Bull. Nov. Comp. Center, Comp. Science, 41 (2017), 55-89

Graph quotients: a topological approach to graphs

L. Vepštas

Abstract. This paper develops general concepts useful for extracting knowledge embedded in large graphs or datasets that have pair-wise relationships, such as relations of cause-effect type. Almost no underlying assumptions are made, other than that the data can be presented in terms of pair-wise relationships between objects/events. This assumption is used to mine for patterns in a dataset defining a reduced graph or a dataset that boils-down or concentrates information into a more compact form. The resulting extracted structure or the set of patterns are manifestly symbolic in nature, as they capture and encode the graph structure of the dataset in terms of a (generative) grammar.

Keywords: knowledge extraction, graph structure, type theory, natural language processing, link grammar, sheaf theory.

1. Introduction

This paper presents some definitions and a vocabulary for working with datasets that contain complex relationships applicable to a large variety of application domains. The concepts borrow from graph theory and several other areas of mathematics. The goal is to define a way of thinking about complex graphs, and how they can be simplified and condensed into simpler graphs that "concentrate" embedded knowledge into a more manageable size. The output of the process is a grammar that summarizes or captures significant or important relationships.

The ideas described here are not terribly complex; they represent a kind-of "folk knowledge" generally known to a number of practitioners. However, I am not currently aware of any kind of presentation of this information, either in a review/summary form, or as a fully articulated book or text. The background knowledge appears to be scattered across wide domains and occur primarily in highly abstract settings, outside of the mainstream computer science and data analysis domain. Thus, this paper tries to provide an introduction to these concepts in a plain-spoken language. The hope is to be precise enough that there will be few complaints from the mathematically rigorous-minded, yet simple enough that "anyone" can follow through and understand.

Some examples are provided, primarily drawn from linguistics. However, the concepts are generally applicable, and should prove useful for analyzing any kind of datasets expressed with pair-wise relationships, but containing hidden (non-obvious) complex cause-and-effect relationships. Such datasets include genomic and proteomic data, social graph data, and even such social policy information.

Consider the example of determining the effectiveness of educational curricula.

When teaching students, one never teaches advanced topics until foundations are laid. Yet many students struggle. Given raw data on a large sample of students, and the curricula they were subjected to, can one discern sequences and dependencies of cause-and-effect in this data? Can one find the most effective curriculum to teach, that advances the greatest number of students? Can one discover different classes of students, some who respond better to one style than another? My belief is that these questions can not only be answered, but that the framework described here can be used to uncover this structure.

Another example might be the analysis of motives and actions in humans. This includes analysis from real life, as well as the narratives of books and movies. In a book setting, the author cannot easily put characters into action until some basic sketch of personality and motives is developed. Motives cannot be understood until a setting is established. If one can break down a large number of books/movies into pairs of related facts/scenes/remarks/actions, one can then extract a grammar of relationships to see exactly what is involved in the movement of a narrative from here to there.

Much of this paper is devoted to stating definitions for a few key structures used to talk about the general problem of discerning relationships and structures. The definitions are inspired by and draw upon concepts from algebraic topology, but mostly avoid both the rigor and the difficulty of that topic.

The definitions provide a framework, rather than an algorithm. It is up to the user to provide some mechanism for judging similarity, and this can be anything: some neural net, Bayesian net, Markov chain, or some vector space or SVM-style technique; the overall framework is agnostic as to these details. The goal is to provide a way of talking about, thinking about and presenting data so that the important knowledge contained in it is captured and described, boiled down to a manageable, workable state from a large raw dump of pair relationship data.

Currently, the ideas described here are employed in a machine learning project that attempts to extract the structure of natural language in an unsupervised way. Thus, the primary, detailed examples will come from the natural language domain. The theory should be far more general than that.

This paper resides in, accompanies source code that implements the ideas here. Specifically,it is in https://github.com/opencog/atomspace/tree/master/opencog/sheaf and it spills over into other files, such as https://github.com/opencog/opencog/blob/master/opencog/nlp/learn/scm/gram-class.scm . This code is in active development, and it most likely has changed a lot since it was written. This paper is *not* intended to describe the code; rather, it is meant to describe the general underlying concepts.

For the mathematically inclined, please be aware that the concepts described here touch on the tiniest tips of some very deep mathematical icebergs, specifically in parsing, type theory and category theory. I have no hope of providing the needed background, as these fields are sophisticated and immense. The reader is encouraged to study these on their own, especially as they are applied in computer science and linguistics. There are many good texts on these

topics.

This paper is organized as follows. The first part provides a definition of a "section" of a graph. A section is a lot like a subgraph, except that it explicitly indicates which edges were cut to form the subgraph. The next part makes use of this concept of "sections" to show how they can be used to talk about and understand pattern mining, clustering and quotienting. The next part indicates how such clusters can be understood to be "types" in the formal sense of the mathematical type theory. Next follows a brief review of the concept of parsing, as it applies to this context. The ability to parse is what motivates the data discovery to begin with: after extracting patterns from a dataset, parsing is how those patterns can be re-assembled. An important part of pattern mining is the ability to distinguish polymorphic behavior. The final part shows how the system as a whole can be understood to be a kind of a sheaf, borrowing a concept from a different branch of mathematics.

2. Sections

Begin with the standard definition of a graph.

Definition. A GRAPH $G=(V, E)$ is an ordered pair (V, E) of two sets, the first being the set V of vertices, and the second being the set E of edges. An edge $e \in E$ is a pair (v_1, v_2) of vertices, where every v_k must be a member of V. That is, edges in E can only connect vertices in V, and not something else.

For directed graphs, the vertex ordering in the edge matters. For undirected graphs, it does not. The subsequent will mostly leave this distinction unspecified and allow either (or both) directed and undirected edges, as the occasion and the need fits. Distinguishing between directed and undirected graphs is not important, at this point. In most of what follows, it will usually be assumed that there are no edges with $v_1=v_2$ (loops that connect back to themselves) and that there is at most one edge connecting any given pair of vertices. These assumptions are being made to simplify the discussion; they are not meant to be a fundamental limitation. It just makes things easier to talk about and less cluttered at the start. The primary application does not require either construct, and it is straightforward to add extensions to provide these features. Similar remarks apply to graphs with labeled vertices or edges (such as "colored" edges, vertices or edges with numerical weights on them, etc). Just keep in mind that such additional markup may appear out of thin air, later on.

Besides the above definition, there are other ways of defining and specifying graphs. The one that will be of primary interest here will be one that defines graphs as a collection of sections. These, in turn, are composed of seeds.

Definition. A SEED is a vertex and the set of edges that connect to it. That is, it is the pair (v, E_v) where v is a single vertex, and E_v is a set of edges containing that vertex, i.e. that set of edges having v as one or the other endpoint. The vertex

v may be called the GERM of the seed.

It should be clear that, given a graph *G*, one can equivalently describe it as a set of seeds (one simply lists all of the vertices, and all of the edges attached to each vertex). The converse is not "naturally" true. Consider a single seed, consisting of one vertex $v1$, and a single edge $e = (v1, v2)$. Then the pair (V, E) with $V = \{v1\}$ and $E = \{(v1, v2)\}$ is not a graph, because $v2$ is missing from the set *V*. Of course, we could implicitly include $v2$ in the collection of vertices, but this is not "natural", if one is taking the germs of the seeds to define the vertices of the graph.

Thus, given a seed, each edge in that seed has one "connected" endpoint and one "unconnected" endpoint. The "connected" endpoint is that endpoint that is *v*. The other end, point will commonly be called the CONNECTOR; equivalently, the edge can be taken to be the connector. Perhaps it should be called a half-edge, as one end-point is specified but missing.

The seed can be visualized as a ball, with a bunch of sticks sticking out of it. A burr one might collect on one's clothing. One can envision a seed as an analog of an open set in topology: the center (the germ) is part of the set, and then there are some more, but the boundary is not part of the set. The vertices on the unconnected ends of the edges are not a part of the seed.

Just as one can cover a topological space with a collection of open sets, so one can also cover a graph with seeds. This analogy is firm: if one has open sets U_i and U_j and $U_i \cap U_j \neq \emptyset$, then one can take U_i and U_j to be vertices, and $U_i \cap U_j$ to be an edge running between them.

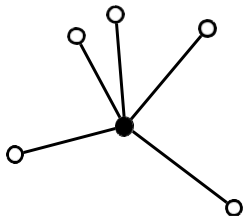


Figure 1. A seed

More definitions are needed to advance the ideas of connecting and covering.

Definition. A SECTION is a set of seeds. ◇

It should be clear that a graph *G* can be expressed as a section; that section has a nice property that all of the germs appear once (and only once) in the set *V* of *G*, and that all of the edges in *E* appear twice, once each in two distinct seeds. This connectivity property motivates the following definition.

Definition. Given a section *S*, a LINK is any edge (v_1, v_2) where both v_1 and v_2 appear as germs of seeds in *S*. Two seeds are CONNECTED when there is a link between them.

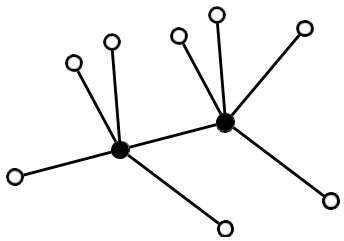


Figure 2. Two linked (connected) seeds

The use of links allows the concepts of paths and connectivity, taken from graph theory, to be imported into the current context. Thus, one can obviously define:

Definition. A CONNECTED SECTION, or a CONTIGUOUS SECTION, is a section where every germ is connected to every other germ via a path through the edges.

In graph theory, this would normally be called a "connected graph", but we cannot fairly call it that because the seeds and sections were defined in such a way that they are not graphs; they only become graphs when they are fully connected. Nevertheless, it is fairly safe and straightforward to apply common concepts from graph theory. Sections are almost like graphs, but not quite.

