University of Otago

Department of Computer Science

COSC480 Project Report

Quantifying Conceptual Density in Text

Author:
Anthony Dickson
(3348967)

Supervisor(s):
Anthony Robins
Alistair Knott

July 13, 2019

Abstract

In introductory programming courses across many institutions, educators have observed an unusually high rate of both fail and high grades. One explanation for the cause of this phenomenon mentions the idea that concepts within programming languages are tightly integrated, perhaps more so than some other disciplines. In domains of tightly integrated concepts, learning a concept depends greatly on the successful learning of the prerequisite concepts. As such, learning outcomes tend to be self-reinforcing; successful learning promotes further successful learning and unsuccessful learning promotes further unsuccessful learning. This idea of a domain of concepts being tightly integrated is referred to as conceptual density. In this report I set out to define conceptual density with reference to cognitive load theory, and develop a method for quantifying conceptual density in well-structured expository text documents using natural language processing methods and graph analysis techniques.

Contents

1 Introduction

It is thought that the unusually high rate of fail and high grades observed in introductory programming courses is possibly due to concepts in programming languages being unusually tightly integrated (Robins, 2010). A tightly integrated concept is a concept that is difficult to describe independently of other concepts. Tightly integrated concepts are said to magnify the effects of learning edge momentum and lead to the polarised outcomes observed in introductory programming courses. This report is interested in conceptual density, a sort of measure concerned with the level of integration for concepts in a given domain.

The notion of this idea of conceptual density affecting learning difficulty is not unique to the aforementioned work. It appears in the literature on cognitive load theory, in a perhaps slightly different form, as the "element interactivity effect", which says that the number of interacting elements in a task is the main contributor to cognitive load - a measure of mental exertion required to perform a given task (Sweller, 1994). Cognitive load theory provides an explanation of aspects of learning difficulty and can be used to guide education and instructional design.

To the best of my knowledge there exists no objective measure in the current literature of the element interactivity effect, or the notion of conceptual density. Element interactivity appears to be estimated manually and it depends greatly on the knowledge of the learner (Chandler and Sweller, 1996), and as such remains subjective. In (Robins, 2010) the notion of concepts being tightly integrated stems from the observation that educators typically fail to reach a consensus on how to structure introductory programming courses (this is supposedly not the case in other disciplines). While no measure of conceptual density was clearly defined in that work, some possible ways to do so were proposed.

In this report, I follow up on the notion of conceptual density and set out to elaborate on its definition and develop an objective measure of conceptual density in the context of well-structured expository text documents (e.g. textbooks, scientific papers). For measuring conceptual density I explore one of the methods mentioned in (Robins, 2010) - analysing a mind-map like graph structure. Whereas in the original work, this approach was suggested as a characterisation of introductory programming course material to be generated by experts, in this report I aim to build up and analyse mind-map like graph structures automatically and systematically through the use

of natural language processing methods and graph analysis techniques.

The proposed method will be evaluated first through experiments using small hand-crafted text documents and then on a selection of expository text documents. The end goal of this report is to be able to systematically generate scores of conceptual density that are objective and in line with human judgements. Ideally, the scores would be able to be used to compare text documents and also identify which sections in a given text are conceptually dense.

2 Background

2.1 Learning Edge Momentum

Learning edge momentum was proposed as an explanation for the unusually high rate of both fail and high grades in introductory computer science courses (Robins, 2010). The hypothesis builds upon the idea that we learn at the edges of what we know - it is easy to learn things that build upon what we already know. The core of the hypothesis is that learning outcomes are self-reinforcing; successful learning helps further successful learning of closely related concepts, and unsuccessful learning tends to lead to further unsuccessful learning of closely related concepts. With successful learning one builds up positive momentum and with unsuccessful learning one builds up negative momentum. It is said that concepts in programming languages tend to be tightly integrated, that concepts are difficult to describe or understand independently of each other. This means that learning a new concept is greatly dependent on the understanding of the prerequisite concepts. And because of this the effects of learning edge momentum in introductory programming courses are particularly pronounced, leading to the polarised distribution of grades.

Conceptual density (as it is defined in this report) was mentioned in this paper as a possible way of providing more concrete evidence to educators on how to structure an introductory computer science curriculum. Mind maps are suggested as a way of representing the material in a course. Here we represent the domain of concepts as a graph structure, where the concepts present in the material are represented with a node in the graph and edges between nodes represent a connection between concepts. This follows the main ideas behind semantic networks and knowledge graphs (Sowa, 1987;

Zhang, 2002; Koncel-Kedziorski et al., 2019), an area of research concerning knowledge representation using graphs. This idea of a mind map of concepts forms the basis for the work in this paper.

2.2 Cognitive Load Theory

Cognitive load theory provides a framework about learning difficulty and instructional design (Sweller, 1994). In CLT (cognitive load theory) the main task of the brain is characterised as information processing, and each person has a certain amount of working memory to process information. Performing tasks requires an amount of mental exertion and concentration, called cognitive load. When the amount of cognitive load exceeds the capacity of an individual's working memory, performing the given task becomes much more difficult.

