Beyond Graphs: Toward Scalable Hypergraph Analysis Systems*

Benjamin Heintz
University of Minnesota
Minneapolis, MN
heintz@cs.umn.edu

Abhishek Chandra
University of Minnesota
Minneapolis, MN
chandra@cs.umn.edu

ABSTRACT

Graph theory has provided a powerful modeling foundation for problems in many domains, but we argue that *group* interactions are better modeled by *hypergraphs*. As we work toward scalable systems for such hypergraph analysis, several major challenges and opportunities arise; here we highlight a sample of those challenges. We consider the need for efficient representations of hypergraphs, and show that in some cases it is possible to exploit the specific structure of a hypergraph to reduce storage overhead. We also explore several challenges in distributing computation on hypergraphs, including the need for more general partitioning approaches. Finally, we discuss several other problems that arise as we move from graphs to hypergraphs, including designing programming models, using hypergraphs to model real-world groups, and the need for a better understanding of the structural characteristics of hypergraphs.

1. INTRODUCTION

Recent years have seen massive amounts of data being generated at unprecedented scales, through the rapid growth in large social networking platforms such as Facebook and Twitter, extensive recording of consumer data for both online and in-store purchases, and discovery of dense genetic networks of various organisms. As such systems consist of a large number of entities interacting with each other, there is an increasing interest in studying group dynamics in order to better understand many social, economic, and biological phenomena. For instance, the *group* is a fundamental building block of many social interactions [9], from attending events together, to collaborating in teams, to sharing information and views on social networks. In fact, many aspects of an individual's behavior depend upon the behavior of the social groups to which that individual belongs. Similarly, in the biology domain [2], many physiological phenomena are driven by group interactions between multiple genes rather than by individual genes, and such group interactions are important for understanding diseases and finding cures.

Graph theory has been used to study and model several phenomena in a variety of domains, allowing data analysts to find and understand relations between various entities, for example friendships between users, or interactions among genes. The standard graph analysis approach has been to consider the interaction between a pair of actors as the basic unit of interaction, modeled as an edge between two vertices, leading to a dyadic graph model.[1] While suitable for many purposes, such a graph model fails to capture group-level interactions between actors, which are considered to be fundamentally different. For example, a group of academics A, B, and C co-authoring a paper is not the same as three separate papers co-authored by A and B, by B and C, and by A and C, respectively. Specifically, the latter model fails to capture the three-way collaborative nature of the relation between the authors. This group of co-authors is better modeled as a hyperedge $\{A, B, C\}$ rather than as the set of dyadic edges $\{A, B\}$, $\{B, C\}$, and $\{A, C\}$.

This observation suggests that the right mathematical object for modeling group interactions is not a graph, but rather a *hypergraph*. Formally, a hypergraph (see Figure 1) can be defined as a tuple $H = (V, E)$, where V is the set of entities, called *vertices*, in the network, and E is the set of subsets of V, called *hyperedges*, representing relations between one or more entities [1]. Research has shown that several social, biological, ecological, and technological systems can be better modeled using hypergraphs than using dyadic proxies [4].

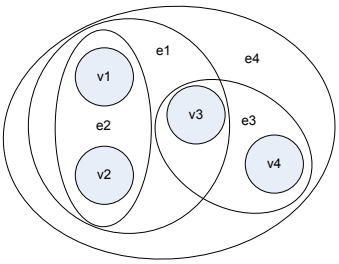


Figure 1: A hypergraph with four vertices (filled circles) and four hyperedges (enclosing ovals).

A key challenge in supporting hypergraph analysis is the large volume of underlying data in many domains. For example, large social networks such as Facebook and Twitter have nearly a billion users, and on the order of millions or billions of group interactions (e.g., likes, comments, and tweets) every day. Thus, hypergraph analysis algorithms need to be scalable both in terms of their memory/storage usage as well as computation by utilizing distributed resources. While several large-scale graph processing systems and frameworks [10, 11] have been proposed in the recent past, and while hypergraphs themselves have been studied for decades, we believe that *scalable distributed systems for hypergraph analysis* present novel challenges and opportunities.