Note that there are two types of edges in a section: those edges that connect to nothing, and those edges that connect to other seeds in that section. Henceforth, the unconnected edges will be called connectors (as defined above), while the fully connected edges will be called links (also defined above). Connectors can be thought of as a kind-of half-edge: incomplete, missing the far end, while links are fully connected, whole.

Seeds and sections can (and should!) be visualized as hedgehogs - a body with spines sticking out of it - the connectors can be thought of as the spiny bits sticking out, waiting to make a connection, while the hedgehog body is that collection of vertices and the fullyconnected links between them.

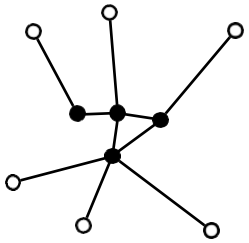


Figure 3. A connected section

Implicit in the above definitions was that, during link formation, an edge is only

allowed to connect to another seed if and only if the connector matches the germ. That is, if (v_1, v_2) is an edge rooted in the seed for v_1 and if (v_3, v_4) is an edge rooted in the seed for v_3, then these two can form a link if and only if $v_2 = v_3$ and $v_4 = v_1$. That is, the connectors are typed: they can only connect to seeds that are of the same type as the unconnected end of the edge.

This motivates a different way of looking at seeds: they can be visualized as jigsaw puzzle pieces, where any given tab on one jigsaw piece can fit into one and only one slot on another jigsaw piece. This union of a tab+slot is the link. Connectors must be of the same type in order to be connectable. The types of the connectors will later be seen to be the same thing as the types of type theory; that is, they are bona-fide types, in the proper sense of the word.

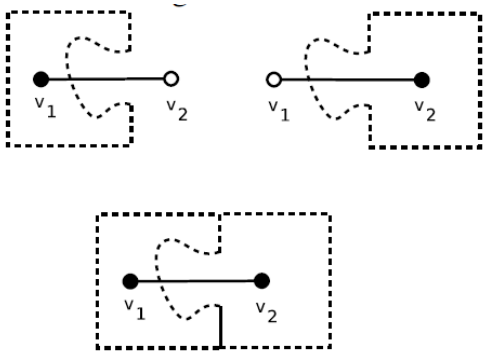


Figure 4. Joining two connectors to form a link

The jigsaw puzzle-piece illustration is not uncommon in the literature; such illustrations are explicitly depicted in a variety of settings [1-4].

Why sections? What is the point of introducing this seemingly non-standard approach to something that looks a lot like graph theory? There are several reasons.

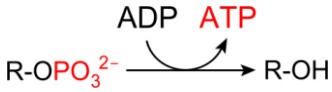
- From a computational viewpoint, sections have nice properties that a list of vertices and edges do not. Given a single seed, one "instantly" knows *all* of the edges attached to its germ: they are listed right there. By contrast, given only a graph description, one has to search the entire list E for any edges that might contain the given vertex. Computationally, searching large lists is inefficient, especially so for very large graphs.
- A subset of a section is always a section. This is not the case for a graph: given $G = (V, E)$, some arbitrary subset of V and some arbitrary subset of E do not generally form a graph; one has to apply consistency conditions to get a subgraph.
- A connected section behaves very much like a seed: just as two seeds can be linked together to form a connected section, so also two connected

sections can be linked together to form a larger connected section. Both have a body, with spines sticking out. The building blocks (seeds), and the things built from them (sections) have the same properties and lie in the same class. Thus, one has a system that is naturally "scalable" and allows the notions of similarity and scale invariance to be explored. There is no need to introduce additional concepts and constructions.

- Given two seeds, one can always either join them (because they connect) or it is impossible to connect them. Either way, one knows immediately. Graphs, in general, cannot be joined, unless one specifies a subgraph in each that matches up. Locating subgraphs in a graph is computationally expensive; verifying subgraph isomorphism is computationally expensive.
- The analogy between graphs and topology, specifically between open sets and seeds and the intersection of open sets and edges, allows concepts and tools to be borrowed from algebraic topology.

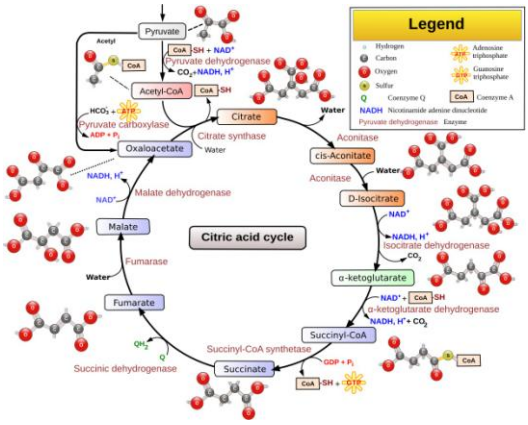
If we stop here, not much is accomplished, other than to define a somewhat idiosyncratic view of graph theory. But that is not the case; the concept of seeds and sections is needed to pursue more complex constructions. They provide a tool to study natural language and other systems.

Example: Biochemical reaction type. An example of a seed applied to the biochemical domain would be the phosphorylation of ADP to ATP shown in the figure below.



The germ of the seed is the point where the semi-circle kisses the line: not labeled here, the germ would be succinate-CoA ligase. The connectors are labeled with their types, and the arrows provide directionality. The connector types clearly indicate what can be linked to what: this particular seed, when linked, *must* link to a source of ADP, or a source of phosphate, or a sink if ATP or a sink of hydroxyls, if it is to be validly linked into any part of a connected section.

An example of a connected section would be the Krebs cycle taken as a whole:



Each distinct reaction constitutes a seed; the heavy lines forming the cycle are the links internal to the section, and each tangent arrow is a pair of connectors, with one end of the arrow being an unconnected reaction input, and the other end of the arrow an unconnected reaction product. Thus, for example, connector types include NAD, NADH, water and ATP, among others. These connectors are free to be attached to other seeds or sections.

Similar concept: Link Grammar. Readers familiar with Link Grammar [1, 5] should have recognized seeds as being more or less the same thing as "disjuncts" in Link Grammar. The formal definition for Link Grammar disjuncts are a bit more complicated than seeds, and is expanded on in later sections. To lay that groundwork, however, consider an unlabelled dependency parse for the sentence "this is an example" shown in the figure below.

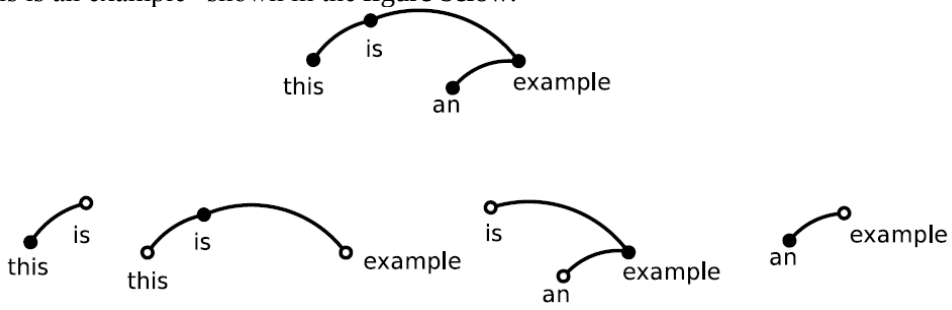


Figure 5. A dependency parse decomposed into four seeds

The dependency parse is shown as a graph with four vertices. Below, the parse is decomposed into the component seeds; as always, the open dots are connectors, the closed dots are germs. Using the notation (v, E_v) for a seed, where $E_v = \{(v,v_a), (v,v_b), \cdots\}$, these seeds can be textually written as follows:

this: {(this, is+)}
is: {(is, this-), (is, example+)}
an: {(an, example+)}
example: {(example, is-), (example, an-)}

The above vertex: edge-list notation is a bit awkward and hard to read. A simpler notation conveying the same idea is

this: is+;
is: this- & example+;
an: example+;
example: an- & is-;

In both textual representations, the pluses and minuses are used to indicate word-order: minuses to the left, pluses to the right. This is an additional decoration added to the connectors, needed to indicate and preserve word-order, but not a part of the core definition of a seed. The ampersand is not symmetric, but enforces order; this is not apparent here, but is required for the proper definition.

In Link Grammar, the objects to the right of the colon are called "disjuncts". The name comes from the idea that they disjoin collocation extractions. After observing a large corpus, one might find that

```
is: (this- & example+) or (banana- & fruit+) or (apple- & green+);
```

which indicates that sentences such as "a banana is a kind of fruit" or "this apple is green" were observed and parsed into (unlabelled) dependencies.

Similar concept: lambda notation. Linguistics literature sometimes describes similar concepts using a lambda-calculus notation. For example, one can sort-of envision the expression $\lambda M.xyz$ as a seed with the germ M and with connectors x, y and z. This notation has been used to express the concept of a seed, as described above. For example, Poon and Domingos [6] write $\lambda y \lambda x.\text{borders}(x, y)$ to represent the attachments of the word "borders" as a synonym for "is next to". This is illustrated with the verb-phrase $\lambda y \lambda x.\text{borders}(x, y)(\text{Idaho})$ which beta-reduces to the verb-phrase $\lambda x.\text{borders}(x, \text{Idaho})$ to indicate that x is next to Idaho. The utility of this device becomes apparent because one can use this same notation to write $\lambda y \lambda x.\text{is\_next\_to}(x, y)$ and $\lambda y \lambda x.\text{shares\_a\_border\_with}(x, y)$ as synonymous phrases. The lambda notation allows x and y to be exposed as connectors, while at the same time hiding the links that were required to assemble seeds for "next", "is", and "to" into a phrase. That is, $\lambda y \lambda x.\text{is\_next\_to}(x, y)$ is an example of a connected section, having x and y as the externally exposed connectors and the internal links between "next", "is", and "to" hidden.