If we are to explain this by analogy, the idea of working memory is similar in some ways to computer RAM and running out of working memory is akin to what happens when a computer runs out of RAM. When a computer runs out of RAM, it starts thrashing and all of the programs will become very slow and take many times longer to run compared to when there is enough RAM. By reducing the number of programs that are running we can reduce the memory usage and prevent the computer from thrashing. Similarly, under CLT we can reduce cognitive load by reducing the number of things that must be considered simultaneously to perform a given task in order to make it easier to perform that task.

The main contributor to the difficulty of a task is number of things that must be considered simultaneously and the degree to which they interact. In CLT this idea is captured by the **element interactivity effect** (Sweller et al., 2011). The cognitive load imposed by element interactivity is said to be intrinsic to the material itself (**intrinsic cognitive load**), meaning that no matter how you present that material there will be a certain level of difficulty.

How we present material can affect how difficult it is understand it. For example, consider trying to teach beginner programmers what all of the keywords mean in a Java program that simply prints a message to the screen (see Figure 1) and what each part of the code does. This would likely be too much to learn at once. Here there are many interacting elements (e.g. access modifiers, classes, types, functions, function arguments, comments, member fields, member functions, function calls, string literals) just in this simple

```java
public class HelloWorld {
    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }
}
```

Figure 1: Java hello world example

program. Instructors typically make these initial lessons easier to follow by giving a simplified, high-level explanation and omitting details that are deemed unnecessary to know at that given point in the course. Changing how the material is presented has the effect of increasing or reducing **extraneous cognitive load**, cognitive load that is not intrinsic to the material itself but to how it is presented.

Once they have practised Java programming more and have learned the fundamentals well, the details can be learned more easily. This is because the learner will be more familiar with the fundamental ideas, and performing tasks with these ideas is easier due to their familiarity (it is easier to perform a task you have done many times than one you have never/just learned). It is theorised that this is because as complex ideas and tasks are learned we learn a more compact representation - a **schema**, which is essentially a single unit of knowledge representing a complex idea or task (Axelrod, 1973; Abelson, 1981; Bartlett et al., 1995).

By learning these compact representations, a task with many interacting elements may be reduced to a task with a few, or even a single, element. And under the element interactivity effect fewer interacting elements mean less cognitive load. This idea of learned knowledge reducing the complexity of a task or idea is captured by **germane cognitive load**. Essentially, germane load is a type of cognitive load incurred by accessing schema. The amount of cognitive load incurred by this type of cognitive load is less than other types due to the use of schemas and their effect of reducing a complex idea/task down to a more compact representation.

To summarise CLT in a rather simplified manner, an individual has a limited amount of working memory to perform tasks and if the amount of cognitive load exceeds the capacity of this working memory, performing the given task becomes much more difficult. There is one major effect and three types of cognitive load that we are interested in: the element interactivity

effect which is the main contributor to intrinsic load and overall cognitive load; extraneous load which is affected by the way in which the material/-task is presented; and germane load that is incurred when accessing schema (previously learned knowledge).

2.3 Summary

In this section I have covered the two main theoretical foundations of conceptual density and the work covered in this report. Learning edge momentum provides the main motivation for this project, while cognitive load theory provides an alternative supporting theory. The rest of this report is structured as follows: in Section 3 I will define a declarative model of text and features that relate to conceptual density; then in Section 4 I will go over the parsing algorithm used for building a mind-map like graph structure; then in Section 5 will talk about some features that can extracted from the graph structure; then in Section 5.3 I will discuss how the extracted graph features can be used to derive a numerical score of conceptual density; in Section 6 I will discuss related work and what distinguishes the work in this project; in Section 7 I will discuss the work-to-date and future work; finally, in Section 8 I will conclude this report.

3 Conceptual Density in Text

At a high level conceptual density is a measure of the complexity of a domain of concepts in a given document. For conceptual density we are interested in the underlying structure of the document and domain of concepts. For example, how many concepts are there, how do concepts interact, how are concepts related, how independent or integrated/interrelated are the concepts, do concepts form self-referential cycles, how are concepts referenced and presented in the document?

The core of conceptual density - the degree to which concepts interact with each other - can be rooted in the element interactivity effect from cognitive load theory. The way that concepts are presented in a document (e.g. how concepts are referenced between sections) can be described in terms of extraneous and germane cognitive load.

In this report I define, and describe ways of measuring, conceptual density with respect to two main components: the degree to which concepts interact

and the nature of the interaction.

