADJP (JJ MUM1))
          (CC and)
          (ADJP (RB BCL6) (JJ positive)))))
    (. .)))
```

```
det(cells-3, These-1)
amod(cells-3, large-2)
nsubj(positive-5, cells-3)
nsubj(MUM1-18, cells-3)
cop(positive-5, are-4)
root(ROOT-0, positive-5)
det(CD20-10, the-7)
amod(CD20-10, B-cell-8)
nn(CD20-10, markers-9)
prep_for(positive-5, CD20-10)
nn(BOB1-14, OCT2-12)
appos(CD20-10, BOB1-14)
cop(MUM1-18, are-16)
advmod(MUM1-18, also-17)
conj_and(positive-5, MUM1-18)
advmod(positive-21, BCL6-20)
conj_and(positive-5, positive-21)
conj_and(MUM1-18, positive-21)
```

Figure 3 Raw Stanford parsing result for example sentence 1

```
(ROOT
  (S
    (NP (DT These) (JJ large) (NNS cells))
    (VP
      (VP (VBP are)
        (ADJP (JJ positive)
          (PP (IN for)
            (NP
              (NP (DT the) (NNS B-cell-markers) (NN CD20))
              (, ,)
              (NP
                (NP (NN OCT2))
                (, ,)
                (NP (NN BOB1)))))))
      (CC and)
      (VP (VBP are)
        (ADVP (RB also))
        (NP (NN MUM1)
          (CC and)
          (NN BCL6)))
      (ADJP (JJ positive)))
    (. .)))
```

```
det(cells-3, These-1)
amod(cells-3, large-2)
nsubj(positive-5, cells-3)
cop(positive-5, are-4)
root(ROOT-0, positive-5)
det(CD20-9, the-7)
nn(CD20-9, B-cell-markers-8)
prep_for(positive-5, CD20-9)
appos(CD20-9, OCT2-11)
appos(OCT2-11, BOB1-13)
cop(MUM1-17, are-15)
advmod(MUM1-17, also-16)
conj_and(positive-5, MUM1-17)
conj_and(positive-5, BCL6-19)
conj_and(MUM1-17, BCL6-19)
acomp(positive-5, positive-20)
```

Figure 4 Stanford parsing result after pre-processing for example sentence 1

2. A list of adjective form immunological factors:

Example sentence 2: “Report of immunostains indicates the cells are CD79a+, CD20+, CD3-, CD5-, BCl6+, BCL2-, and CD10+ consistent with follicle center origin.”

Figure 5 shows the raw parsing result, in which many tokenizations, POS tags and dependencies are incorrect. Figure 6 shows the parsing result after pre-processing. Improvements on tokenizations and POS tags are seen, but dependency errors are still present as highlighted.

```
(ROOT
  (S
    (NP
      (NP (NNP Report))
      (PP (IN of)
        (NP (NNS immunostains))))
    (VP (VBZ indicates)
      (SBAR
        (S
          (NP (DT the) (NNS cells))
          (VP (VBP are)
            (NP
              (NP
                (NP (NNP CD79a) (NNP +))
                (, ,)
                (NP (NNP CD20) (CD +) (, ,) (CD CD3))
                (: -))
              (, ,)
              (NP
                (NP (NNP CD5))
                (PRN (: -)
                  (NP
                    (NP (, ,) (JJ BCl6) (NN +))
                    (, ,)
                    (NP (NNP BCL2)))
                  (: -)))
              (, ,)
              (CC and)
              (NP
                (NP (NNP CD10) (NNP +))
                (ADJP (JJ consistent)
                  (PP (IN with)
                    (NP (JJ follicle) (NN center) (NN origin)))))))))
    (. .)))
```

```
nsubj(indicates-4, Report-1)
prep_of(Report-1, immunostains-3)
root(ROOT-0, indicates-4)
det(cells-6, the-5)
nsubj(+-9, cells-6)
cop(+-9, are-7)
nn(+-9, CD79a-8)
ccomp(indicates-4, +-9)
appos(+-9, CD20-11)
num(CD20-11, +-12)
num(CD20-11, CD3-14)
ccomp(indicates-4, CD5-17)
conj_and(+-9, CD5-17)
amod(+-21, BCl6-20)
dep(CD5-17, +-21)
appos(+-21, BCL2-23)
nn(+-28, CD10-27)
ccomp(indicates-4, +-28)
conj_and(+-9, +-28)
amod(+-28, consistent-29)
amod(origin-33, follicle-31)
nn(origin-33, center-32)
prep_with(consistent-29, origin-33)
```

Figure 5 Raw Stanford parsing result for example sentence 2