The problem with this notation is that, properly speaking, lambda calculus is a system for generating and working with strings, not with graphs, and lambdas are designed to perform substitution (beta-reduction), and not for connecting things.

That is, lambda terms are always strings of symbols, and the variables bound by the lambda are used to perform substitutions. To illustrate the issue, suppose that M above is $axbyczd$ and suppose that $\lambda N.w = ewf$. Can these be "connected" together, linked together like seeds? No: if one tried to "connect" N to z, one has the beta-reduction $(\lambda M.xyz)\lambda N.w \rightarrow \lambda axbycewfdxyw$. There is no way to express some symmetric version of this, because $(\lambda N.w)\lambda M.xyz \rightarrow \lambda eaxbyczdf.xyz$ which is hardly the same. Now, of course, lambda calculus has great expressive power, and one could invent a way encoding graph theory, and/or seeds, in lambda calculus; however, doing so would result in a verbose and complex system. It is easier to work with graphs directly, and just sleep peacefully with the knowledge that one could encode them with lambdas, if that is what your life depended on.

Note also that there have been extensions of the ideas of lambda calculus to

graphs; however, those extensions cling to the fundamental concept of beta reduction. Thus, one works with graphs that have variables in them. Given a variable, one plugs in a graph in the place of that variable. The OpenCog (https://opencog.org/) works in exactly this way. The beta-reduction is fundamentally not symmetrical: putting A into B is not the same as putting B into A. The concept of "connecting" in a symmetric way does not arise.

Similar concept: tensor algebra. The tensor algebra is an important mathematical construct underlying large parts of mathematical analysis, including the theory of vector spaces, the theory of Hilbert spaces, and, in physics, the theory of quantum mechanics.

It has been widely noted that tensor algebras have the structure of monoidal categories; perhaps the most insightful and carefully explained such development is given by Baez and Stay [4]. The diagram of a tensor shown above is taken from that paper; it is a diagrammatic representation of a morphism $f : X_1 \otimes X_2 \otimes X_3 \to Y_1 \otimes Y_2$. There are several interesting operations one can do with tensors. One of them is the contraction of indexes between two tensors. For example, to multiply a matrix M_{ik} by a vector v_k, one sums over the index k to obtain another vector: $w_i = \sum_k M_{ik} v_k$. The matrix M_{ik} should be understood as a 2-tensor, having two connectors, while vectors are 1-tensors. The intent here is that M_{ik} is to be literally taken as a seed with the germ M, and the connectors i and k on the germ. The vector v_k is another seed with the germ v and connector k. The inner product $\sum_k M_{ik} v_k$ is a connected section. The multiplication of vectors and matrices is the act of connecting together connectors to form links: multiplication is linking.

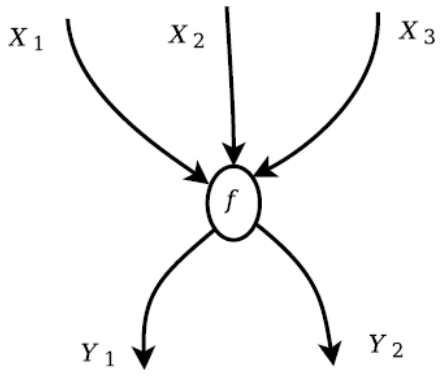


Figure 6. A tensor with three input wires and two output wires

Tensors have additional properties and operations on them, the most important of which, for analysis, is their linearity. For the purposes here, the linearity is not important, whereas the ability to contract indexes is. The contraction of indexes, that is, the joining together of connectors to form links, gives tensor algebras the structure of a. monoidal category. This is a statement that seems simple, and yet carries a lot of depth. As noted above, the beta-reduction of lambda calculus also

looks like the joining together of connectors This is not accidental; rather, it is the side effect of the fact that the internal language of closed monoidal categories is simply typed lambda calculus. The words "simply typed" are meant to convey that there is only one type. For the above example morphism, that would mean that X_1 and X_2 and so on all have the same type: $X_1 = X_2 = X_3 = Y_1 = Y_2$. The end-points on the seed are NOT labeled; equivalently, they all carry the same label. This is in sharp contrast to the earlier example

is: this- & example+;

where the two connectors are labeled and have different types, which sharply limits what they connect to. The this- connector has the this-is and can only attach to another connector having the same type, namely, the is+ connector on "this"

this: is+;

It may seem strange to conflate the concept of tensors and monoidal categories with linguistic analysis, yet this has a rich and old history, briefly touched on in the next section. The core principle driving this is that the Lambek calculus, underpinning the categorial grammars used in linguistic analysis, can be embedded into a fragment of non-commutative linear logic. The remaining step is to recall that the linear logic is the logic of tensor categories; the non-commutative aspect is a statement that the left and right products must be handled distinctly.

Similar concept: Lambek Calculus. The foundations of categorial grammars date back to Lambek in 1961 [7, 8] and the interpretation in terms of tensorial categories proliferates explosively in modern times. One direct example can be found in works by Kartsaklis [9, 3], where one can find not only a detailed development of the tensorial approach, together with its type theory, but also explicit examples, such as the tensor

$$\overrightarrow{men} \otimes \overline{built} \otimes \overrightarrow{houses}$$

together with explicit instructions on how to contract this with a different tensor

$$\mathcal{F}(\alpha_{\text{subj verb obj}}) = \varepsilon_W \otimes 1_W \otimes \varepsilon_W$$

to obtain the "quantization" of the sentence "men built houses". This notation will not be explained here; the reader should consult [9] directly for details. The point to be made is that this kind of tensorial analysis can be, and is, done and often invokes words like "quantum" and "entanglement" to emphasize the connection to linear logic and linear type theory.

Unfortunately, it is usually not clearly stated that it is only a fragment of linear

logic and linear type theory that applies. In linguistics, it is not the linearity that is important, but rather the conception of frames (in the sense of Kripke frames in proof theory). Frames have an important property of presenting choices or alternatives: one can have either this or can have that. The property of having alternatives is described by intuitionistic logic, where the axiom of double-negation is discarded. This either-or choice appears as the concept of a "multiverse" in quantum mechanics, and far more known as alternative parses in linguistics.

Another worthwhile example of tensor algebra can be found in equation 13 of [3] reproduced below:

$$\overrightarrow{verb} = \sum_i (\overrightarrow{subject}_i \otimes \overrightarrow{object}_i)$$

where $\overrightarrow{object}_I$ and $\overrightarrow{subject}_i$ are meant to be the ith occurance of a subect/object pair in an observed corpus. If the corpus consisted of two sentences, "a banana is a kind of fruit" or "this apple is green", then one would write

$$\overrightarrow{verb} = (\overrightarrow{banana} \otimes \overrightarrow{fruit}) + (\overrightarrow{apple} \otimes \overrightarrow{green}),$$

where the verb, in this case, is "is". The control over the word order, that is, the left-right placement of the dependencies, is controlled by means of the pregroup grammar. The pregroup grammar and its compositionality properties follow directly from the properties of the left-division, right-division and multiplication in the Lambek calculus. A quick modern mathematical review of the axioms of the Lambek calculus can be found in Pentus [10], which also provides a proof of equivalence to context-free grammars.

Similar concept: history and Bayesian inference. Some first-principles applications of Bayesian models to natural language explicitly make use of a sequential order, called the "history" of a paper [11]. That is, the probability of observing the n-th word of a sequence is taken to be $P(w_n|h)$, where $h = w_{n-1}, w_{n-2}, \ldots, w_1$ is termed "the history". This conception of probability is sharply influenced by the theory of Markov processes and finite-state machines, dating back to the dawn of information theory [12]. In a finite-state process model, the future state is predicated only on the current state, and thus the Markov assumption holds. In deciphering such a process, one might not know how the current state is correlated to the output symbol, thus leading to the concept of a Hidden Markov Model (HMM). The concept of "history" is well-suited for such analysis. Several issues, however, make this approach impractical for many common problems, including natural language.

One issue, already noted, is the sequential nature of the process. One can try to hand-wave away this issue: given a graph of vertices, it is sufficient to write the vertices in some order, any order will do. This obscures the fact that n vertices have $n!$ (n-factorial) possible

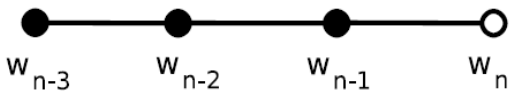


Figure 7. The history of a text as a sequence of words

interactions: a combinatorial explosion, when the actual data graph may have a much smaller number of interactions between vertices (aka "edges"). By encoding the known interactions as edges, a graphical approach avoids such a combinatorial explosion from the outset.

Actually a sequential history model of genomic and proteomic data is inappropriate. Although the base pairs and amino acids come in sequences, the interactions between different genes and proteins are not in any way form sequential. The interactions are happening in parallel, in distinct, different physical locations in a cell. These interactions can be depicted as a graph. Curiously, that graph can resemble the one depicted below, although the depiction is meant to show something different: it is meant to show a history.

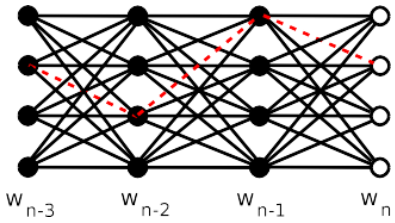


Figure 8. A Viterbi parse lattice of a Markov chain

The above depicts the lattice of a Viterbi parse of a Markov chain. The dashed red line depicts a maximum-likelihood path through the lattice, that is, the most likely history. Viterbi decoding, using an "error correcting code", is a process by which the validity of the dashed red path is checked, and failing paths discarded. For natural language, the dashed red path must be a grammatically correct sequence of words. For a radio receiver, the dashed red path must be a sequence of bits that obey some error-correction polynomial; if it does not, the next-most-likely path is selected.