3.1 Identifying Concepts in Text

To understand the relationships between concepts, we must first be able to identify concepts in text. A simple definition of a concept that could be used is a noun phrase - a phrase in text that is made up of nouns and optionally adjectives. Using words from the previous sentence as an example, both "definition" and "simple definition" would be valid noun phrases, if we include determiners into the definition of a noun phrase "a simple definition" would also be a valid noun phrase. A more precise definition of the type of noun phrase that will be used is: an optional determiner, followed by a (possibly empty) sequence of nouns and/or adjectives, terminated by a noun. This can also be expressed in regex-like syntax using part of speech tags:

```
NBAR = <DT>?<NN.*|JJ>*<NN.*>
```
(1)

where `NBAR` is used to represent a simple noun phrase, `DT` is a determiner (e.g. a, the), `NN.*` is any type of noun and `JJ` is an adjective.

We can go one step further with this definition. The pattern described above describes simple noun phrases such as 'apple', 'an apple' and 'a red apple'. However, these simple noun phrases may also be joined together by prepositions (e.g. in, of) or coordinating conjunctions (e.g. 'Zeus is the thunder **and** sky god of the ancient Greek religion'). Here I extend the definition of a concept to be a simple noun phrase optionally followed by a sequence of preposition/coordinating conjunction and simple noun phrase pairs. More precisely:

```
NP = <NBAR>(<IN|CC><NBAR>)*
```
(2)

where `NP` is short for noun phrase and represents more complex noun phrases, `IN` is a preposition and `CC` is a coordinating conjunction.

While 'things' as captured by these kinds of noun phrases make up many concepts, there are many concepts that not just things but 'doing things', in other words many concepts are represented as verbs or verb phrases. For example, in programming we can talk about 'calling functions', 'looping' and 'instantiating an object' as concepts, and all of these are verbs or verb phrases. At this point in time, the only type of verb that I have included in the definition is the gerund verb, a type of verb that is typically suffixed with

-ing, such as eat**ing**, sleep**ing**. This rather simple definition of a concept is in line with what is currently implemented. More rich definitions of a concept are to be explored in future work and are briefly discussed in Section 7.

3.2 Presentation of Concepts

The presentation of concepts is included in the definition of conceptual density. What I mean by presentation of concepts is the way that concepts are talked about in a document and how concepts are referenced between sections of the document. For example, a concept in section A may make reference to a concept in section B, likewise a concept in section B may reference a concept that was introduced in section A. These types of interactions between concepts are of interest, and are of relevance to the idea of conceptual density, since the interaction introduces another element that must be considered simultaneously by the reader. This increases element interactivity which is the core of conceptual density. In this section I will propose two types of concepts present in text, and four types of references that are used to measure the effect of presentation on conceptual density.

3.2.1 A Priori & Emerging Concepts

A priori concepts are concepts that are only referenced from within a single section. Typically, this indicates that the concept is one that is expected to be known already by the reader a priori. Similarly to backward references, references to these types of a priori concepts impose minimal cognitive load since it only requires the reader to call on pre-existing and well integrated knowledge.

Emerging concepts are concepts that are referred to from multiple sections. These types of concepts may be similar to a priori concepts in how they are likely to be expected to be known by the reader.

Flesh out this section.

3.2.2 Forward References

Forward references are where the text makes reference to a concept that is not fully explained until later in the document. These types of references introduce extraneous cognitive load since they make the reader *park* the involved concepts, without much existing knowledge to associate them with. Remembering unfamiliar words and terms that carry little meaning is more

```
Bread

Bread
Bread is a staple food typically prepared from a dough of
wheat flour and water, usually by baking.

Wheat
Wheat is a type of grain.
Wheat is commonly used for making wheat flour, a typical
ingredient of bread dough.
```

(a) A simple text document on the topic of bread.

```
<document>
    <title>
        Bread.
    </title>

    <section>
        <title>Bread</title>
        <text>
            Bread is a staple food typically prepared from a
    dough of wheat flour and water, usually by baking.
        </text>
    </section>

    <section>
        <title>Wheat</title>
        <text>
            Wheat is a type of grain.
            Wheat is commonly used for making wheat flour, a
    typical ingredient of bread dough.
        </text>
    </section>
</document>
```

(b) A version of the document in Figure 2a where concepts have been explicitly marked.

Figure 2: A sample document with a plain text version (Figure 2a) and a version marked up with XML (Figure 2b).

of a demanding task than remembering a unit of knowledge that the reader is highly familiar with. For example, when teaching Java programming to beginners (especially in the case of simple programs such as hello world) the meaning and function of keywords such as `class` and `static` are often not explained in full and they are usually just rote learned until a full explanation can be given.

3.2.3 Backward References

Backward references are where the text makes reference to a concept that was explained previously in the text. These types of references are largely unavoidable since they are typically introducing a connection between concepts, something that is intrinsic to the knowledge domain and can only be modified through instructional design by means of omission.

The cost of backward references are relatively low when compared to forward references. With forward references we are making the readers consider an additional element that is not well understood by the reader along with the current context. However, in the case of backward links we are simply asking the reader to recall a previously explained concept and possibly introducing a new relationship between the two concepts. This type of reference imposes germane load as it only requires the reader to call on pre-existing knowledge.