```
(ROOT
  (S
    (NP
      (NP (NN Report))
      (PP (IN of)
        (NP (NNS immunostains))))
    (VP (VBZ indicates)
      (SBAR
        (S
          (NP (DT the) (NNS cells))
          (VP (VBP are)
            (NP
              (NP (JJ CD79a+))
              (, ,)
              (ADJP (JJ CD20+) (, ,) (JJ CD3-) (, ,) (JJ CD5-) (, ,) (JJ BC16+) (, ,) (JJ BCL2-) (, ,)
                (CC and)
                (JJ CD10+) (JJ consistent)))
              (PP (IN with)
                (NP (NN follicle) (NN center) (NN origin)))))))
    (. .)))
```

```
nsubj(indicates-4, Report-1)
prep_of(Report-1, immunostains-3)
root(ROOT-0, indicates-4)
det(cells-6, the-5)
nsubj(CD79a+-8, cells-6)
cop(CD79a+-8, are-7)
ccomp(indicates-4, CD79a+-8)
dep(consistent-22, CD20+-10)
amod(CD79a+-8, CD3--12)
conj_and(consistent-22, CD3--12)
amod(CD79a+-8, CD5--14)
conj_and(consistent-22, CD5--14)
amod(CD79a+-8, BC16+-16)
conj_and(consistent-22, BC16+-16)
amod(CD79a+-8, BCL2--18)
conj_and(consistent-22, BCL2--18)
amod(CD79a+-8, CD10+-21)
conj_and(consistent-22, CD10+-21)
amod(CD79a+-8, consistent-22)
nn(origin-26, follicle-24)
nn(origin-26, center-25)
prep_with(CD79a+-8, origin-26)
```

Figure 6 Stanford parsing result after pre-processing for example sentence 2

3. A list of nominal immunological factors modifying adjectives:

   Example sentence 3: “Most interstitial lymphocytes are CD3 positive T-cells with fewer CD20 and PAX5 positive B-cells.”

   Figure 7 shows the raw parsing result with POS tags errors such as for “CD3”. Figure 8 shows the parsing result with pre-processing. Highlighted parts indicate the error in not recognizing that “B-cells” are “CD20” “positive”.

```
(ROOT
  (S
    (S
      (NP
        (NP (NNP Immunohistochemistry))
        (PP (IN of)
          (NP (DT the) (NN bone) (NN marrow) (NN core))))
      (VP (VBZ reveals)
        (SBAR (IN that)
          (S
            (NP (RBS most) (JJ interstitial) (NNS lymphocytes))
            (VP (VBP are)
              (VP (VBG CD3)
                (NP (JJ positive) (NNS T-cells))
                (PP (IN with)
                  (NP
                    (NP (JJR fewer) (NN CD20))
                    (CC and)
                    (NP (CD PAX5) (JJ positive) (NNS B-cells)))))))))
    (: ;)
    (S
      (NP (DT the) (NN latter))
      (VP (VBP are)
        (ADJP (JJ small)
          (PP (IN in)
            (NP (NN size))))))
    (. .)))
```

