Semantic Graph Compression with Hypergraphs

Arber Borici
Department of Computer Science
University of Victoria
Victoria, Canada
borici@uvic.ca

Alex Thomo
Department of Computer Science
University of Victoria
Victoria, Canada
thomo@cs.uvic.ca

***Abstract*—Can we model complex networks as hypergraphs and compress them for faster storage, transmission, and mining of data? In this paper, we propose a modeling and compression technique that consists of two phases: (i) mapping networks to hypergraphs by exploiting inherent or structural semantic features; and (ii) partitioning the resulting hypergraph such that similar nodes are grouped into a number of possibly disconnected parts. The partitioned hypergraph is then processed in order to yield more structural redundancy to increase compression. We provide empirical results that compare the proposed method to random and natural orderings of select real networks using an information-theoretic measure. When modeling networks using hypergraphs as proposed here, the potential for compactness and compression increases, as observed in our experimental evaluation. This benefits a variety of domains in a variety of ways, such as social networks, biological systems, and the need to represent these as compactly as possible for faster execution of queries. We also address questions for eventual investigation.**

***Keywords*-Graph compression; graph mining; data mining; hypergraphs; hypergraph partition; hypergraph modeling; clustering;**

I. Introduction

Many real world problems and situations are effectively modeled as graphs and investigated using graph-theoretic tools as well as data mining algorithms. The resulting graphs are usually very large structures and posit difficulties with respect to processing or mining. On top of structural complexity, most graphs curated in various domains also consist of rich semantic information that could be effectively exploited by graph processing techniques. However, given a sufficiently large graph, many techniques and queries concede their otherwise empirical efficiency unless the graph is reasonably reduced via compression.

Graph compression remains an active area of research given the humongous number of edges and vertices (structured data) as well as auxiliary information and semantic features (unstructured data) that comprise a real-world graph. There are two general ramifications of graph compression. First, we may compress graphs to save space and time in order to execute graph algorithms more efficiently. This is known as *algorithmic graph compression* [1]. Second, we may exploit structural similarities among the vertices of a graph, which hinders at *structural graph compression*.

Either way, the central objective of graph compression is to exploit inherent redundancies in a graph. For instance, a traditional approach at compressing graphs is by attempting to diminish the number of edges. This can be attained by grouping similar vertices into a cluster, which is then connected via an edge to another cluster, a structure referred to as a 'cavemen community' [2]. Another corollary method is to order nodes based on some structural property, e.g. web page URLs are sorted lexicographically in [3], and establish compression-friendly cliques out of them.

Most, if not all, graph compression techniques are solely applicable to simple graphs, i.e. no other information is compressed besides what is provided by the structural layout of vertices and edges. However, real-world problems modeled as graphs possess auxiliary information (e.g. semantic features, vertex and edge labels, etc.) that should be captured and compressed along with the graph for at least two practical reasons. First, capturing such information provides context to eventual graph processing and management. Second, such information may provide clues for more effective node and edge layouts that could help to achieve higher compression.

The lack of capturing and incorporating auxiliary information on graphs is a weakness of most existing techniques because modeling real world problems merely as a collection of nodes and edges is, we believe, a naïve approach. For instance, cellular networks and many other biological processes cannot be efficiently modeled as simple graphs because such processes consist of complex functional relationships between any tuple of objects. An example of a complex biological process that cannot be efficiently modeled as a simple graph is the metabolic process (which requires a few related biological entities) or protein complexes (which encompass more than two proteins at a time) [4], [5]. Multigraphs, which are graphs that allow multiple links between two nodes, cannot be efficiently captured by a simple graph. We believe that many problems can and should be modeled as hypergraphs for at least the following reasons: (1) hypergraphs are essentially multisets that can encompass more information about complex interactions between entities than simple graphs; (2) hypergraphs can effectively capture auxiliary data – such as features and labels – related

to nodes in a single hyperedge, which results in a more compact representation of information. For example, protein networks could be modeled as hypergraphs in order to better capture and store information about protein complexes [5]. This approach raises two questions. First, how do we model such problems using a hypergraph? Second, what techniques can we employ to compress the resulting hypergraph?

In this paper, we present a method to model and compress such networks that consists of two phases. In the first phase, we establish several mappings to transform problems modeled as graphs into hypergraphs. Auxiliary information such as labels or semantic features on nodes and edges, as well as other data can be efficiently captured and modeled in the hypergraph. In the second phase, we partition the hypergraph into a number of parts in such a way that: (1) each part entails structurally-similar vertices (e.g. based on centrality scores of degree or betweenness as motivated by [2]); and (2) the number of cut (or external) hyperedges spanning two or more parts is minimized. The resulting partitioned hypergraph is then processed in order to yield more structural redundancy to increase compression. The contributions of this paper are listed below:

- **Paradigm shift in modeling.** We provide ways to model problems as directed or undirected hypergraphs as well as mappings to transform simple graphs with semantic features into hypergraphs. Moreover, we serendipitously observed that semantic graphs were significantly reduced in size when modeled as hypergraphs. This is due to the fact that a hyperedge may encompass several vertices, which reduces the number of rows in the incident matrix.
- **Hypergraph compression.** We compress hypergraphs by grouping vertices in each part based on centrality measures such as degree or betweenness. We also define a generalized theoretical cost function to evaluate the proposed compression scheme. This function may be applied to both directed and undirected hypergraphs.
- **Experimental results.** We perform a detailed experimental analysis on real social network datasets. Our evaluations show that modeling real-world problems as hypergaphs can lead to significantly compressed structures.