Each black line represents a probability p_{ij} of moving from a state i to a state j at the next time-step. That is, $p_{ij} = P(w_n = j | w_{n-1} = i)$ is the likelihood of the word j given the word i in the immediate past. The probabilities are arranged so that $\sum_i p_i = 1$. This is called a Markov model, because only the most recent state transitions are depicted: there are no edges connecting the nodes more than one time-step apart; there are no edges connecting w_n to w_{n-2}, etc. Put it differently,

$P(w_n|h) = P(w_n|w_{n-1})$. That is, this depicts the use of 2-grams to predict the current state.

Non-Markov models would have edges connecting nodes further in the past. An *n*-gram approach to language digs *n* steps into the past. If there are *k* states, and *n* steps into the past, then k^n edges are required: that is, a rank-*n* tensor. Here, $k = 4$ and $n = 2$ are depicted; *k* is the number of words in natural language (say, $k = 10^4$ for a common subset of the English language), while *n* is the length of a longer sentence, say $k = 12$. In this case, the history tensor $P(w_n|h)$ has $k^n = 10^{48} = 2^{160}$ edges. But of course, this is computationally absurd. It is also theoretically absurd: almost all of those edges have zero probability. Almost none of the edges are needed; the actual tensor is very-very sparse.

The reason for this sparsity becomes apparent from the viewpoint of dependency parsing. So, for example, if $w_{n-3} = this$ and $w_{n-2} = is$ and $w_{n-1} = an$, a dependency parse will tell you that w_n must be a noun starting with a vowel. It also tells you that, for this particular history, this noun can depend only on w_{n-2} and w_{n-1} but not w_{n-3}. A collection of dependency parses obtained from a corpus allows you to figure out which edges matter and which do not.

Dependency parses also give a more holistic perception of what might be going on in natural language. That is, the notation

$$\overline{is} = (\overrightarrow{banana} \otimes \overrightarrow{fruit}) + (\overrightarrow{apple} \otimes \overrightarrow{green})$$

and

is: (banana- & fruit+) or (apple- & green+);

and

$$P(w_n = fruit|w_{n-1} = is,\ w_{n-2} = banana) + P(w_n = green|w_{n-1} = is,\ w_{n-2} = apple)$$

all represent the same knowledge, the dependency notation appears to be less awkward than viewing history as some Bayesian probability. The dependency notion focuses attention on a different part of the problem.

Another popular way to deal at least partly with the sparsity of the history tensor $P(w_n|h)$ is to use skip-grams. The idea recognizes that many of the edges of an *n*-gram will be zero, and so these edges can be skipped. This is not a bad approach, except that it is "simply typed": it does not leverage the possibility that different words might have different types (verb, noun, ...) and that this typing information delivers further constraints on the structure of the skip-gram. That is, the notion of subj-verb-obj not only tells you that your skip-gram is effectively a

3-gram, but also that the first and third words belong to a class called "noun", and the middle is a transitive verb. This sharply prunes the number of possibilities *before* the learning algorithm is launched, instead of *during* or *after*. The fact that such pruning is even possible is obscured by the notation and language of n-grams and the history $P(w_n|h)$.

A different stumbling block of the "history" approach is that it ignores "the future": the fact that the words might be said next have already influenced the choice of the words already spoken. This can be hand-waved away by stating that the history is creating a model of (hidden) mental states, and that this model already incorporates those, and thus is anticipating future speech actions. Although this might be philosophically acceptable to some degree, it again forces complexity onto the problem, when the complexity is not needed. If you've already got the document, look at all of it; go all the way to the end of the sentence. Don't arbitrarily divide it into past and future, and discard the future.

To summarize: dependency structures appear naturally; flattening them into sequences places one at a notional, computational and conceptual disadvantage, even if the flattening is conceptually isomorphic to the original problem. The tensor $P(w_n|h)$ may indeed encode all possible knowledge about the text in a rigorously Bayesian fashion; but it is unwieldy.

3. Quotienting

The intended interpretation for the graphs discussed in this paper is that they represent or are the result of capturing a large amount of collected raw data. From this data, one wants to extract commonalities and recurring patterns.

The core assumption being made in this section is that, when two local neighborhoods of a graph are similar or identical, then this reflects some important similarity in the raw data. That is, similarity of subgraphs is the be-all and end-all of extracting knowledge from the larger graph, and the primary goal is to search for, mine, such similar subgraphs.

Exactly what it means to be "similar" is not defined here; this is up to the user. Similarity could mean subgraph isomorphism, or subgraph homomorphism, or something else: some sort of "close-enough" similarity property involving the shape of the graph, the connections made, the colors, directions, labels and weights on the vertices or edges. The precise details do not matter. However, it is assumed that the user can provide some algorithm for finding such similarities, and that the similarities can be understood as a kind-of "equivalence relation".

*Example of similarity***.** To motivate this, consider the following scenario. One has a large graph, some dense mesh, and one decides, via some external decision process, that two vertices are similar. One particularly good reason to think that they are similar is that they share a lot of nearest neighbors. In a social graph, one might say they have a lot of friends in common. In genomic or proteomic data,

they may interact with the same kinds of genes/proteins. In natural language, they might be words that are synonyms, and thus get used the same way across many different sentences; specifically, the syntactic dependency parse links these words to the same set of heads and dependents. At any rate, one has a large graph, and some sort of equivalence operation that can decide if two vertices are the "same" are not in any way, shape, or form sequential or are "similar enough". Whenever one has an equivalence relation, one can apply it to obtain a quotient, of grouping together into an identity all things that are the same.

To make this even more concrete, consider this example from linguistics. Suppose, given a corpus, one has observed three sentences: "Mary walked home", "Mary ran home" and "Mary drove home". A dependency parse provides three seeds:

walked: Mary- & home+;
ran: Mary- & home+;
drove: Mary- & home+;

which seem to be begging for an equivalence relation that will reduce these to

walked ran drove: Mary- & home+;

Using a tensorial notation, one starts with

$$\overrightarrow{Mary} \otimes \overline{walked} \otimes \overrightarrow{\text{hom}e} + \overrightarrow{Mary} \otimes \overline{ran} \otimes \overrightarrow{\text{hom}e} + \overrightarrow{Mary} \otimes \overline{drove} \otimes \overrightarrow{\text{hom}e}$$

and applies the equivalence relation to obtain

$$\overrightarrow{Mary} \otimes (\overline{walked} + \overline{ran} + \overline{drove}) \otimes \overrightarrow{\text{hom}e}$$

The structure here strongly resembles the application of the distributive law of multiplication over addition. This distributivity property is one of the appeals of the tensor notation.

One can obtain a similar sense of distributivity by using the operator "or" to separate the Link Grammar style stanzas, and note that the change also appears to be an application of the distributive law of conjunction over disjunction.

This is illustrated pictorially below.

It need not be the case that an equivalence relation is staring us in the face, yet here, it is. The vertices "walked", "ran" and "drove" can be considered similar, precisely because they have the same neighbors. The upper graph can be simplified by computing a quotient,

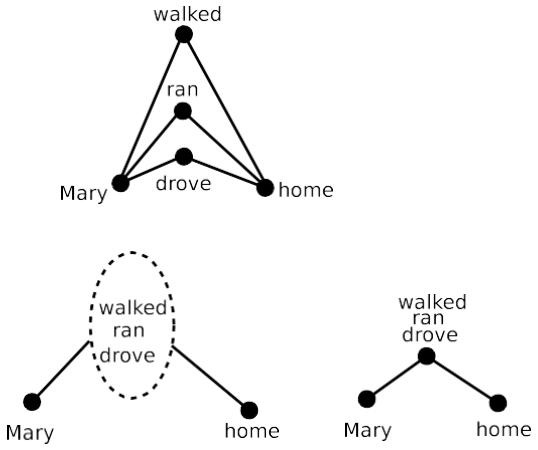


Figure 9. Creating a quotient graph

shown in the lower part: the quotient merges these three similar vertices into one. The result is not only a simpler graph, but also some vague sense that "walked", "ran" and "drove" are synonymous in some way.

Quotienting. If one has an equivalence relation that can be applied to a graph, then the obvious urge is to attempt to perform quotienting on the graph. That is, to create a new graph, where the "equal" parts are merged into one.

The first issue to be cleared out of the way is the use of the word "quotienting", which seems awkward, since the example above seemed to involve some sort of factoring, or the application of a distributive law of some sort. The terminology comes from modulo arithmetic, and is in wide use in all branches of mathematics. A simple example is the idea of dividing by three: given the set of integers Z, one partitions it into three sets: the set $\{0,3,6,9,\ldots\}$, the set $\{1,4,7,\ldots\}$ and the set $\{2,5,8,\ldots\}$. These three sets are termed the cosets of 0,1 and 2, and all elements in each set are considered to be equal, in the sense that, for any m and n in any one of these sets, it is always true that $m=n$ mod3: they are equal modulo 3. In this way, one obtains the quotient set $Z_3 = Z/3Z = Z/\text{mod } 3 = \{0, 1, 2\}$. Modulo arithmetic resembles division, ergo the term "quotient".

Given a set S and an equivalence relation $\sim$, it is common to write the quotient set as $Q = S/\sim$. In the above, S was Z and $\sim$ was mod 3. In general, one looks for and works with equivalence relations that preserve desirable algebraic properties of the set, while removing undesirable or pointless distinctions. In the modulo arithmetic example, addition is preserved: it is well defined and works as expected. In the linguistic example, the subj-verb-obj structure of the sentence is preserved; the quotienting removes the "pointless" distinction between different

verbs.

Quotienting is often described in terms of homomorphisms, functions $f : S \to Q$ that preserve the algebraic operations on S. For example, if m: $S \times S \times S \to S$ is a three-argument endomorphism on S, one expects that f preserves it: $f(m(a, b, c)) = m(f(a), f(b), f(c))$. For the previous example, if m was used to provide or identify a subj-verb-obj relationship, then, after quotienting, one expects that m can still identify the verb-slot correctly.