4 Building a Mind-Map

The degree to which concepts in a given text document are integrated can be found through a graph representation of the concepts present in the document. The graph structure that is built is similar to a mind-map, except instead of a single central idea/concept, we are building a rather free-form graph with concepts branching off other related concepts. The type graph that is being built has precedent in semantic networks and knowledge graphs. This graph-based approach to representing the knowledge domain that a text document encodes is a natural representation for the problem of quantifying conceptual density. This is because the core concern of conceptual density is the degree of interaction between the concepts found in a text document, in other words the relationships between the concepts. With respect to the quantification of conceptual density, well-established graph theory and re-

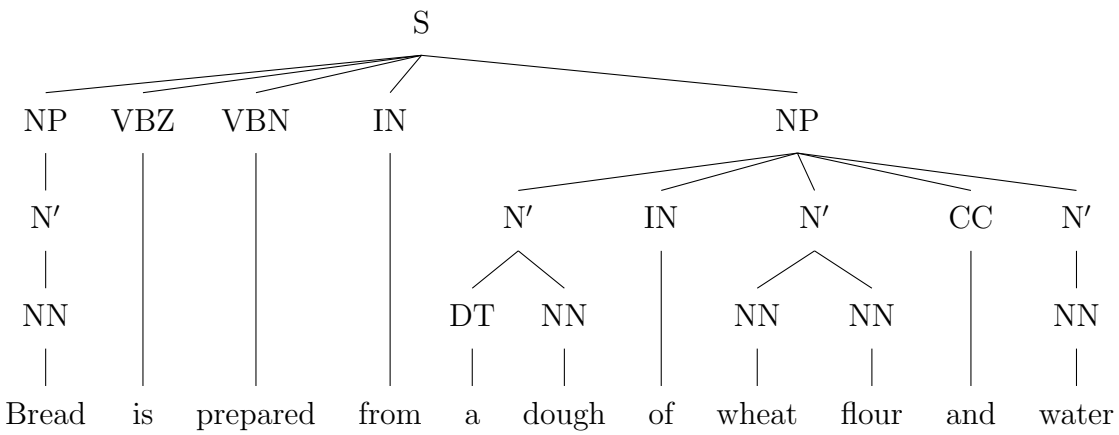

Figure 3: The parse tree resulting from tagging and chunking the sentence "Bread is prepared from a dough of wheat flour and water". The nodes directly above the terminal nodes denote each word's part of speech tag. N′ (NBAR) is the pattern in Equation 1 and NP (noun phrase) is the the pattern in Equation 2.

lated analysis techniques will provide ways to derive a numerical score of conceptual density.

In this section, I will discuss the process of building up a graph structure in text and the aspects unique to this project. The next section, Section 5, will discuss how the graph structure built in this section is analysed and Section 5.3 will discuss how a score is derived from the graph structure and analysis.

4.1 Nodes & Extracting Concepts

In the graph structure we are building a node represents a concept present in a given text document. As discussed in Section 3.1, we create a definition of what constitutes a concept in terms of patterns of parts of speech. The formal definition of the pattern given defines what is called a grammar. To extract a concept from text the basic process is to segment the document into sentences, then split each sentence into a set of tokens (typically words separated by white-space), then for each set of tokens we assign a tag to each token denoting the part of speech of the token. From here we chunk the sentences into groupings based on the patterns described in the grammar.

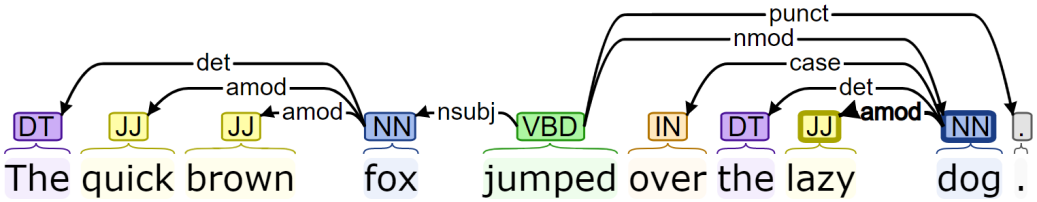

Figure 4: An example of a dependency parse.[1]For a description of what the relations on the arcs mean refer to page 3 of (Martin and Jurafsky, 2009).

An example is shown in Figure 3.

From the example in Figure 3 we can see that noun phrases can be complex and made up of smaller, simpler noun phrases. For example, from the phrase "a dough of wheat flour and water" we could extract the simple noun phrases 'a dough', 'wheat flour' and 'water'. And even from the simple noun phrase 'wheat flour' we can extract two distinct nouns, 'wheat' and 'flour'. As such, I choose to add these constituent concepts to the graph as nodes.