The rest of the paper is organized as follows. In Section II, we describe the proposed method in detail. We start with a motivating example, which leads to the general idea behind the hypergraph modeling paradigm. Next, we describe the hypergraph partitioning problem modified to fit our compression objectives. Results are discussed in Section III. Work consulted in the context of the proposed method is reviewed in Section IV. Future directions are posited in Section V.

II. Proposed Method

In this section, we propose a method to compress large graphs by first modeling them as hypergraphs, and then partitioning and compressing the hypergraphs. This method serves both algorithmic compression (e.g. more efficient, parallelizable graph operations) and structural compression (e.g. less space required to store and curate graph data).

A. Definitions

A hypergraph, $\mathcal{H} = \langle V, \mathcal{E} \rangle$, is a set of sets, or a system of subsets of a given set of objects [6]. Here, V denotes the set of vertices of the hypergraph, and $\mathcal{E} \subseteq 2^V$ denotes the set of hyperedges. A hyperedge $e \in \mathcal{E}$ is merely a set of vertices, i.e. $e \subseteq V$, such that $|e| \geq 2$. If direction of vertices in a hyperedge is important, the hyperedge is referred to as a *hyperarc* and the hypergraph is a called a *directed hypergraph*. An example of a directed hypergraph is illustrated in Figure 1. In the special case when $|e| = 2$, the hypergraph is reduced to a simple graph.

The incidence matrix of an undirected hypergraph $\mathcal{H} = \langle V, \mathcal{E} \rangle$ is the $|\mathcal{E}| \times |V|$ binary matrix $\mathbf{A} = (a_{ij})$, where $|\bullet|$ denotes the set cardinality operator and

$$a_{ij} = \begin{cases} 1 & : \quad v_j \in e_i \\ 0 & : \quad \text{otherwise} \end{cases}$$

This binary matrix can be extended to the following ternary matrix if $\mathcal{H}$ is a directed hypergraph:

$$a_{ij} = \begin{cases} -1 & : \quad v_j \in Tail(\mathcal{E}) \\ 1 & : \quad v_j \in Head(\mathcal{E}) \\ 0 & : \quad \text{otherwise} \end{cases}$$

Here, $Tail(\mathcal{E})$ denotes the set of "from" vertices, and $Head(\mathcal{E})$ denotes the set of "to" vertices. If there is only one "from" vertex (i.e. $|Tail(\mathcal{E})| = 1$), the hyperarc is referred to as a *backward hyperarc*, or B-arc. An F-arc, or *forward hyperarc*, is the case when $|Head(\mathcal{E})| = 1$ [7].

B. A Motivating Example

We motivate the proposed method by transforming the philosophers' graph (data extracted from `http://dbpedia.org/`) into an actionable hypergraph. The data model is straightforward. There are two entities – `era` and `philosopher` – with the following rules:

1) An era contains one or more philosophers.
2) A philosopher may belong to one or two eras.
3) A philosopher may influence zero or more philosophers.

A simple way to model these rules into a graph is by considering each instance of an era and philosopher as a vertex and adding an edge between each pair of vertices induced by the three rules above. For example, the pre-Socratic era encompasses, among others, philosophers Democritus and Parmenides. With a simple-graph approach, we would model these data as pairs: $\langle \text{pre} - \text{Socratic}, \text{Democritus} \rangle$ and $\langle \text{pre} - \text{Socratic}, \text{Parmenides} \rangle$.

However, we believe this simple graph approach is naïve (for many real-world problems as well) and does not reflect

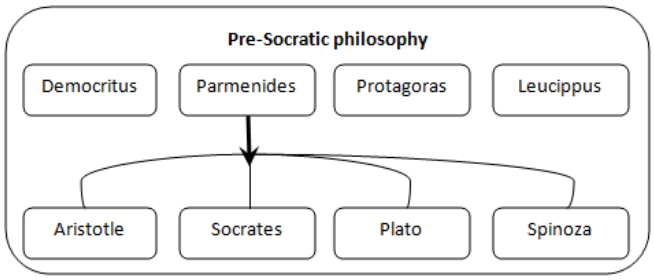

Figure 1. Modeling eras and philosophers as hyperedges and hyperarcs: the most compact approach.