Graph quotients. In graph theory, the notion of quotienting is often referred to as working "relative to a subgraph". Given a graph G and a subgraph $A \subset G$, one "draws a dotted line" or places a balloon around the vertices and edges in A, but preserves all of the edges coming out of A and going into G. The internal structure of A is then ignored. The equivalence relation makes all elements of A equivalent, so that A behaves as if it were a single vertex, with assorted edges attached to it, running from A to the rest of G.

Stalks. Given the above notion of a graph quotient, it can be brought over to the language of seeds and sections established earlier.

Given two vertices v_a and v_b, let s_a and s_b be the corresponding seeds, as defined previously. That is, $s = (v, E_v)$ with E_v being the set of edges connecting v to all of its nearest neighbors. Consider now creating the object $(\{v_a, v_b\}, E_{ab})$. This is no longer a seed, as the first item is no longer a single vertex, but a set of vertices. The set E_{ab} is still a set of edges, depending on the two initial sets of edges E_a and E_b. The precise definition of E_{ab} is not given: it might be the union of E_a and E_b, or the intersection, or some other function. In general, one writes $E_{ab} = f(E_a, E_b)$ for some function f. It is typically desirable for f to be both commutative and associative, so that $f(E_a, E_b) = f(E_b, E_a)$ and $f(f(E_a, E_b), E_c) = f(E_a, f(E_b, E_c)) = f(E_a, E_b, E_c)$.

The mashing together of several vertices creates a structure resembling a graph quotient, as described above. This will be called a stalk in what follows. The stalk is defined in such a way that it is not actually a graph quotient; it merely resembles one. It is a partial step on the way to a graph quotient. The definition asks that the internal structure of the stalk be preserved. One obtains a true graph quotient only after projecting the stalk down to a base space.

Definition. A STALK is an ordered pair $S = (V, E)$ of vertices and edges such that every edge in E has one endpoint being a vertex in V and the other endpoint being a vertex not in V. That is, each edge in E is a connector, and no edge in E is a link (back into V).

This definition of a stalk is meant to be a straightforward generalization of the previously defined seed, replacing the germ vertex by a germ that is a set of vertices. Stalks can be linked together, much as seeds are:

Definition. A LINK between two different stalks $S_a = (V_a, E_a)$ and $S_b = (V_b, E_b)$ is any edge $e = (v_1, v_2)$ running between them, *viz.* where $v_1 \in V_a$

and $v_2 \in V_b$ and $e \in E_a$ and $e \in E_b$. Two stalks are CONNECTED when there are one or more links between them.

It is convenient (it is suggested) that the vertices in the stalk be visualized as being stacked one on top another, forming a tower or a fiber, with the edges sticking out as spines. Perhaps one can visualize a kind-of melted stack of jigsaw-puzzle pieces. This visualization is suggested only to enforce the idea that two different stalks project down to two different base-points. In particular, one now can have the notion of a meta-graph where each stalk is a vertex, and each link is an edge. That is, if one flattens the meta-graph down to two dimensions, then one can imagine a stalk growing up as a pole above each meta-vertex, and each meta-edge as being the projection of a link between two stalks. To maintain consistency with standard mathematical terminology, this meta-graph should really be called a "base space", and the stalks and links project down onto it in the usual sense.

The notion of projection can be formalized.

Definition. A projection π is a function that accepts a stalk $S = (V, E)$ and produces a seed $s = (b, E')$ such that for each $v \in V$ it produces b, and for each edge $(v, w) \in E$ it produces the edge $(b, w) \in E'$. That is $\pi(v,(v, w)) = (b,(b, w))$. By abuse of notation, one may write $\pi(v) = b$ so that $\pi(v,(v, w)) = (\pi(v), \pi(v), w)$.

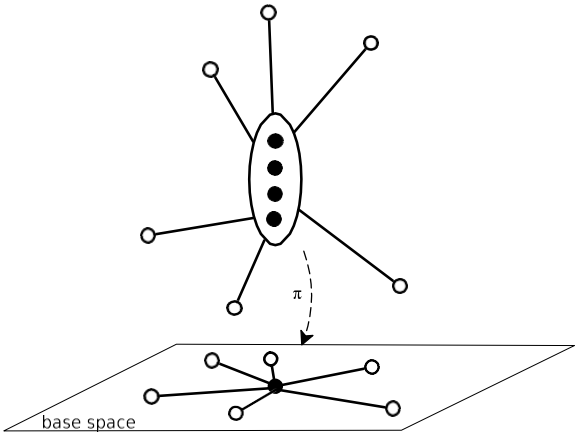


Figure 10. A stalk and its projection

The desirable properties of the projection can be summarized in a lemma:

Lemma. *The projection function π is a morphism of graphs, taking a graph and mapping it to a quotient graph.* ◇

The proof of the lemma is left to the reader, in the hopes that it is self-evident. An actual proof requires a few additional definitions, as follows.

The projection down to a base space suggests that the equivalence relation on

vertices can be extended to an equivalence relation on edges: two edges are equivalent if they form the same link. That is, one has an equivalence class of edges:

Definition. A LINK between two different stalks $S_a = (V_a, E_a)$ and $S_b = (V_b, E_b)$ is the set $l = \{e_k\}$ of all edges e_k that connect some pair of vertices in V_a and V_b. That is, every $e_k = (v_{k1}, v_{k2})$ in L has the property that $v_{k1} \in V_a$ and $v_{k2} \in V_b$ and $e_k \in E_a$ and $e_k \in E_b$.

This redefines the notion of a link. Perhaps it should be given a different name, but it should be OK, because the intended sense should be clear from the context. This allows us to redefine the notion of a stalk as well:

Definition. A STALK is an ordered pair $S = (V, L)$ of vertices and links such that every link in L has one endpoint that is V and the other endpoint not being V. That is, each link in L is a connector or half-edge. ◇

One then has an obvious lemma about the projection.

Lemma. *The projection function π can be extended consistently to the revised definition of a stalk.*

As before, the proof is left as being obvious.

Sheaves

The above definitions, along with the projection function, indicate that stalks can be thought of as seeds. The stalk preserves the graph structure; the projection creates graph quotients. The above finally allows the definition of a section to be understood in a way that is in keeping with the usual notion of a section as commonly defined in covering spaces and fiber bundles.

Figure 11. Corn

Definition. A sheaf is a graph G represented as a collection of seeds, together with a projection function π that can be taken to be an equivalence relation. That is, π maps G to a quotient graph G/π such that, for each pair of vertices $v, w \in G$, one has $\pi(v) = \pi(w)$ if and only if v, w are in germs in the same stalk and, for any pair of edges $e, e' \in G$, one has $\pi(e) = \pi(e')$ if and only if the pair e, e' are connectors in the same stalk.

The formal definition of a sheaf also requires that it obeys a set of axioms, called the gluing axioms. Before giving these, it is useful to look at an example.

Example: collocations. A canonical first step in corpus linguistics is to align

text around a shared word or phrase:

```
                      fly like a butterfly
  airplanes that      fly
                      fly fishing
                      fly away home
                      fly ash in concrete
  when sparks         fly
  let's               fly a kite
  learn to            flyhelicopters
```

Each word is meant to be a vertex; edges are assumed to connect the vertices together in some way. In standard corpus linguistics, the edges are always taken to join together neighboring words, in a sequential fashion. Note that each phrase in the collocation obeys the formal definition of a section given above. It does so trivially: it is just a linear sequence of vertices connected with edges. If the collocated phrases are chopped up so that they form a word-sequence that is exactly n words long, then one calls that sequence an n-gram.

The projection function π is now also equally plain: it simply maps all of the distinct occurrences of the word "fly" down to a single, generic word "fly". The stalk is just the vertical arrangement of the word "fly", one above another. Each phrase or section can be visualized as a botanical branch or botanical leaf branching off the central stalk. The projection of the stalk is shown below. The words in each phrase are connected as a linear sequence. Identical words are projected down to a common base point.

The sections do not have to be linear sequences; the phrases can be parsed with a dependency parser of one style or another, in which case the words are joined with (directed) edges that denote dependencies. Parsing with a head-phrase parser introduces additional vertices, typically called NP, VP, S, and so on. The next figure shows the projection that results from alignment on an (unlabeled, undirected) dependency parse of the text. Most noticeable is that the determiner "a" does not link to "fly" even though it stands next to it; instead, the determiner links to the noun it determines. This figure also shows "ash" as modifying "fly", which, as a dependency, is not exactly correct but does serve to illustrate the difference between the N-gram and the dependency alignment.

Both of these figures represent a quotient graph that results from a corpus alignment, where all uses of a word have been collapsed (projected down) to a single node. The resulting graph is the graph of the language.

Why sheaves? Are projections useful? Yes. A collapsed graph like that might appear strange; why would one want to do that, if one has individual sentence data?