```
nsubj(reveals-7, Immunohistochemistry-1)
det(core-6, the-3)
nn(core-6, bone-4)
nn(core-6, marrow-5)
prep_of(Immunohistochemistry-1, core-6)
root(ROOT-0, reveals-7)
mark(CD3-13, that-8)
advmod(lymphocytes-11, most-9)
amod(lymphocytes-11, interstitial-10)
nsubj(CD3-13, lymphocytes-11)
aux(CD3-13, are-12)
ccomp(reveals-7, CD3-13)
amod(T-cells-15, positive-14)
dobj(CD3-13, T-cells-15)
amod(CD20-18, fewer-17)
prep_with(CD3-13, CD20-18)
num(B-cells-22, PAX5-20)
amod(B-cells-22, positive-21)
prep_with(CD3-13, B-cells-22)
conj_and(CD20-18, B-cells-22)
det(latter-25, the-24)
nsubj(small-27, latter-25)
cop(small-27, are-26)
parataxis(reveals-7, small-27)
prep_in(small-27, size-29)
```

Figure 7 Raw Stanford parsing result for example sentence 3

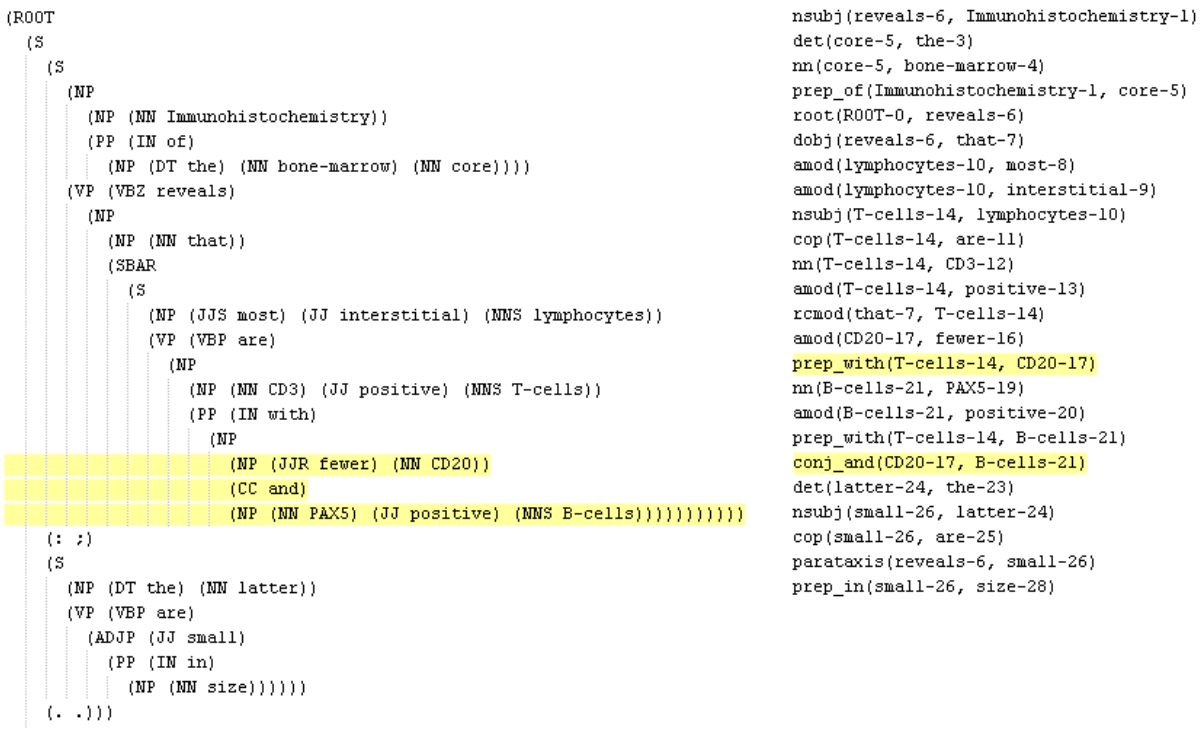

Figure 8 Stanford parsing result after pre-processing for example sentence 3

To correct the parsing errors introduced by the above list patterns, we perform the following steps. We first recognize the immunologic list patterns by checking the UMLS semantic types of parsing nodes and record those belonging to immunologic factors. The semantic types along with their specific TUI numbers that are considered as immunologic factors are shown in Table 1. Multiple semantic types are included because some cell surface markers may belong to one or more semantic types. For example, "CD2" belongs to "Amino Acid, Peptide, or Protein", "Immunologic Factor", "Receptor", "CD10" belongs to "Enzyme", "CD138" belongs to "Amino Acid, Peptide, or Protein", "Biologically Active Substance", "BCL2" belongs to "Gene or Genome" and "EBV" belongs to "Virus". After recognizing such list patterns, we check the POS tags of immunologic factor parse nodes. If they are adjectives (pattern 2), we replace the whole list with a dummy adjective "atypical". If they are nouns, and if the list is followed by an adjective (pattern 3), we replace the whole list and the following adjective with a dummy adjective "atypical". If the list is not followed by an adjective, we replace the whole list by a dummy proper noun "ATG"[1]. We then parse those modified sentences using Stanford Parser. At last, we fill back the immunologic factors in the original list. For pattern 2, we copy the

[1] We use a dummy proper noun so that they can fit in sentences with either singular form or plural form predicates.

dependencies of "atypical" to each immunologic factor adjectives. For pattern 3, we copy the dependencies of "atypical" to the adjective following the list and connect each immunologic factor with that adjective. For pattern 1, we copy the dependencies of "ATG" to each immunologic factor.

Table 1 Semantic types considered as immunologic factors

TUIs	Semantic Types
T123	Biologically Active Substance
T129	Immunologic Factor
T192	Receptor
T116	Amino Acid, Peptide, or Protein
T126	Enzyme
T028	Gene or Genome
T005	Virus

Duplicate Removal for Frequent Subgraph Mining

We ran Gaston [8] on our training dataset having 17,186 sentences, with a frequency threshold of 5, and obtained 180,863 frequent subgraphs. Analyzing these subgraphs, we found that many smaller subgraphs are subisomorphic to other larger frequent subgraphs. Many of these larger subgraphs have the same frequencies as their subisomorphic smaller subgraphs. This arises when a larger subgraph is frequent; all its subgraphs automatically become frequent as well. Furthermore, if the smaller subgraph is so unique that it is not subisomorphic to any other larger subgraph, then this pair of larger and smaller subgraphs shares identical frequency. Therefore, we only kept the larger subgraphs in such pairs. Note that it is cost prohibitive to perform a full pairwise check because the subisomorphism comparison between two subgraphs is already NP complete [8], and a pairwise approach would ask for ~16 billion such comparisons for our dataset. We developed an efficient algorithm using hierarchical hash partitioning that reduces the number of subgraph pairs to compare by several orders of magnitude. The key idea is that we only need to compare subgraphs whose sizes differ by one, and we can further partition the subgraphs so that only those within the same partition need to be compared.