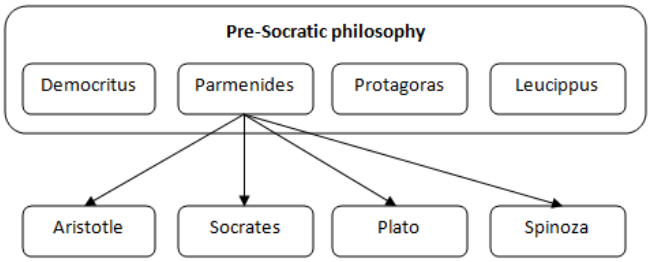

Figure 2. Modeling eras and philosophers as hyperedges and simple edges: the hybrid model approach.

the three rules above. A natural approach would be to model the relationships between vertices in terms of hyperedges. Based on the first and second rules, we may model a particular era and all the philosophers it encompasses as a single hyperedge. The third rule induces us to model the 'influences' relationship as a hyperarc (directed hyperedge). That is, a philosopher along with the set of philosophers they influence are modeled as a hyperarc. The resulting hypergraph is a hybrid model (directed and undirected hypergraph), as shown in the example in Figure 1. This model, however, yields a significantly reduced incidence matrix compared to the simple graph approach. An intermediary alternative would be to model the 'influences' relationship as a simple (undirected) edge between any two philosophers (see Figure 2). The corresponding adjacency or incidence matrices are displayed in figures 3, 4, and 5, where the number of nodes is 1546, and the number of edges/hyperedges varies according to the three mapping schemes discussed here.

C. Hypergraph Modeling

The first component of the proposed method is graph modeling via hypergraphs. Here, we provide the general scheme to transform a simple graph with semantic structure into a hypergraph. We focus on transforming undirected graphs noting that the mapping for directed graphs can be easily derived. An undirected graph $G = \langle U, E, F_U, F_E \rangle$ with semantic features (or, more generally, labels) F_U and F_E on nodes U and edges E, respectively, can be represented as a hypergraph $H = \langle V, \mathcal{E} \rangle$ as follows. Let

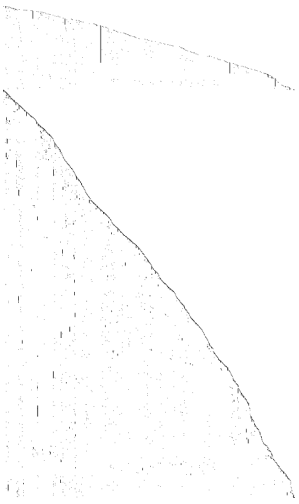

Figure 3. Adjacency matrix of the simple graph approach: 1546 nodes $\times$ 2602 edges.

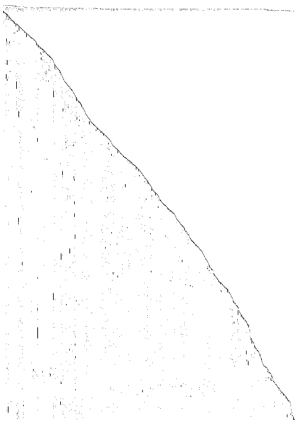

Figure 4. Incidence matrix of the hybrid model approach: 1546 nodes $\times$ 2191 hyperedges.

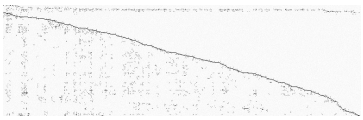

Figure 5. Incidence matrix of the most compact model approach: 1546 nodes $\times$ 475 hyperedges.

$\phi(u) \subseteq F_U$, $u \in U$, denote the set of features belonging to vertex u, and $\phi(\{u,t\}) \subseteq F_E$, $\{u,t\} \in E$ denote the set of features belonging to edge $\{u,t\}$. Then

- features in $\phi(u)$ can be viewed as vertices and, along with u, form a hyperedge, e. That is, $e = \phi(u) \cup \{u\}$.
- features in $\phi(\{u,t\})$ can be viewed as vertices and, along with vertices u and t, form a hyperedge e. That is, $e = \phi(\{u,t\}) \cup \{u\} \cup \{t\}$.

According to this mapping, the set of vertices V of the hypergraph is given by $V = \{U \cup F_U \cup F_E\}$, and $\mathcal{E} \in 2^V$. Note that if $F_U = F_E = \emptyset$, as in the case of the

philosophers' graph, a hyperedge would be a set composed of only vertices such that if two corresponding entries a_{ij} and a_{ik} in the incidence matrix are equal to 1, vertices v_j and v_k are connected by an edge in G.