4.2 Edges & Relating Concepts

In the graph structure we are building a relationship between concepts is represented as a directed edge between nodes. Edges are used to represent a 'references' type relationship between concepts. The scheme I choose for relating two concepts has two main criteria: an edge is created from the subject of the sentence to any other concepts in the same sentence; an edge is created between any 'complex' concept and each of the constituent concepts extracted from the complex concept. So how do we identify the subject of a sentence? We can do this through a dependency parse, which marks the relationships between words in a sentence. See Figure 4 for an example.

4.3 Summary

By using part of speech tags and a regex-based chunker we can extract concepts from text. At present concepts are only defined in a limited scope (mainly just nouns). We can derive relationships between concepts by looking at dependency parses of sentence structures and by looking at the constituent parts of complex concepts represented by compound nouns. Concepts are

[1] Figure generated from `https://corenlp.run/`.

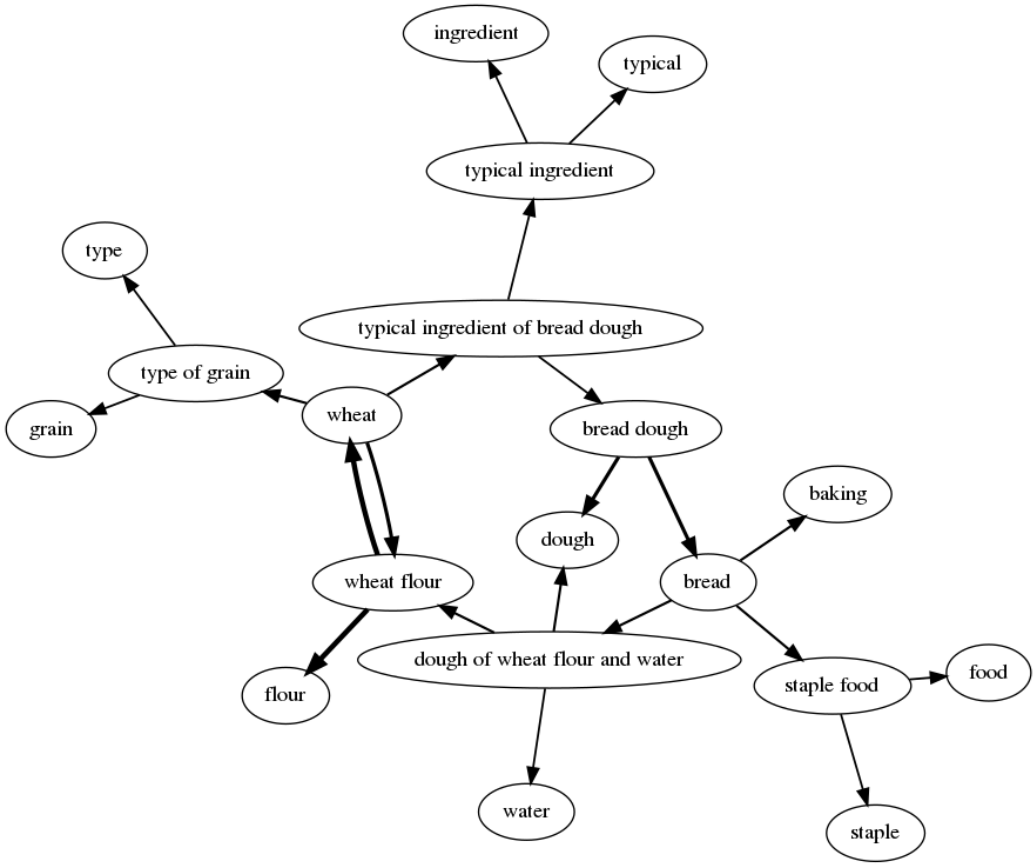

Figure 5: An example of a dependency-based graph created using the XML document from Figure 2b. Nodes represent the concepts that are referenced in the document. Edges in the graph represent the relationship 'A references B'. Blue edges represent forward references and red edges represent backward references.

represented as a node in the graph and relationships between concepts are represented as edges in the graph.

5 Graph Analysis

In this section I will discuss how different types of concepts and references can be identified in a graph, and then how we can derive a score of conceptual density from the graph.

5.1 A Priori & Emerging Concepts

A priori and emerging concepts can be identified in a graph by looking at the incoming edges. For a given node, we can look at each of the edges associated with that node. If all of the incoming edges come from nodes in the same section as the node in question, then we can say that the node represents an a priori concept. Otherwise, if there are two or more sections that the incoming edges originate from, then we can say that the node in question represents an emerging concept.

5.2 Forward and Backward References

In the section-centric graph, forward and backward references can be found through a DFS (depth-first search) and identifying when the algorithm crosses between sections (see Algorithm 1). A prerequisite is that we must record the sections that appear in the text, and the order in which they are introduced during the parsing of the text document. Then, we can perform a DFS originating from the nodes that represent the main concept of the sections. We can identify a forward reference when the DFS algorithm visits a node that is associated with a section that comes after the section of the previously visited node. Similarly, backward references can be identified when the DFS algorithm visits a node that is associated with a section that comes before the section of the previously visited node. When the algorithm crosses between sections the edges between the current and previous node can then be appropriately marked as a forward or backward reference. An example of a graph where the forward and backward egdes have been coloured is shown in Figure 6.