To fully explore this idea, we make the following observations. First, we only need to check subgraph pairs whose frequencies are the same (H1). Second, we only need to check subgraph pairs whose sizes in term of node numbers differ by one (H2). In fact, let G_s, G be subgraphs with G_s being subisomorphic to G, and $\#(G_s) = \#(G)$ and $|G_s| < |G| - 1$, where $\#(\cdot)$ denotes

the frequency and $|\cdot|$ denotes the number of nodes in a graph. Then given the subisomorphism between G_s and G, one can construct a G_1 by simply adding one additional node (and associated edges) in G. It is clear that $\#(G) \leq \#(G_1) \leq \#(G_s)$, but because $\#(G) = \#(G_S)$, we have $\#(G) = \#(G_1) = \#(G_s)$. Thus we only need to check subisomorphism between G_s and G_1, and between G_1 and G, where G_1 differ from G_s in size by only one. Carrying on such construction, we therefore only need to check pairs of subgraphs whose sizes differ by one. Based on the H2, we can first order subgraphs in descending order according to their sizes. Then it suffices to progress down the hierarchy, checking among subgraphs that are in the neighboring two levels.

To further reduce unnecessary subisomorphic comparisons, we make another observation that for a graph G_s to be subisomorphic to G, the node labels of G_s must be a subset of G. Moreover, as we restrict ourselves in comparing only subgraphs from neighboring levels, we are able to adopt a hash partition scheme to avoid enumerating all possible pairs from neighboring levels. Precisely speaking, at level n, a subgraph has n nodes, if we consider its $n-1$ size subgraphs, there are only n possible set of labels. We can then construct a hash table and hash the level-n subgraphs n times using their $n-1$ node label subsets as keys. We also hash subgraphs from level $n-1$ using their node label sets (size $n-1$) as key. We note that it is only necessary to check subgraph pairs in the same partition (H3). Although an upper level subgraph is hashed multiple times into the hash table, hashing has both constant amortized update time and constant amortized look up time. The time for multiple hashes is much less than the time for unnecessary subisomorphism comparisons. Moreover, in practice, the size of the subgraph is often small, and multiple hashes only multiply a constant factor to the total hash update and look up times.

A summary of our algorithm is shown in Figure 9. Lines 1 and 2 sort the set of graphs so that they are first ordered (in descending order) first by their number of nodes and then by their number of edges. This ensures that subisomorphism only needs to be checked by looking at graphs before the current one. Line 3 partition graphs into levels according to their sizes while keeping the previously sorted order. Lines 5 to 29 progress down the hierarchy performing subisomorphism check when necessary. Lines 7 to 11 hash each upper level graph into possibly multiple buckets. Lines 12 to 15 partition lower level graphs into different hash buckets. Lines 20 to 23 check subisomorphism within the same hash partition on the lower level. Lines 24 to 29 check subisomorphisms between corresponding lower level bucket and upper level buckets. In

lines 22 and 28, we generalize from the condition requiring two subgraphs to have identical frequencies to a condition customizable by the user.