As an example, consider two related users, v_1 and v_2, in any complex social network. Each of these users has a set of corresponding features such as date of birth, sex, geographical locale, movies they like, etc. The relationship between v_1 and v_2 may be more complex than simply 'friends', in the sense that it may contain more features that describe it. For instance, v_1 and v_2 may be in a relationship, are members of the same family, or schoolmates, etc. Modeled as a simple graph, we would have users v_1 and v_2 and an edge connecting the two. Modeled as a hypergraph, however, the semantic features would be captured in a lossless way and would be modeled as a hyperedge (or hyperarc, if direction matters). In general, if there are K types of semantic features in the simple graph, then there are $K+1$ different types of vertices in the corresponding hypergraph. A compression scheme would take $\log_2(K+1)$ bits to encode the type of vertex. For the three types of vertices mentioned above, V, F_U, and F_E, we would require $\log_2 3$ bits to encode each.

D. Hypergraph Partitioning

The second component of the proposed method is the partitioning of the hypergraph into compression-friendly clusters[1] of vertices subject to certain constraints as discussed below. Formally, a k-way hypergraph partitioning consists of k vertex sets, $P^k = \{V_1, V_2, \ldots, V_k\}$, $V_i \in V$ also known as clusters or parts. A weighting function $w : V \rightarrow R$ may be used for each vertex and may be aggregated for each cluster of vertices: $w(V_i) = \sum_{v \in V_i} w(v)$. The set of hyperedges cut by a partitioning solution, P^k, is given by:

$$E(P^k) = \{e \in \mathcal{E} | \exists u, v \in e, v \in V_i, u \in V_j, i \neq j\}.$$

The overall objective of hypergraph partitioning is to minimize $|E(P^k)|$.

The primary purpose of the proposed method is to transform the hypergraph into a compression-friendly structure. This can be achieved by permuting the vertices and hyperedges of the hypergraph in a way that increases redundancy – and thus the degree of compressibility – within non-overlapping regions of the incidence matrix. The procedure we have devised to attain a compression-friendly incidence matrix is as follows. First, compute a k-way vertex partitioning such that vertices with high centrality scores (e.g. degree or betweenness) form their own communities within the first few parts, thus potentially yielding to a small cardinality of the set of cut hyperedges. More formally, let $w(V_i) = \sum_{v \in V_i} w(v)$ denote an aggregate centrality score for part V_i, $i = 1, 2, \ldots, k$. The constraint we impose on the solution is yielding k communities with a decreasing order of aggregate centrality:

$$w(V_1) \geq w(V_2) \geq \ldots \geq w(V_k).$$

If we were to view such a solution in terms of the incidence matrix, each part V_i would correspond to a collection of columns (which represent vertices) of the matrix. Then, part V_1 would contain vertices with higher centrality scores than part V_2, i.e. the first region in the matrix would contain more 1-valued entries than the second region, and so forth until the k-th region, which would contain mostly 0-valued entries. In principle, this ordering of vertices would yield a more compression-friendly matrix.

In addition to ordering vertices per the aforementioned weight constraints, the objective is to minimize the number of cut hyperedges. That is,

$$Min.F(P^k) = \sum_i \left(\lambda(e \in E(V_i)) - 1\right) w(e \in E(V_i)),$$

where $E(V_i) \in E(P^k)$ denotes the set of hyperedges cut by part V_i, λ denotes the number of parts hyperedge e spans less 1 (i.e. if the hyperedge doesn't span any parts, $\lambda - 1 = 0$), and $w(e \in E(V_i))$ is the weight of hyperedge e. The weight of a hyperedge may be defined depending on the specific situation the hypergraph models. In the case of social networks, for instance, the weight of a hyperedge could be set to the sum of a centrality score of each vertex the hyperedge contains.

Next, once a solution P^k is yielded, we order the hyperedges such that uncut hyperedges are adjacent to each other. This increases redundancy in the resulting incidence matrix thus leading to potentially higher compression, especially when block-wise encoding (discussed in the next sub-section) is used. Technically, the 1-valued entries of the incidence matrix become adjacent to each-other (and so do the 0-valued entries).

There are several reasons we take the aforementioned partitioning approach. First, real-world graph data reveal a power-law distribution of vertices [2]. That is, there are very few nodes with very high centrality scores in a given graph. This is why we impose a constraint such as $w(V_1) \geq w(V_2) \geq \ldots \geq w(V_k)$. Second, this type of partitioning is highly parallelizable and scalable [9]. Third, clustering vertices with high centrality scores may reduce the querying time for selected query types, such as path finding. Thus, vertices in part V_1 (ordered in a decreasing way per part weight) have a much lower entropy than vertices in part V_k.

It is worthwhile mentioning that classic partitioning of simple graphs has been criticized due to its hardness and "no good cuts", primarily due to the fact that such partitioning doesn't take into account the power-law vertex distribution of real-world graphs [2]. However, hypergraph partitioning

[1]Note: Technically, clustering is different from partitioning in that the former is about merging vertices into a larger group of vertices, whereas the latter is about dividing vertices and grouping them under a set based on similarity or other criteria [8].

in the sense of the proposed method is not handicapped by this empirical observation. Instead, it considers this observation as an essential constraint in the problem formulation, as stated above. Finally, empirical research on real-world graphs such as social networks or ground-truth communities reveals that vertices tend to exhibit links to densely organized communities [10].