Algorithm 1: Marking Forward and Backward References

```
   Data: sections: List of sections in a given document D
   nodes: Set of nodes in the graph G
   adj: Adjacency list for a directed graph G.
 1
 2 visited = ∅
 3
 4 for node ∈ nodes do
 5 |   markEdges(node, NULL, visited)
 6 end
 7
 8 Function markEdges(curr, prev, visited)
   |   /* Arguments                                          */
   |   /* curr:  Current node                                */
   |   /* prev:  Previous node                               */
   |   /* visited:  Set of visited nodes                     */
 9 |
10 |   if prev ≠ NULL AND curr.section ≠ prev.section then
11 |   |   curr_i ← sections.indexOf(curr.section)
12 |   |   prev_i ← sections.indexOf(prev.section)
13 |   |
14 |   |   if curr_i < prev_i then
15 |   |   |   Mark edge {prev, curr} as a backward reference
16 |   |   else if curr_i > prev_i then
17 |   |   |   Mark edge {prev, curr} as a forward reference
18 |   else if curr ∉ visited then
19 |   |   visited ← visited ∪ {curr}
20 |   |
21 |   |   for neighbour ∈ adj[curr] do
22 |   |   |   markEdges(neighbour, curr, visited)
23 |   |   end
24 end
```

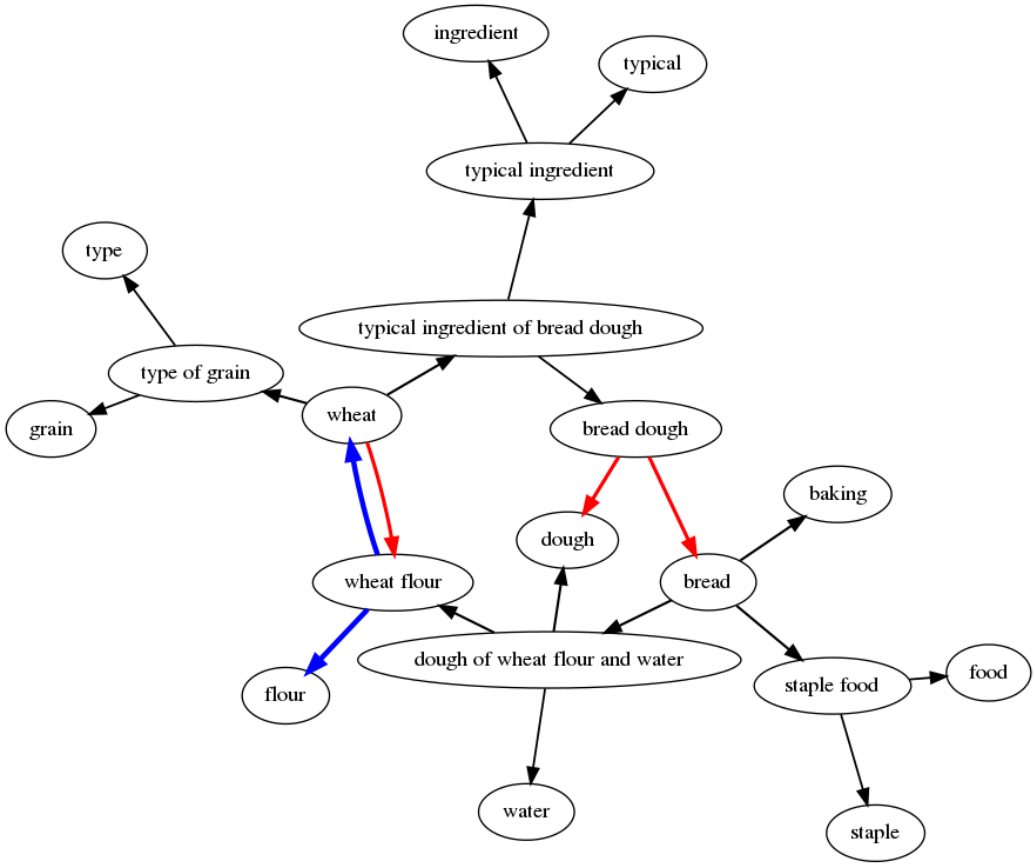

Figure 6: An example of a dependency-based graph created using the XML document from Figure 2b where forward and backward references have been coloured. Nodes represent the concepts that are referenced in the document. Edges in the graph represent the relationship 'A references B'. Blue edges represent forward references and red edges represent backward references.

5.3 Quantifying Conceptual Density

At the centre of conceptual density is the notion of the degree of integration between concepts. In the graph structure this can be measured by looking at the connectivity of nodes. One useful measure may be the average outdegree:

$$S_D = \frac{1}{|V|} \sum_{v \in V} \deg^+(v) \tag{3}$$

where S_D is the score of the document D, V is the set of vertices in the graph, and $\deg^+(v)$ is the outdegree, or the number of outgoing edges from the vertex v. The interpretation of a high average degree can be either: the concept is related to many other concepts and therefore harder to learn since you must learn the other concepts as well; or the concept is related to many other concepts and therefore easier to learn since you have more knowledge to which you can relate the concept. I choose the former interpretation.