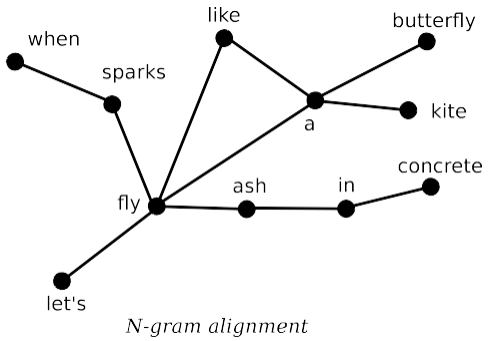


Figure 12. N-gram corpus text alignment

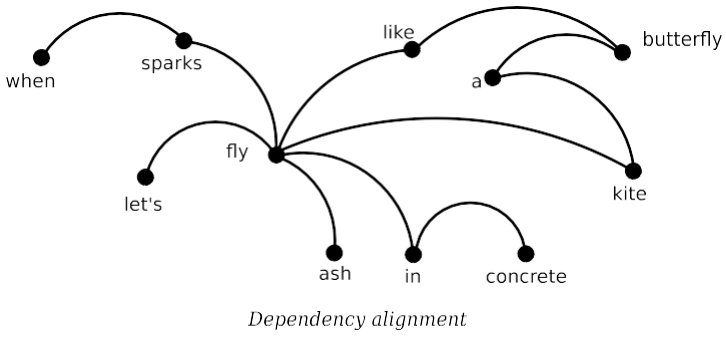


Figure 13. Dependency parse corpus text alignment

By collapsing in this way, one obtains a natural place to store marginal distributions. For example, when accumulating statistics for large collections of sentences, the projected vertex becomes an ideal place to store the frequency count of that word; the edge becomes an excellent place to store the joint probability or the mutual information for a pair of words. The projected graph – the quotient graph – is manageable in size. For example, in a corpus consisting of ten million sentences, one might see 130K distinct, unique words (130K vertices) and perhaps 5 million distinct word-pairs (5M edges). Such a graph is manageable and can fit into the RAM of a contemporary computer.

By contrast, storing the individual parses for 10 million sentences is more challenging. Assuming 15 words per sentence, this requires storing 150M vertices and approximately 20 links per sentence for 200M edges. This graph is two orders

of magnitude larger than the quotient graph. It makes sense to collapse the stalks down to their projections as soon as possible. They can be envisioned to still be there, virtually, in principle, but the actual storage can be avoided.

Figure 14. A sheaf of stalks; a sheaf of paper

Every graph can be represented as an adjacency matrix. In this example, it would be a sparse matrix with 5 million non-zero entries out of 130K total. The marginals stored with the graph can be accessed as marginals in the normal sense of values written in the margin of the matrix. Standard linear-algebra tools, such as the R programming language, can access the matrix and the marginals.

One way of visualizing the sheaf is as a stack of sheets of paper, with one sentence written on each sheet. The papers are stacked in such a way that words that are the same are always vertically above one-another. This stacking is where the term "sheaf" comes from. Each single sheet of paper is a section. Each collocation is a stalk.

A different example can be taken from biochemistry. There, one might want to write down specific pathways or interaction networks on the individual sheets of paper, treating them as sections. If one specific gene is up-regulated, one can then try to view everything else that changed as belonging to the same section, as if it were an activation mode within the global network graph of all possible interactions. Thus, for example, the Krebs cycle can be taken to be a single section through the network: it shows exactly which coenzymes are active in aerobic metabolism. The same substrates, products and enzymes may also participate in other pathways; those other pathways should be considered as other sections through the sheaf. Each substrate, enzyme or product is itself a stalk. Each reaction type is a seed.

The sheaf, its decomposition into sections and its projection down to a single base unified network, provides a holistic view of a network of interactions.

Activations or modes of the network correspond to grammatically valid sentences, when the network is a language network. Sections correspond to activated biological pathways, when the network is a map of biochemical interactions.

Gluing Axioms. Sections can be cut down, or they can be enlarged, yet each section must consist of a grammatically-correct sentence, phrase (or paragraph, or larger! ... or smaller, just a sentence fragment). The base space consists simply of individual words in a language, each word being a vertex, connections between words being projections of the connections discovered from the N-grams, or dependency graphs, as the case may be. One can examine how the language structure changes as words are removed from the base space: such changes correspond to the "restriction morphisms" of a sheaf. These restriction morphisms obey all of the axioms of a sheaf; this observation is what drives the peculiar naming convention given here. The reason this "works", that the axioms apply, is in fact rather shallow: it is because the seeds, as initially defined, behave very much like open sets, and when they are projected to the base space, they serve to cover the base space, much as a topological covering does.

The restriction morphisms appear to continue to satisfy the sheaf axioms even after projection (at least, at the informal level given here).

Why sheaves? The primary reason for introducing this notion is to consolidate the otherwise vague idea of a "language graph". One has dueling notions: the graph of all sentences; the generative power of grammars. Surface realizations of language are studied in corpus linguistics, where differences in regional dialects, differences according to socio-economic status and politically motivated differences are found. However, these surface realizations are almost never refined into a grammar, and thus, one does not obtain a generative model of how different speakers in different socio-economic classes speak: corpus linguistics examples are just that: examples that are not further refined. By applying a pattern mining approach, the underlying grammar can be discovered computationally. But what is it, really, that is being discovered, other than some collection of grammatical classes and relations? By looking at the collection of all words, sentences and paragraphs in a corpus as if it were a sheaf, one gets a more "holistic" view of what language is: one can start seeing a "big picture" instead of just the trees. This holistic view is the primary point of this exercise.

Related ideas. As before, the "sections" presented above (sentence fragments) are presented minimally. In practice, sections will be adorned with additional information, such as frequency counts and mutual information values. Once clustering and quotienting have been performed, non-trivial type tags become available.

4. Clustering

typical bullshit

Why clustering? foo
x
x

By contrast, the goal here is not to talk about a graph *G* relative to just a single *A*, but relative to a huge number of different *A*'s. What's more, the internal structure of these *A*'s will continue to be interesting, and so is carried onwards. Finally, the act of merging together multiple vertices into one *A* may result in some of the existing edges being cut or new edges being created. The clustering operation applied to the graph alters the graph structure. These considerations are what makes it convenient to abandon the traditional graph theory and to replace it by the notion of sheaves and sections.

The above establishes a vocabulary, a means for talking about the clustering of similar things on graphs. It does not suggest how to cluster. Without this vocabulary, it can be very confusing to visualize and talk about what is meant by clustering on a graph. It is worth reviewing some examples.

- In a social graph, a cluster might be a clique of friends. By placing these friends into one group, the stalk allows you to examine how different groups interact with one another.
- In proteomic or genomic data, if one can group together similar proteins or genes into clusters, one can accomplish a form of dimensional reduction, simplifying the network model of the dataset. It provides a way to formalize network construction, without the bad smell of ad-hoc simplifications.
- In linguistic data, the natural clustering is that of words that behave in a similar syntactic fashion; such clusters are commonly called "grammatical classes" or "parts of speech". In particular, it allows one to visualize language as a graph. So: consider, for example, the set of all dependency parses of all sentences in some corpus, say Wikipedia. Each dependency parse is a tree; the vertices are words, and the edges are the dependencies. Taken as a graph, this is a huge graph, with words connecting to other words, all over the place. It is not terribly interesting in this raw state, because it is overwhelmingly large. However, we might notice that all sentences containing the word "dish" resemble all sentences containing the word "plate"; that these two words always get used in a similar or the same way. Grouping these two words together into one reduces the size of the graph by one vertex. Aggressively merging similar words together can sharply shrink the size of the graph to a manageable size. One gets something more: the resulting graph can be understood as encapsulating the structure of the English language.

This last example is worth expanding on. Two things happen when the compressed graph is created. First, that graph encodes the syntactic structure of the language: the links between grammatical classes indicate how words can be

arranged into grammatically correct sentences. Second, the amount of compression applied can reveal different kinds of structures. With extremely heavy compression, one might discover only the crudest parts of speech: determiners, adjectives, nouns, transitive and intransitive verbs. All these classes are distinct, because they link differently. However, if instead, a lot less compression is applied, then one can discover synonymous words: so, "plate" and "dish" might be grouped together, possibly with "saucer", but not with "cup". Here, one is extracting a semantic grouping, rather than a syntactic grouping.

So, the answer to "why clustering?" is that it allows information to be extracted from a graph, and encoded in a useful, usable fashion. No attempt is made here to suggest how to cluster; merely, that if an equivalence relation is available and if it is employed wisely, then one can construct quotient graphs that encode important relationships of the original, raw graph.

Similar concepts. One can think of a stalk as a kind of hypergraph, but this view does not seem to be particularly productive.

5. Types

It is notationally awkward to have to write stalks in terms of the sets of vertices that they are composed of; it is convenient to instead replace each set by a symbol. The symbol will be called a TYPE. As it happens, these types can be seen to be the same things occurring in the study of type theory; the name is justified.

The core idea can be illustrated with Link Grammar as an example. The Link Grammar disjuncts *are* one and the same thing as stalks. It is worth making this very explicit. A subset of the Link Grammar English dictionary looks like this:

```
cat dog: D- & S+;
the a: D+;
ran: S-;
```

This states that "cat" and "dog" are both vertices, and they are in the same stalk. That stalk has two connectors: D- and S+, which encode the other stalks that can be connected to. So, the D+ can be connected to the D- to form a link. The link has the form ({the, a},

{cat, dog}) and the connector symbols D+ and D- act as abbreviations for the vertex sets that the unconnected end can connect to. The + and - symbols indicate directionality: to the right or to the left. They capture the notion that, in English, the word-order matters. To properly explain the + and -, we should have to go back to the definition of a graph on the very first page and introduce the notion of left-right order among the vertices. Doing so from the very beginning would do nothing but clutter up the presentation, so that is not done. The reader is now invited to treat the initial definition of the graph as a monad: there are additional

details "under the covers", but they are wrapped up and ignored, and only the relevant bits are exposed. Perhaps the vertices had a color. Perhaps they had a name, or a numerical weight; this is ignored. Here, we unwrap the idea that the vertices must be organized in a left-right order. It is sufficient, for now, to leave it at that.

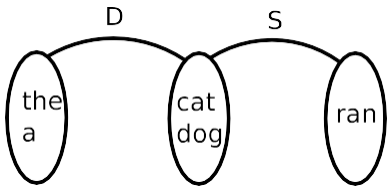


Figure 15. Three stalks and two typed links

The three stalks here encode a set of grammatically valid English language sentences. Hooking together the S- and S+ connectors to form an S link, one obtains the sequence [{the, a} {cat, dog} {ran}]. This can be used to generate grammatically valid sentences: pick one word from each set, and one gets a valid sentence. Alternatively, this structure can be taken to encode the sum-total knowledge about this toy language: it is a kind-of graphical representation of the entire language, viewed as a whole.