Next, we look at a definition of compression ratio and a compression measure we use in our experiments. From the observations above, we may surmise that compressing the final incidence matrix in a block-wise fashion is more efficient than compressing the matrix all at once because the matrix as a whole would introduce more sparsity and, therefore, less redundancy than smaller non-overlapping blocks. The cost measure we use here is a variation of the cost function used in [2] and assumes an information-theoretic lower bound. We generalize this cost function to apply to ternary (and higher-order) matrices as well. Let $\mathbf{A}$ denote the incidence matrix and $a \times b$ the dimensions of the block, $1 < a < |\mathcal{E}|$, $1 < b < |V|$. Then, we have

$$cost_{\mathrm{b}}(\mathbf{A}, a, b) = |T| \cdot \log_2 \left(\frac{|\mathcal{E}||V|}{ab} \right) + \sum_{\tau \in T} ab \cdot H_{\mathrm{b}} \left(\frac{z(\tau)}{ab} \right),$$

where, T is the set of non-empty blocks of size $a \times b$, $z(\tau)$ is the number of zeros in block τ, and $H_{\mathrm{b}}(x) = -x \log_2(x) - (1-x) \log_2(1-x)$ is the binary entropy function. The term $\frac{|\mathcal{E}||V|}{ab}$ counts the total number of $a \times b$ blocks. Thus, its logarithm gives the number of bits required to encode each block in the incidence matrix. The second term provides the information-theoretic lower bound to encode the bits in given block τ. That is, we require on average $I_0 = -log_2 \frac{z(\tau)}{ab}$ bits to encode a 0 and $I_1 = -log_2 \left(1 - \frac{z(\tau)}{ab}\right)$ bits to encode a 1. Thus, the cost to encode block τ is $z(\tau)I_0 + (ab - z(\tau))I_1 = ab\frac{z(\tau)}{ab} I_0 + ab\frac{ab-z(\tau)}{ab} I_1 = abH_{\mathrm{b}}(\frac{z(\tau)}{ab})$. The generalized cost function is given by:

$$cost(\mathbf{A}, a, b) = |T| \cdot \log_2 \left(\frac{|\mathcal{E}||V|}{ab} \right) + \sum_{\tau \in T} ab \cdot H\left(\mathbf{p}\right),$$

where $\mathbf{p}$ denotes the probability distribution of $(-1, 0, 1)$ and $H(\mathbf{p}) = -\sum_{x \in \mathbf{p}} x \log_2(x)$ is the information entropy function. When evaluating both cost functions, we will use square blocks (i.e., $a = b$) for processing simplicity.

E. Overview of the Algorithm

The steps discussed above can be summarized in Algorithm 1. Hypergraph modeling merges vertex and edge features with the vertex set U to yield V. The worst-case complexity of this step is when $|F_U| \geq |V|$, thus $Convert(G) \in O(\max\{|F_U|, |V|\}^2)$. In practice, however, this is not the case. Also, various data structures may provide better empirical complexity – we leave the actual representation for future research. The complexity of hMeTis is to be empirically evaluated due to the way that framework is designed, i.e. computational efficiency depends on which partitioning and coarsening scheme is chosen and which parameters are selected for each scheme. Permuting the hyperedges is achieved by going through all $|\mathcal{E}|$ hyperedges and determining if each hyperedge is internal or external. A list for the reordering of hyperedges is preserved and, at every step, internal hyperedges are consecutively linked. The computational complexity of this step is bound by $O\left(|\mathcal{E}||P^k|\right)$. This step is observed to be empirically fast, at least for the real-world hypergraphs we experimented with. Overall, efficiency is presently bound by the modeling step. Further adjustments in the selected data structures may be needed for both time and space complexity. The latter complexity will be determined thereafter.

Algorithm 1 Hypergraph Compression

Input: Graph $G = \langle U, E, F_U, F_E \rangle$
Output: Incidence matrix $\mathbf{A}$ and $cost(\mathbf{A}, a, b)$
1. Hypergraph Modeling (Section II-C):
function CONVERT(G)
  $V = U \cup F_U \cup F_E$
  $\mathcal{E} = \emptyset, e = \emptyset$
  for $v \in V$ **do**
    $e = \{v\}$
    for $f \in F_U$ **do**
      if $f \in \phi(v)$ **then**
        $e = e \cup \{f\}$
      end if
    end for
    $\mathcal{E} = \mathcal{E} \cup e, e = \emptyset$
  end for
  return $\mathcal{H} = \langle V, \mathcal{E} \rangle$
end function
2. Hypergraph Partitioning (Section II-D): Partition $\mathcal{H}$ using hMeTis [11] given the constraint and objective function
3. Permute hyperedges in $\mathbf{A}$ such that all internal hyperedges are adjacent to each-other. The external hyperedges are permuted in a increasing order based on λ_e. That is, the hyperedge with that spans the least number of parts follows the internal hyperedge last permuted, and so forth.