```
subisomorphim_for_set_of_graphs
input:  S – set of graphs
effect: compute subisomorphism relation among graphs in S
1   stable sort S in descending order of number of edges
2   stable sort S in descending order of number of nodes
3   levels ← put graphs of size n into levels[n]
4   max_level ← length(levels)

5   for  n = max_level  downto 2
6      h_upper = {}; h_lower = {}
7      if n != max_level
8         ulevel = levels[n+1]
9         for i = 1 to length(ulevel)
10           foreach key : labels of n-1 subset of nodes of ulevel[i]
11              add ulevel[i] into the list h_upper[key]

12     llevel = levels[n]
13     for i = 1 to length(llevel)
14        foreach key : set of labels llevel[i]
15           add llevel[i] into the list h_lower[key]

16     foreach key in h_lower.keys()
17        g_lower = h_lower[key]
18        for i = 1 to length(g_lower)
19           gs = g_lower[i]
20           for j = 1 to i-1
21              gb = g_lower[j]
22              if condition = true
23                 subisomorphism(gs, gb)
24           if h_upper.has_key(key)
25              g_upper = h_upper[key]
26              for j = 1 to length(g_upper)
27                 gb = g_upper[j]
28                 if condition = true
29                    subisomorphism(gs, gb)
```

Figure 9 A hierarchical hash partition algorithm for determining subisomorphism relation among graphs in a set

Regular Expressions to Catch Lymphoma Mentions

"(?i)(burkit|burket)"

"(?i)\bBL\b"

"(?i)\bDLBCL\b"

"(?is)(follicular|follicle).*(type|origin)" // e.g. “low grade lymphoma, follicle center cell type”

"(?i)\bFL\b"

"(?i)\b(nlphl|nlphd|hl|hd)\b"

"(?i)\bNHL\b"

"(?i)hodgkin"

"(?i)lymphoma"

"(?i)leukemia"

"(?is)diffuse.*large.*b.*cell"

"(?i)T/HRBCL"

"(?is)(nodular\s+sclerosis|mixed\s+cellularity|lymphocyte-rich.*type|lymphocyte\s+predominant)" // e.g., “Hodgkin lymphoma, mixed cellularity type”

References

[1] Apache OpenNLP project team, “Apache OpenNLP @ONLINE,” Apr. 2013.

[2] Y. Huang, H.J. Lowe, D. Klein, and R.J. Cucina, “Improved identification of noun phrases in clinical radiology reports using a high-performance statistical natural language parser augmented with the UMLS specialist lexicon,” *Journal of the American Medical Informatics Association*, vol. 12, 2005, pp. 275–285.

[3] B. Santorini, “Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision),” 1990.

[4] A.R. Aronson, “Effective Mapping of Biomedical Text to the UMLS Metathesaurus: The MetaMap Program,” 2001.

[5] IHTSDO, “SNOMED CT http://www.ihtsdo.org/snomed-ct/.”

[6] D.A. Lindberg, B.L. Humphreys, A.T. McCray, and others, “The Unified Medical Language System.,” *Methods of information in medicine*, vol. 32, 1993, p. 281.

[7] Y. Chen, H. Gu, Y. Perl, M. Halper, and J. Xu, “Expanding the extent of a UMLS semantic type via group neighborhood auditing,” *Journal of the American Medical Informatics Association*, vol. 16, 2009, pp. 746–757.

[8] S. Nijssen and J.N. Kok, “The gaston tool for frequent subgraph mining,” *Electronic Notes in Theoretical Computer Science*, vol. 127, 2005, pp. 77–87.