Definition. Given a stalk $S = (V, L)$, the CONNECTOR TYPE of L is a symbol that can be used as a synonym for the set L. It serves as a short-hand notation for L itself. ◇

Just as in type theory, a type can be viewed as a set. Yet, just as in type theory, this is the wrong viewpoint: a type is better understood as expressing a property: it is an intensional, rather than an extensional description. Formally, in the case of finite sets, this may feel like splitting hairs. For an intuitive understanding, however, it is useful to think of a type as a property carried by an object, not just the class that the object can be assigned to.

Why types? Types are introduced here primarily as a convenience for working with stalks. They are labels, but they can be useful. Re-examining the examples:

- In a social graph, one group of friends might be called "students" and another group of friends might be called "teachers". The class labels are useful for noting the function and relationship of the different social groups.
- In a genetic regulatory network, sub-networks can be classified as "positive regulatory pathways" or "negative regulatory pathways" with respect to the activation of a particular gene.

These examples suggest that the use of types is little more than a convenient labeling system. In fact, more can be made here, as types interact strongly with category theory: types are used to describe the internal language of monoidal categories. But this is a rather abstract viewpoint, of no immediate short-term use. Suffice it to say that appearance of types in grammatical analysis of a language is not accidental.

What kind of information do types carry? The above example oversimplifies the notion of types, presenting them as a purely syntactic device. In practice, types also carry semantic information. The amount of semantic information varies inversely to the broadness of the type. In language, coarse-grained types (noun, verb) carry almost no semantic information. Fine-grained types carry much more: a "transitive verb taking a particle and an indirect object" is quite specific: it must be some action that can be performed on some object using some tool in some fashion. An example would be "John sang a song to Mary on his guitar": there is a what, who and how yoked together in the verb "sang". The more fine-grained the classification, the more semantic content is contained in it.

This suggests that the proper approach is hierarchical: a fine-grained clustering, that captures the semantic content, followed by a coarser clustering, that erases much of this, leaving behind only the syntactic" content.

6. Parsing

The introduction remarked that not every collection of seeds can be assembled in such a way as to create a valid graph. This idea can be firmed up and defined more carefully. Generically, a valid assembly of seeds is called a parse, and the act of assembling them is called parsing, which is done by parse algorithms. To illustrate the process, consider the following two seeds:

$v_2 : \{(v_2, v_1), (v_2, v_3)\}$
$v_3 : \{(v_3, v_2)\}$

Represented graphically, these seeds are

Figure 16. Two unconnected seeds

The connector (half-edge) (v_2, v_3) appears with both polarities and can be linked together to form a link. The connector (v_2, v_1) has nothing to connect to. Even after maximally linking these two seeds, one does not obtain a valid graph:

the vertex v_1 is missing from the vertex set of the graph, even though there is an edge ready to attach to it. This provides an example of a failed parse. It is enough to add the seed v_1: (v_1, v_2) to convert this into a successful parse. Adding this seed and then attempting to maximally link it results in a valid graph; the parse is successful.

Note the minor change in notation: the colon is used as a separator, with the germ appearing on the left and the set of connectors on the right. The relevance of this notational change becomes more apparent, if we label the vertices in a funny way: let v_1 carry the label "the", v_2 carry the label "dog" and v_3 carry the label "ran". The failed parse is meant to illustrate that "dog ran" is not a grammatically valid sentence, whereas "the dog ran" is.

Converting these seeds to also enforce the left-right word order requires the notation

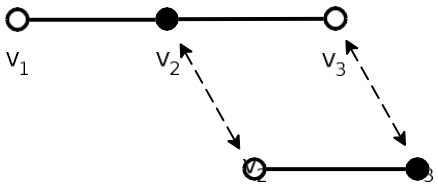


Figure 17. Parsing is the creation of links

the: {(the, dog+)}
dog: {(dog, the-), (dog, ran+)}
ran: {(ran, dog-)}

This notation is verbose and slightly confusing. Repeating the germ as the first vertex in every connector is entirely unnecessary. Write instead:

the: { dog+ }
dog: { the-, ran+}
ran: { dog- }

The set-builder notation is unneeded and perhaps slightly confusing. In particular, the word "dog" has two connectors on it; both must be connected to obtain a valid parse. The ampersand can be used to indicate the requirement that both connectors are required. This notation will also be useful in the next section.

the: dog+ ;
dog: the- & ran+;
ran: dog-;

This brings us almost back to the previous section, but not quite. Here, we are working with seeds; previously we worked with stalks. Here, the connector type labels were not employed. In the real-world use-cases, using stalks and type labels is much more convenient.

This now brings us to a first draft of a parse algorithm. Given an input set of vertices, it attempts to find a graph that is able to connect all of them.

(1) Provide a dictionary D consisting of a set of unconnected stalks.
(2) Input a set of vertices $V = \{v_1, v_2, \ldots, v_k\}$.
(3) For each vertex in V, locate a stalk which contains that vertex in its germ.
(4) Attempt to connect all connectors in the selected stalks.
(5) If all connectors can be connected, the parse is successful; else the parse fails.
(6) Print the resulting graph. This graph can be described as a pair (V, E) with V the input set of vertices, and E the set of links obtained from fully connecting the selected stalks.

The above algorithm is "generic" and does not suggest any optimal strategy for the crucial steps 3 or 4. It also omits discussion of any further constraints that might need to be applied: perhaps the edges need to be directed; perhaps the resulting graph must be a planar graph (no intersecting edges); perhaps the graph must be a minimum spanning tree; perhaps the input vertices must be arranged in a linear order. These are additional constraints that will typically be required in some specific application.

Why parsing? The benefit of parsing for the analysis of the structure of natural language is well established. Thus, an example of parsing in a non-linguistic domain is useful. Consider having used the above graph compression or vertex-edge clustering techniques to obtain a collection of stalks that describe genomic interactions. This collection provides the initial dictionary D. Now imagine a process where a certain specific set of genes are associated with some particular trait or reaction. Is this a complete set? Can it be said that their interactions are fully understood?

One way to answer these last two questions would be to apply the parse algorithm, using the known dictionary, to see if a complete interaction network can be obtained. If so, then this new specific gene-set fits the general pattern. If not, if a complete parse cannot be found, then one strongly suspects that there remain one or more genes, yet undetermined, that also play a role in the trait. To find these, one might examine the stalks that might have been required to complete the parse: these will give hints as to the specific type of gene, or the style of interaction to search for.

Thus, parsing new gene expressions and pathways offers a way of discovering whether they resemble existing, known pathways, or whether they are truly novel. If they seem novel, parsing also gives strong hints as to where to look for any

missing pieces or interactions.

Is this really parsing? The above description of parsing is sufficiently different from standard textbook expositions of natural language parsing that some form of an apology needs to be written.

The first step is to observe that the presented algorithm is essentially a simplified, generalized variation of the Link Grammar parsing algorithm [5]. The generalization consists in the removal of word-order and link-crossing constraints.

The second step is to observe that the theory of Link Grammar is more or less isomorphic to the theory of pregroup grammars [3] (see Wikipedia), the primary differences being notational. The left-right directional Link Grammar connectors correspond to the left and right adjoints in a pregroup. A Link Grammar disjunct (that is, a seed) corresponds to a sequence of types in a pregroup grammar. The correspondence is more or less direct, except that Link Grammar is notationally simpler to work with.

The third step is to observe that the Link Grammar is a form of dependency grammar. Although the original Link Grammar formulation uses undirected links, it is straight-forward and unambiguous to mark up the links with head-dependent directional arrows.

The fourth step is to realize that dependency grammars (DG) and head-phrase-structure grammars (HPSG) are essentially isomorphic. Given one, one can obtain the other in a purely mechanistic way.

The final step is to realize that most introductory textbooks describe parsers for a context-free grammar, and that, for general instructional purposes, such parsers are sufficient to work with HPSG. The primary issue with HPSG and context-free language parsers is that they obscure the notion of linking together pieces; this is one reason why dependency grammars are often favored: they make clear that it is the linkage between various words that has a primary psychological role in the human understanding of language. It should be noted that many researchers in the psychology of linguistics are particularly drawn to the categorial grammars; these are quite similar to the pregroup grammars and are more closely related to Link Grammar than to the phrase-structure grammars.

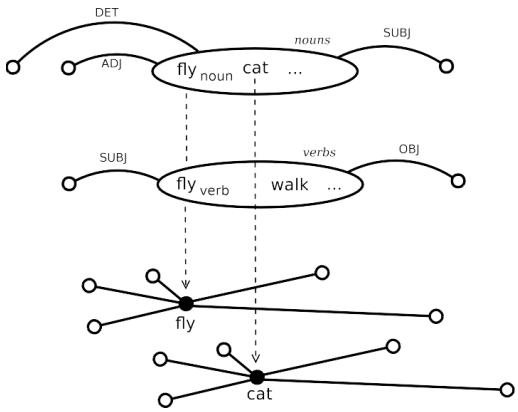


Figure 18. Polymorphism

This figure illustrates a polymorphic assignment for the word "fly". It is split into two parts: the first, a noun, classed with other nouns, showing labeled connectors to determiners, adjectives, and a connector showing that nouns can act as the subject of a verb. The second class shows labeled connectors to subjects and objects, as is appropriate for transitive verbs. Underneath are the flattened raw seeds, showing the words "fly" and "cat" and the myriad of connectors on them. The flattened seeds cannot lead to grammatical linkages, as they mash together into one the connectors for different parts of speech.

7. Polymorphism

Any given vertex may participate in two or more seeds, independently from one-another. It is this statement that further sharpens the departure from naive graph theory. This is best illustrated by a practical example.