III. EXPERIMENTAL RESULTS

In this section, we provide empirical results guided by the following research questions:

- How well can we compress hypergraphs per the procedure in Section II-E? What are some results pertaining to the empirical efficiency of the proposed method using various parameters, such as block size, and partition variables?
- How does the proposed method per the algorithm in Section II-E perform relative to methods such as random and natural ordering?

To study these questions, we leverage insights from experiments on the following available data set, noting that obtaining data pertaining to graphs with semantic features in the format required for the proposed algorithm has been a difficult task, with few data, such as Twitter, being removed due to privacy issues. We then provide a discussion for each research question and ask more questions for future experimental results and improvements of the proposed method. The dataset used here along with the source code developed to evaluate the compression cost as in Algorithm 1 are located at `http://goo.gl/hmXodC`.

Table I shows some properties of the selected data sets for the experiments.[2] The vertex set includes all vertices and features, where applicable. The Philosophers data set has been described above. The Orkut Communities data set includes the top 5000 communities, i.e. groups of vertices with highest quality [10]. Each community is viewed here as a hyperedge. The General Relativity authorship collaborations represent undirected relationships between any two authors. In this case, each pair is viewed as a hyperedge in the simple case. Note that this representation can be extended to form hyperedges based on an ego-vertex, i.e. an author with many collaboration links. We leave this for further investigation.The AS Oregon-1 shows the AS routing views from Oregon between March 31 and May 26, 2001. Here, each hyperedge consists of a pair of routing points. Next, the Amazon Co-Purchases shows the network of products following Amazons *Customers Who Bought This Item Also Bought...* functionality. Finally, the CAIDA AS Relationship dataset is a directed hypergraph with four additional types of vertices (features on edges in the simple graph): *is a customer of*, *is a provider of*, *is peers with*, and *is siblings with*. Compression results for these graphs are shown in Table II, which contains the total number of bits required to theoretically encode the given hypergraph, and Table III, which contains bits per hyperedge required to encode a hyperedge of the given graph using the theoretical cost function described above for various block sizes and parts from the partitioning step. Each hypergraph compression result is compared versus a random ordering of vertices as well as permuted vs. non-permuted hyperedges (step 3 of the proposed algorithm).

As can be seen from Table III, the proposed algorithm compresses hypergraphs with vertices and hyperedges laid out as described hitherto with a relatively high theoretical lossless compression ratio. We applied the proposed method on both directed and undirected hypergraphs. Direction in AS-CAIDA, for instance, is defined by the four relationship features, which were considered vertices of the hypergraph and included in the evaluation of the compression cost. On all datasets, we used random ordering of vertices, which requires the largest number of bits per hyperedge, as well as non-permuted and permuted hyperedges. Note that for the latter two there is little or no significant improvement in compression ratios. This is probably due to the fact that for the hypergraphs with small hyperedge cardinality (2 or 3, in the selected dataset, except for Orkut), external hyperedges are in minority. Therefore, permuting these hyperedges would make no perceivable difference in terms of compression. However, permuting hyperedges made a slight difference in the case of Orkut, where hyperedges comprised a large number of vertices (the first community consisted of over 6000 members, for example). As such, external hyperedges vary in number, and reordering these could lead to better compression for similar hypergraphs. This is also the reason why results for Orkut stand out contrasted to other hypergraphs. In the Orkut incidence matrix, there is less redundancy, given the very small number of hyperedges (5000) and the high number of non-zero bits dispersed throughout the matrix. If we were to consider the one-to-one Orkut relationships (i.e. having hyperedge cardinalities of 2 or 3), the compression ratio would probably decrease to a two-digit value, as in the case of other hypergraphs. Furthermore, the effect of a large number of vertices can be seen in the case of a random-ordered Amazon incidence matrix, where the ratio is 44,402 bits (5.42 KB) per hyperedge. Laying out vertices randomly in such an incidence matrix will increase the number of external hyperedges upon partition. As such, redundancy in that matrix will decrease, leading to lower compression. Yet, when vertices are ordered naturally and hyperedges permuted, the compression ratio is decreased to about 17 bits per hyperedge. This implies that

Data set	$\lvert V\rvert$	$\lvert\mathcal{E}\rvert$	**Directed?**
Philosophers	1546	2602	Yes
Orkut Communities	731514	5000	No
Relativity Co-Authors	5242	28980	No
AS Oregon-1	11174	23410	No
Amazon Co-Purchases	334863	925872	No
CAIDA AS Relationships Dataset	26479	106762	Yes

Table I
TEST DATA FOR OUR EXPERIMENTS

Data set	**Random**	**Non-permuted**	**Permuted**
Philosophers	85.7708	24.9004	20.4405
Orkut	3857.2918	973.6721	973.5965
Co-Authors	15.6038	12.2423	12.2423
AS Oregon-1	26.3607	17.8530	17.8530
Amazon	44402.1379	17.4805	17.4805
AS-CAIDA	17.8889	12.53069	12.53069

Table III
BITS PER HYPEREDGE REQUIRED TO THEORETICALLY ENCODE EACH HYPEREDGE: RANDOM ORDERING, NON-PERMUTED HYPEREDGES, PERMUTED HYPEREDGES. THIS METRIC PUTS ALL HYPERGRAPHS IN A COMPARISON-INDUCING PERSPECTIVE. THE RATIOS IN THE LAST COLUMN ARE THE MOST EFFICIENT COMPRESSION RATIOS IN BITS PER HYPEREDGE.