6 Related Work

6.1 Information Extraction

The process of identifying concepts and how they are related that I have described in this report is similar to the task of information extraction. However, there are some key differences between what I have done and what is usually done in information extraction. First and foremost, information extraction is typically concerned with named-entity recognition - the identification of named-entities such as names of people, companies, locations, and dates. In this project we are concerned with concepts in general (all types of entities), which can be thought of as a superset of named-entities. Furthermore, information extraction is also interested in the relationship between named-entities. However, in this project and the work-to-date we are not too concerned about the nature of a relationship between concepts as much as we are interested in just the existence of a relationship.

6.2 Semantic Networks & Knowledge Graphs

The type of graph structure that is being built up in this project is very similar to semantic networks and knowledge graphs. Some differences between the graphs in this project and semantic networks are that: we are not

annotating edges with the type of relationship (e.g. X IS_A Y or X HAS_A Y), and the criteria for deciding what constitutes a node or edge in the graph may differ from what is common in the literature.

How so?

7 Future Work

7.1 Transitioning to Unstructured Text

I have implemented a section-centric graph from semi-structured text (e.g. Figure 5), but recreating the same graph from the corresponding plain text version of a document remains difficult. I have discussed some challenges that may arise when building a section-centric graph from unstructured text. A combination of topic segmentation methods and natural language processing techniques to identify topics may enable the creation of such a graph from unstructured text. The quality of the extracted relationships between concepts can also be improved through the use of dependency trees.

Another issue with semi-structured text is that it requires someone to manually mark up the text. Having an implementation that can work with unstructured text would remove the need to do this and allow for much easier use.

7.2 Improving Information Extraction

One issue with the current work is the quality of the extracted concepts and how the concepts are related. There are many cases where the extracted concepts do not align with the concepts that a human would intuitively pick out, and similarly for how concepts are related. While the current quality is adequate enough to build up a graph structure, important concepts and relationships between concepts are likely to be missed. So it is important that future work focuses on improving the quality of extracted concepts and relations.

7.2.1 Extending the Definition of a Concept in Text

In addition to noun phrases we can include other types of phrases into the definition of a 'concept'. We can extend the definition of noun phrases to include two noun phrases joined by a preposition. For example, consider the phrase "relationships between concepts". Under the previous definition we

would consider "relationships" and "concepts" as two separate noun phrases, however it is equally sensible to consider the entire phrase as a single entity. We can also include particular types of verb phrases. For example, consider the sentence "Listening to music is his favourite way to pass time", what are the concepts being talked about here? I believe it would be sensible to say that the concepts of "listening to music" and "passing time" are being discussed. Adding these types of phrases leads to a more rich definition of a 'concept'.

7.2.2 Better Dependency Parsing

The current scheme for relating concepts in the dependency-based graph simply relates the subject of a sentence to the other concepts that appear in the sentence. This does not capture the nuances of natural language very well. For example, consider the sentence "Wheat is used for making wheat flour, a typical ingredient of bread dough" from Figure 2a. Under the current scheme the subject, 'Wheat', would have an edge created between itself and the other concepts 'wheat flour' and 'typical ingredient of bread dough'. However, the sentence is phrased such that wheat flour is being to referred to as a 'typical ingredient of bread dough', not 'wheat'. A more correct set of edges in this case would connect 'wheat' to 'wheat flour' and 'wheat flour' to 'typical ingredient of bread dough'.

7.2.3 Information Extraction Frameworks

Much of the work in this project has been inspired by other work but built from scratch. As discussed earlier, much of the process of extracting and relating concepts is similar to the task of information extraction. There exists a framework called OpenIE (Banko et al., 2007) which is made for performing information extraction. The benefits of using such a framework is that many of the issues with the current dependency parsing in this project have more or less been solved in OpenIE. A couple of issues that I foresee with adapting a framework such as OpenIE to use in this project are that OpenIE is geared towards working with named-entities and it may be difficult to recover section information relating to extracted entities.

7.3 Other Graph Features

There are many graph analysis techniques that have not been explored in this report (see (NetworkX Developers, 2004), for example). It would be interesting to see if incorporating features derived from these analysis techniques into the scoring scheme improves the quality of it.

7.3.1 Bottlenecks

Another interesting possible feature in the graph topology is the idea of a bottleneck vertex/vertices that might be able to be found by identifying the minimum cuts of the graph. One could imagine a bottleneck in a graph structure being visually similar to a hourglass. The vertex at the centre of the hourglass shape would be the bottleneck vertex. This would correspond to a concept that is defined in terms of other concepts and is used to define a set of other concepts. For example, in programming polymorphism and inheritance cannot be explained if classes are not explained, similarly classes cannot be explained if variables and functions are not explained. While in this example we could imagine back references from polymorphism to functions, to explain polymorphism we must first explain classes. Therefore, directionality must included when defining and identifying bottleneck concepts. Bottleneck concepts are important to the idea of conceptual density because they make learning more difficult.