Consider a large graph constructed from a large corpus of English language sentences. As subgraphs, it might contain the two sentences "A big fly landed on his nose" and "It will fly home". The vertex "fly" occurs as a noun (the subject, with the determiner and adjective) in one sentence, and a verb (with the subject and object) in the other. Suppose that the equivalence relation described in the clustering section also has the power to discern that this one word should really be split into two, namely fly_{noun} and fly_{verb}, and placed into two different stalks, namely, in the "noun" stalk in the first case, and the "verb" stalk in the second. Recall that these two stalks must be different, because the kinds of connectors that are allowed on a noun must necessarily be quite different from those on a verb. One is then led to the image shown in Figure 18.

The point of the figure is to illustrate that, although the "base graph" may not

distinguish one variant of a vertex from another, it is important to discover, extract and represent this difference. The concept of "polymorphism" applies, because the base vertex behaves as one of several distinct types in practice. There are several ways the above diagram can be represented textually. As before, the Link Grammar-style notation is used, as it is fairly clear and direct. One representation would be to expose the polymorphism only in the connectors, and not in the base vertex label:

```
fly:(DET- & ADJ- & SUBJ+) or (SUBJ- & OBJ+);
```

A different possibility is to promptly split the vertex label into two, and ignore the subscript during the parsing stage:

```
fly.noun cat: (DET- & ADJ- & SUBJ+);
fly.verb walk: (SUBJ- & OBJ+);
```

Either way, the non-subscripted version of *fly* behaves in a polymorphic fashion.

Note that the use of the notation "or" to disjoin the possibilities denotes a choice function, and not a boolean "or". That is, one can choose either one form, or the other; one cannot choose both. During the parse, both possibilities need to be considered, but only one selected in the end. This implies that at least some fragment of linear logic is at play, and not boolean logic (this should be expanded upon in future drafts).

Similar concept: part of speech. It is tempting to identify the connectors DET, ADJ, SUBJ, OBJ in the diagrams above with "parts of speech". This would be a mistake. In conventional grammatical analysis, there are half-a-dozen or a dozen parts of speech that are recognized: noun, verb, adjective, and so on. By contrast, these connector types indicate a grammatical role. That is, the disjunct `SUBJ- & OBJ+` indicates a word that takes both a subject and an object: a transitive verb. That is, the disjunct is in essence a fine-grained part of speech, indicating not only verb-ness, but the specific type of verb-ness (transitive).

The Link Grammar English dictionary documents more than 100 connector types; these are subtyped, so that approximately 500 connectors might be seen. These connectors, when arranged into disjuncts, result in tens of thousands of disjuncts. That is, Link Grammar defines tens of thousands of distinct "parts of speech". They can be thought of as parts of speech, but they are quite fine-grained, far more fine-grained than any text on grammar might ever care to list.

If one uses a technique, such as MST parsing [13], and then extracts disjuncts, one might observe more than 6 million disjuncts and 9 million seeds on a vocabulary of 140K words. These are, again, in the above technical sense, just "parts of speech", but they are hyperfine-grained. The count is overwhelming. So, although it is technically correct to call them "parts of speech", it is a conceptual

error to think of a class that has six million representatives as if it were a class with a dozen members.

Similar concept: skip-grams. The N-gram [11] and the more efficient skip-gram [14] models of semantic analysis provide somewhat similar tools for understanding connec-tivity, and differentiating different forms of connectivity. In a skip-gram model, one might extract two skip-grams from the above example sentences: "a fly landed" and "it fly home". A clustering process, such as adagram or word2vec, might be used to classify these two strings into distinct clusters, categorizing one with other noun-like words, and the other with verb-like words.

The N-gram or skip-gram technique works only for linear, sequenced data, which is sufficient for natural language, but cannot be employed in a generic non-ordered graphical setting. To make this clear: a seed representation for the above would be: "fly: a- landed+" indicating that the word "a" (written as the connector "a-") comes sequentially before "fly", while the word "landed" (written as the connector "landed+") comes after.

The other phrase has the representation "fly: it- home+". These two can now be employed in a clustering algorithm, to determine whether they fall into the same, or into different categories. If one treats the skip-grams and the seeds as merely two different representations of the same data, then applying the same algorithm to either should give essentially the same results.

The seed representation, however, is superior in two different ways. First, it can be used for non-sequential data. Second, by making clear the relationship between the vertex and its connectors, the connectors can be treated as "additional data", tagging the vertex carrying additional bits of information. That additional information is manifested from the overall graph structure and is explicit. By contrast, untagged N-grams or untagged skip-grams leave all such structure implicit and hidden.

Polymorphism and semantics. The concept of polymorphism introduced above lays a foundation for semantics, for extracting meaning from graphs. This is already hinted at by the fact that any English-language dictionary will provide at least two different definitions for "fly": one tagged as a noun, the other as a verb. The observation of hyperfine-grained parts of speech can push this aggressively farther.

In a modern corpus of English, one might expect to observe the seeds "apple: green and "apple: iphone+". The disjuncts "green-" and "iphone+" can be interpreted as a kind-of tag on the word "apple". Since there are exactly two tags in this example, they can be viewed as supplying exactly one bit of additional information to the word "apple". Effectively, a single apple has been split into two distinct apples. Are they really distinct, however? This can only be judged on the basis of some clustering algorithm that can assign tagged words to classes. Even very naive, unsophisticated algorithms might be expected to classify these two different kinds of apple into different classes; the extra bit of information carried by the disjunct is a bit of actual, usable information.

To summarize: the arrangement of vertices into polymorphic seeds and sections

enables the vertices to be tagged with extra information. The tags are the connectors themselves: their presence or absence carries information. That extra information can be treated as "semantic information", identifying different types or kinds, rather than as purely syntactic information about arrangements and relationships.

8. Conclusion

This paper presents a way of thinking about graphs that allows them to be decomposed into constituent parts fairly easily and then brought together and reassembled in a coherent, syntactically correct fashion. It does so without having to play favorites among competing algorithmic approaches and scoring functions. It makes only one basic assumption: that knowledge can be extracted at a symbolic level from pair-wise relationships between events or objects.

It touches briefly, all too briefly, on several closely related topics, such as the application of category theory and type theory to the analysis of graph structure. These topics could be greatly expanded upon, possibly clarifying much of this content. It is now known to category theorists that there is a close relationship between categories, the internal languages that they encode, and that these are reflections of one another, reflecting through a theory of types. A reasonable but incomplete reference for some of this material is the HoTT book. It exposes types in greater detail, but does not cover the relationship between internal languages, parsing, and the modal logic descriptions of parsing. It is possible that there are texts in proof theory that cover these topics, but I am not aware of any.

This is a bit unfortunate, since I feel that much or most of what is written here is "well known" to computational proof theorists; unfortunately, that literature is not aimed at the data-mining and machine-learning crowd that this paper tries to address. Additions, corrections and revisions are welcomed.

References

[1] Sleator D., Temperley D. Parsing English with a link grammar // Tech. Rep. Carnegie Mellon University Computer Science technical report CMU-CS-91-196. – 1991. : http://arxiv.org/pdf/ cmp-lg/9508004.

[2] Aron J. Quantum links let computers read // New Scientist. – 2010. – Vol. 208, No 2790.:http://www.cs.ox. ac.uk/people/bob.coecke/NewScientist.pdf

[3] Kartsaklis D., Sadrzadeh M. A study of entanglement in a categorical framework of natural language // Proceedings Quantum Physics and Logic, Electronic Proceedings in Theoretical Computer Science. – 2014. – No 172. – P. 249–260.:

https://arxiv.org/abs/1405.2874

[4] Baez J. C., Stay M. Physics, topology, logic and computation: A rosetta stone. // Arxiv/abs/0903.0340, 2009.: http://math.ucr.edu/home/baez/rosetta.pdf.

[5] Sleator D. D., Temperley D. Parsing English with a link grammar. // Proc. Third International Workshop on Parsing Technologies. – 1993. – P. 277–292.: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/link/pub/www/papers/ps/LG-IWPT93.ps.

[6] Poon H., Domingos P. Unsupervised semantic parsing // Proc. of the 2009 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Singapore. – 2009. – P. 1–10.: http://www.aclweb.org/anthology/D09-1001.

[7] Lambek J. On the calculus of syntactic. – 1961. – P. 166-178.

[8] Marcus S. Algebraic Linguistics Analytical Models, 1967.: https://monoskop.org/images/2/26/Marcus_Solomon_editor_Algebraic_Linguistics_Analytical_ Models_1967.pdf

[9] Kartsaklis D., Sadrzadeh M., Pulman S., Coecke B. Reasoning about meaning in natural language with compact closed categories and frobenius algebras. Logic and Algebraic Structures in Quantum Computing .–2013. – P. 1-21.: https://www.cs.ox.ac.uk/files/ 5468/sadrzadeh_kartsaklis.pdf.

[10] Pentus M. Lambek calculus and formal grammars // American Mathematical Society Translations. – 1998. – P. 1-31 : http://lpcs.math.msu.su/~pentus/ftp/papers/ams.pdf.

[11] Rosenfeld R. A maximum entropy approach to adaptive statistical language modeling // Computer Speech & Language. – 1996. – No 10. – P. 187–228.: http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/roni/papers/me-csl-revised.pdf.

[12] Ash R. B. Information Theory . – NY: Dover Publications, 1965.

[13] Yuret D. Discovery of linguistic relations using lexical attraction : PhD thesis. – MIT, 1998. : http://www2.denizyuret.com/pub/yuretphd.html.

[14] Guthrie D., Allison B., Liu W., Guthrie L., Wilks Y. A closer look at skip-gram modeling // Proc. of the Fifth international Conference on Language Resources and Evaluation. – 2006. – P.1–4.: https://homepages.inf.ed.ac.uk/ballison/pdf/lrec_skipgrams.pdf.