In this paper, we discuss three sets of issues in developing scalable hypergraph analysis systems: efficient data representation (Section 2), distributed computation (Section 3), and hypergraph modeling and characterization (Section 4).

To illustrate these issues, we use two publicly available datasets: the DBLP authorship database,[2] and Subversion logs from the Apache

*This research is funded in part through an IBM Faculty Award.

[1]We will refer to a dyadic graph as a *graph* unless otherwise noted.

[2]http://dblp.uni-trier.de/xml

Software Foundation's subversion repository.[3] In the DBLP hypergraph, each vertex models an author, and each hyperedge models a distinct collaborating set of authors. The Apache hypergraph models collaboration on open-source software projects in the Apache Software Foundation's Subversion repository. In this hypergraph, each vertex represents a committer, and each hyperedge represents a unique set of committers that have collaborated on one or more files. For example, if committers A, B, and C have all modified file F, then this group is modeled in the hyperedge {A, B, C}. (If they have collaborated on multiple files, then the count or full list of these files can be recorded as a hyperedge attribute.)

Table 1: Datasets.

Dataset	Vertices	Hyperedges
DBLP	1,199,273 authors	1,383,304 author sets
Apache SVN	3,360 committers	80,922 committer sets

Table 1 provides further details, and Figure 2 shows the distribution of vertex degrees and hyperedge sizes for these hypergraphs. We note that while these datasets are relatively small, they provide several insights about the challenging nature of hypergraph analysis. Moreover, we can study the hypergraph properties of these datasets because they are amenable to analysis on a small scale, further motivating the need for scalable hypergraph analysis systems to handle much larger datasets.

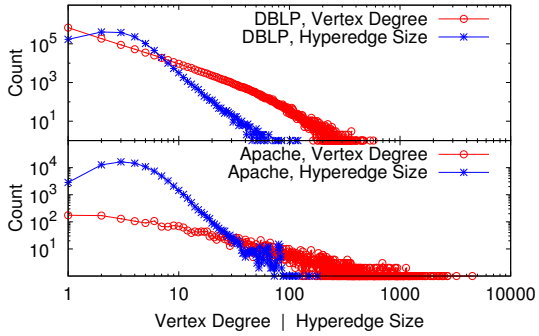
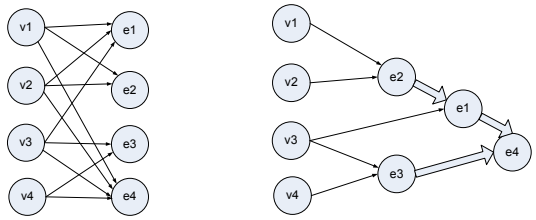


Figure 2: Distributions of vertex degrees and hyperedge sizes for the DBLP hypergraph (top), and the Apache hypergraph (bottom).

2. EFFICIENT DATA REPRESENTATION

The first issue we discuss is that of data representation; i.e., how hypergraphs should be represented at a system level to enable efficient storage and processing. Even at the fundamental level of data representation, hypergraphs present richer challenges and opportunities compared to their graph counterparts. We illustrate these differences through one example property of hypergraphs: the unrestricted vertex membership of hyperedges. In particular, a key distinction between a graph and a hypergraph is that while an edge in a graph can be incident on exactly two vertices, each hyperedge in a hypergraph is an arbitrary subset of the vertex set. Thus, many hyperedges may be subsets of other hyperedges. As an example, in Figure 1, hyperedges e_1 and e_3 are subsets of hyperedge e_4. We next discuss how this property could impact the data representation efficiency of a hypergraph.

One way to represent a hypergraph is to use an underlying bipartite graph, which we will describe in terms of its *nodes* and *links* to avoid confusion with the hypergraph counterparts. In a bipartite representation, nodes on one side of the bipartition represent vertices in the hypergraph, while nodes on the other side represent hyperedges. Links in the bipartite graph represent the membership of vertices in hyperedges. Figure 3a shows the bipartite graph representation of the example hypergraph from Figure 1.