7.3.2 A priori concepts

It may be possible to redefine what an a priori concept is in terms of sinks in a graph. A sink is a node in a directed graph that has an outdegree of zero, i.e. it has no outgoing edges. Identifying sinks in a graph is trivial, but we could also extend the idea of an a priori concept being a concept that is only referenced from within one section to: a concept that only leads to a sink and is not part of a self-referential system.

7.4 Other Text Features

There are other features that could be extracted from text. In (Robins, 2010) it is suggested that a measure of distance between keywords (concepts) could be used to help measure conceptual density. There are some methods that may help with this such as: using lexical chains for keyword extraction (Ercan

and Cicekli, 2007); and looking at long distance dependency as a metric of reading comprehension difficulty (Liu, 2008).

7.5 Evaluation of the Scoring Scheme

Regarding the evaluation of the scoring algorithm, the quality of the scores derived from graphs that were generated from semi-structured text have only been evaluated qualitatively in an ad-hoc manner. One particular experiment that I envision is one performed on a labelled data set of abstracts created from human judgements. We need to use human judgements since, as discussed near the start of this report, there no objective measures of conceptual density or element interactivity and as such human judgements are the only baseline available to compare the model with. For the collection of documents, we could gather a number of snippets from various textbooks that consist of a few sections. The reason for choosing snippets that contain is that they are short, self-contained, expository documents. And if we are to get human judgements then it will be a lot easier and a lot less time consuming to use abstracts as apposed to long documents such as textbooks. Once a collection of abstracts have been created we could group abstracts into pairs and get a panel of human judges to judge which document appears to be more conceptually dense. Then, we could compare the output of the model against the human judgements and see how well it mimics the human judgements.

7.6 Feature Importance Analysis

From the process described in Section 7.5 we would create a labelled data set. From this we could train a machine learning algorithm such as a simple linear regression or random forest. Then through feature importance analysis we could investigate which features out of all of the extracted features (from text and the graph structure) are the most salient in regards to conceptual density. And with a large enough data set, we could simply extract as many features as possible, regardless of whether or not they seem relevant to conceptual density, and then use this feature importance analysis to filter out the unimportant features in a principled way.

8 Conclusion

In this report I have explored a graph-based approach to quantifying conceptual density - an idea relating to the degree that concepts of a particular knowledge domain, encoded by a text document, are integrated. I have discussed how this idea of conceptual density is grounded in, and related to, learning edge momentum and cognitive load theory. I have discussed the ways in which the notion of conceptual density can be defined and measured in semi-structured and unstructured text documents. By analysing the language in the documents and observing the patterns in natural language we can create an operational definition of a 'concept' in text. Then by analysing the relationships between concepts in a given document we can build up a graph structure from which we can derive a score of conceptual density.

References

Robert P Abelson. Psychological status of the script concept. *American psychologist*, 36(7):715, 1981.

Robert Axelrod. Schema theory: An information processing model of perception and cognition. *American political science review*, 67(4):1248–1266, 1973.

Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*, volume 7, pages 2670–2676, 2007.

Frederic Charles Bartlett, Frederic C Bartlett, and Walter Kintsch. Remembering: A study in experimental and social psychology, 1995.

Paul Chandler and John Sweller. Cognitive load while learning to use a computer program. *Applied cognitive psychology*, 10(2):151–170, 1996.

Gonenc Ercan and Ilyas Cicekli. Using lexical chains for keyword extraction. *Information Processing & Management*, 43(6):1705–1714, 2007.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. Text generation from knowledge graphs with graph transformers. *arXiv preprint arXiv:1904.02342*, 2019.

Haitao Liu. Dependency distance as a metric of language comprehension difficulty. *Journal of Cognitive Science*, 9(2):159–191, 2008.

James H Martin and Daniel Jurafsky. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson/Prentice Hall Upper Saddle River, 2009.

NetworkX Developers. Algorithms - networkx 2.3 documentation. https://networkx.github.io/documentation/stable/reference/algorithms/index.html, 2004. Accessed on 05 July 2019.

Anthony Robins. Learning edge momentum: A new account of outcomes in cs1. *Computer Science Education*, 20(1):37–71, 2010.

John F Sowa. Semantic networks. 1987.

John Sweller. Cognitive load theory, learning difficulty, and instructional design. *Learning and instruction*, 4(4):295–312, 1994.

John Sweller, Paul Ayres, and Slava Kalyuga. The element interactivity effect. In *Cognitive Load Theory*, pages 193–201. Springer, 2011.

Liecai Zhang. *Knowledge graph theory and structural parsing*. Twente University Press, 2002.