[2]All data sets but the Philosophers' graph were taken from `http://snap.stanford.edu/data/`.

Data set	Random	Non-permuted	Permuted	Parts	$a \times b$
Philosophers	40741.13	11827.69	9709.24	30	16×16
Orkut	19286459.21	4868360.71	4867982.32	150	16×16
Co-Authors	452197.68	354782.47	354782.47	30	16×16
AS Oregon-1	620848.03	420473.68	420473.68	40	256×256
Amazon	41110696218.68	16184615.36	16184615.36	300	256×256
AS-CAIDA	1909837.02	1337795.74	1337795.74	40	16×16

Table II
TOTAL NUMBER OF BITS REQUIRED TO THEORETICALLY ENCODE EACH HYPERGRAPH: RANDOM ORDERING, NON-PERMUTED HYPEREDGES, PERMUTED HYPEREDGES.

vertex ordering has its own effects on compression. In our experiments, we did not focus on permuting vertices and we leave this for future research.

In terms of empirical efficiency, the partitioning step for Amazon, for example, took 167.65 seconds to complete. Hypergraph partitioning with hMeTis – the tool used here – is reportedly the fastest algorithm. However, there are parallel partitioning methods, such as the disk-based hypergraph partitioning proposed in [9], which significantly improve performance. We leave parallelization of both partition and compression for future research. Furthermore, it must be noted that parameters, such as block dimensions and number of parts, were empirically chosen based on the number of vertices and hyperedges for each graph. More results are needed to determine the effects of various parameters on lossless compression.

To answer the question on how well the proposed method can perform relative to other schemes, we need to compare existing hypergraph-based compression results with the proposed method. To our knowledge, no such results have been published to date. However, if our results are juxtaposed with those in [2], we may conjecture that the proposed method yields very high compression ratios.

IV. RELATED WORK

Most related research has focused on compressing structured graphs for both algorithmic and structural compression. In other words, semantic features and other labels – the unstructured data which are conspicuous in social networks – are excluded from any compression. Vertices are laid out or clustered in a particular way that is more compression-friendly, although optimal orderings have been shown to be NP-hard [12]. A seminal work on structured compression is the Webgraph Framework [3], where vertices represent URLs of web pages. Sorting vertices by URL – also referred to as lexicographic ordering – was observed to group similar web pages together, thus increasing data redundancy for better compression of the graph structure of the web. In graphs where edges represent relationships, such as the "influences" relationship in the philosophers' graph (Section II-B), ordering nodes lexicographically may not be as efficient as the shingle ordering proposed in [12]. Shingle ordering is an instance of natural ordering of vertices by obtaining a vertex neighborhood schema and ordering all other vertices based on this schema. For instance, if a neighborhood schema is built for "influences" and "is student of" relationships in the philosophers' graph, all vertices will be ordered conforming to a similarity coefficient per the chosen schema. Grammar-based ordering has also been suggested as a fast compression method [13].

Besides rearranging vertices, clustering them in closely-related communities has been extensively studied. METIS algorithms for k-way multilevel graph partitioning have been extensively studied [11], [14] with recent advances in parallelizing the partitioning steps [15]. Other clustering methods include cross-association [16], spectral clustering [17], or folksonomies [18], which detect structural patterns in order to reduce the number of edges when relating communities of vertices to one-another.

Other types of graph compression techniques are contextualized by the purpose of compression, such as query-preserving or neighbor-query-friendly methods [19], [20], geographic clustering for network graphs [1], and methods based on centrality scores, such as SlashBurn [2].

Hypergraph modeling techniques have been developed to study problems arising primarily in VLSI and circuitry [21], as well as protein complexes [5]. Hypergraph partitioning has been an active area of research [11] with a variety of modern techniques [8]. However, we identified a dearth of literature in (i) integrating semantically-rich data (features, labels, relationships, etc.) with the structural layout of the vertices, and (ii) modeling such data along with the corresponding graphs as hypergraphs for the purpose of devising efficient compression schemes. The method proposed in this paper addresses these two points while naturally committing to a trade-off between empirical efficiency (a characteristic of structural graph compression) and quality (a characteristic of hypergraph modeling and partitioning techniques).