(a) Bipartite graph (b) Hierarchy DAG (HDAG)

Figure 3: Hypergraph representations.

As an alternative, we can explicitly encode the subset/superset relationships in the underlying representation. In general, we add a link from e_i to e_j if $e_i \subset e_j$ and there does not exist a hyperedge e_k such that $e_i \subset e_k$ and $e_k \subset e_j$. In the example from Figure 3a, we add a link from e_1 to e_4 to reflect that $e_1 \subset e_4$. We similarly add links from e_3 to e_4 and from e_2 to e_1 to reflect additional subset relationships. We do not, however, add a link from e_2 to e_4 even though $e_2 \subset e_4$; there is already a path of links that establishes this connection. Figure 3b shows how we can represent the hypergraph from Figure 1 using this alternative representation, which we refer to as the *Hierarchy DAG*, or HDAG.[4] The HDAG can be constructed efficiently from the bipartite representation: the subset/superset relationships can be established in $O(1)$ rounds of messages between hyperedges and vertices, and redundant links (e.g., from e_2 to e_4) can then be removed by propagating messages in a parallel breadth-first manner from the vertices.

To explore further, we consider the DBLP hypergraph and observe the distribution of the number of strict subsets contained within each hyperedge. We notice a high degree of skew, with most hyperedges containing fewer than a handful of strict subset hyperedges, while some others contain 50 or more subsets.

Given this observation, we would expect that by removing direct links from large hyperedges to each of their adjacent vertices, and instead using the superset-to-subset links to achieve this connectivity indirectly, we could reduce storage overhead. Indeed this is what we find: In the bipartite representation, there are a total of about 4.23 million hyperedge-to-vertex links (and the same again from vertices to hyperedges, as we model an undirected hypergraph). In the HDAG representation, more than 47% of these links are removed, and 1.68 million new hyperedge-to-hyperedge links are added to maintain the original connectivity. Overall, the number of links in the underlying representation decreases by 7.9%, showing that *we can exploit the structure of a hypergraph to achieve a more efficient representation.*

These savings are critical not only for the raw storage of the hypergraph (e.g., on disk), but also in terms of the amount of memory required to operate on the hypergraph, especially for memory-resident algorithms (even if the memory is distributed across machines). Moreover, such space savings may also improve the communication efficiency during analysis. As an example, the links of the underlying graph representation might be used for passing messages among vertices and hyperedges across different phases of an iterative computation (similar to that in graph-processing systems

[3] http://svn.apache.org/repos/asf

[4] We also add symmetric superset-to-subset links to build a separate reversed DAG to allow traversal in either direction.

such as Pregel or GraphLab). Reducing the number of links reduces the number of messages and might also facilitate better aggregation of such messages where needed. Further, reduced overall memory requirements might reduce the number of machines needed for distributed computation on the hypergraph, resulting in fewer messages crossing physical machine boundaries.

3. DISTRIBUTED COMPUTATION

The large size of interesting datasets requires that we distribute computation across a cluster of machines, and the need for efficient distributed computation leads to interesting scheduling and load balancing issues. Distributed graph analysis systems such as Pregel [11] and GraphLab [10] provide a graph-specific execution model, where computation is carried out in the form of a vertex program that runs at each vertex, receiving messages from and sending messages to its neighbors. Gonzalez et al. [6] generalize these models to a GAS (Gather, Apply, Scatter) model that can describe either Bulk Synchronous Parallel [12] execution, or asynchronous execution.

While these models have served well in the graph context, hypergraphs present different challenges. Most notably, hyperedges are not merely passive entities in the way that edges are often treated in the graph setting. Instead, they are first-class citizens in hypergraph analysis, and systems must support hyperedges that maintain attribute state and perform computation just as vertices do. Therefore, supporting hypergraph analysis—including hyperedge-based computations—will require either extending existing graph execution models, or developing new execution models that are a better fit for these new requirements.