V. CONCLUSIONS AND FUTURE DIRECTIONS

We have proposed a method to model real-world graphs with simple-to-advanced semantic structures via specially-constructed hypergraphs. First, we model such graphs as hypergraphs by considering node and edge labels as vertices that can be combined in hyperedges (or hyperarcs in the case of directed hypergraphs). Second, we partition the resulting

hypergraph in ways that increase structural redundancy – which can, in turn, be exploited by a compression scheme. To gain insights on the theoretically achievable compression ratio, we used block-wise encoding via binary entropy.

The experimental results reveal that modeling even simple graphs as hypergraphs can yield sufficiently good compression ratios. The strength of the proposed hypergraph modeling should also be determined in the context of real complex network (such as those in Biology). This is one of the next steps of this research, including studying the properties of the resulting hypergraphs in order to determine what type of queries can be efficiently executed and what type of query-friendly compression methods we could devise. Particular properties we are interested in include: vertex degree distribution, which also determines the structural heterogeneity of the hypergraph in an information-theoretic sense [22], [23]; path length, which determines whether a hypergraph has a small-world structure; and the maximum k-core, which determines which nodes are the most crucial or critical in the hypergraph (c.f. [4]).

REFERENCES

[1] A. Gilbert and K. Levchenko, "Compressing network graphs," in *The Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD'04*, 2004.

[2] U. Kang and C. Faloutsos, "Beyond 'Caveman Communities': Hubs and spokes for graph compression and mining," in *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, 2011, pp. 300–309.

[3] P. Boldi and S. Vigna, "The Webgraph Framework I: Compression Techniques," in *Proceedings of the 13th international conference on World Wide Web*, 2004, pp. 595–602.

[4] S. Klamt, U.-U. Haus, and F. Theis, "Hypergraphs and cellular networks." *PLoS Computational Biology*, vol. 5, no. 5, p. e1000385, May 2009.

[5] E. Ramadan, A. Tarafdar, and A. Pothen, "A hypergraph model for the yeast protein complex network," in *18th International Parallel and Distributed Processing Symposium Proceedings.* Ieee, 2004, pp. 189–196.

[6] C. Berge, *Graphs and Hypergraphs*. Amsterdam: North Holland, 1973.

[7] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen, "Directed hypergraphs and applications," *Discrete applied mathematics*, vol. 42, no. 2-3, pp. 177–201, Apr. 1993.

[8] D. Papa and I. Markov, "Hypergraph partitioning and clustering," *Approximation algorithms and metaheuristics.*, pp. 1–38, 2006.

[9] A. Trifunovic and W. J. Knottenbelt, "Towards a Parallel Disk-Based Algorithm for Multilevel k-way Hypergraph Partitioning," in *Proceedings of the 18th International Conference on Parallel and Distributed Processing*.

[10] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics - MDS '12*. New York, New York, USA: ACM Press, 2012, pp. 1–8.

[11] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proceedings of the 36th Design Automation Conference*. Ieee, pp. 343–348.

[12] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *The 15th acm sigkdd international conference on knowledge discovery and data mining KDD '09*, 2009, pp. 219–227.

[13] F. Claude and G. Navarro, "Fast and compact Web graph representations," *ACM Transactions on the Web (TWEB)*, vol. 4, no. 4, pp. 16:1–16:31, 2010.

[14] K. Schloegel, G. Karypis, and V. Kumar, "Parallel Multilevel Algorithms for Multi-constraint Graph Partitioning," 2000, pp. 296–310.

[15] D. LaSalle and G. Karypis, "Multi-Threaded Graph Partitioning," *27th IEEE International Parallel and Distributed Processing Symposium*, pp. 225–236, May 2013.

[16] D. Chakrabarti, S. Papadimitriou, D. Modha, and C. Faloutsos, "Fully automatic cross-associations," in *Proceedings of the Knowledge and Data Mining Conference KDD*, no. 22, 2004, pp. 79–88.

[17] U. Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Aug. 2007.

[18] C. Bothorel and M. Bouklit, "An algorithm for detecting communities in folksonomy hypergraphs," in *Proceedings of the 2011 Innovative Internet Community Services, IICS*, 2011, pp. 159–168.

[19] W. Fan, J. Li, X. Wang, and Y. Wu, "Query preserving graph compression," in *Proceedings of the 2012 international conference on Management of Data - SIGMOD '12*. Scottsdale, Arizona, USA: ACM Press, p. 157.

[20] H. Maserrat and J. Pei, "Neighbor query friendly compression of social networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10*. New York, New York, USA: ACM Press, 2010, pp. 533–541.

[21] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integration, the VLSI Journal*, vol. 19, no. 1-2, pp. 1–81, Aug. 1995.

[22] T. Eliassi-Rad, I. Fodor, and B. Gallagher, "A Collection of Features for Semantic Graphs," 2007.

[23] R. V. Sole and S. Valverde, "Information theory of complex networks: On evolution and architectural constraints," in *Complex Networks*, E. Ben-Naim, H. Frauenfelder, and Z. Toroczkai, Eds., 2004, pp. 1–15.