This added complexity presents interesting challenges in terms of partitioning the hypergraph for the purpose of distributing computation across a cluster. Just as with graphs, hypergraphs will exhibit highly skewed vertex degree distributions. But unlike dyadic graphs, where all edges have the same cardinality, hypergraphs can also exhibit high skew in terms of hyperedge cardinality (size). For example, Figure 2 shows the skewed distributions of vertex degree and hyperedge size in both the DBLP and Apache hypergraphs. This pronounced skew can lead to some hyperedges or vertices requiring much more computation than others, making partitioning difficult due to imbalanced load, just as in the case of dyadic graphs [6].

Partitioning a dyadic graph is typically a matter of partitioning the vertex set [11] and in turn "cutting" edges by replicating them across partition boundaries, or by partitioning the edge set [6], and cutting vertices. The goal of partitioning is to minimize the number of edge (respectively, vertex) cuts subject to some balance constraint. The arbitrary size of hyperedges makes the partitioning problem more challenging: in particular, while a vertex-based partitioning heuristic has been shown to be inefficient for graphs that exhibit highly skewed vertex degrees [6], a hyperedge-based partitioning heuristic may suffer from the same problem in the presence of highly skewed *hyperedge sizes*. Thus, we may require a combination of vertex- and hyperedge-based partitioning heuristics in order to enable efficient partitioning in the presence of *both* vertex degree skew and hyperedge size skew. At the same time, these heuristics must facilitate efficient *distributed* partitioning of a hypergraph.

Another question to consider is the distinction between vertex and hyperedge computations. For instance, in some applications, vertex computations may be much more heavy-weight (i.e., carrying out more operations or requiring more state), while hyperedge computations may be relatively light (e.g., they may simply forward state between vertices). Such computation heterogeneity might impact scheduling and partitioning decisions; it is no longer the case that we can simply use a one-size-fits-all partitioning heuristic (e.g., minimizing vertex cuts). Instead, an effective partitioning heuristic needs to minimize a more holistic cost function based on the relative cost of vertex and hyperedge computations as well as the communication between them. This might require cutting vertices in some cases, cutting hyperedges in others, or cutting both in other cases.

Hypergraph partitioning was a popular topic in previous decades, particularly in the context of VLSI design. This led to mature and highly efficient hypergraph partitioning algorithms such as hMETIS [8]. Such algorithms focus largely on hyperedge-based metrics such as hyperedge cut or sum of external degrees (SOED), whereas systems that allow arbitrary computation and state at both hyperedges and vertices will require more general partitioning objectives.

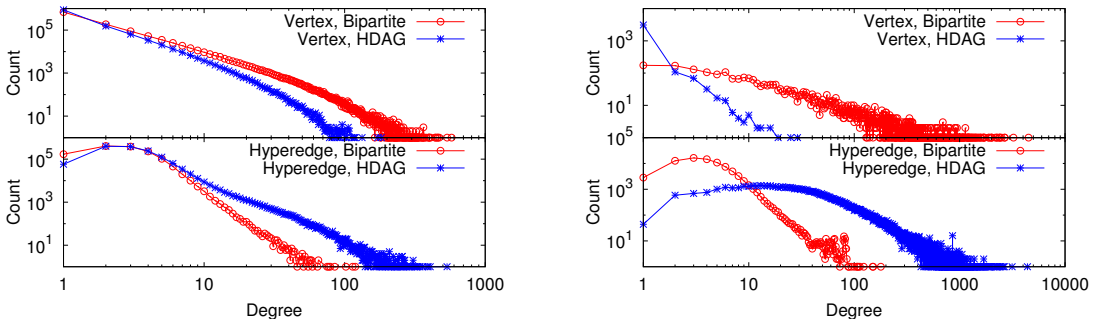
The choice of data representation itself might also depend on this tradeoff. For instance, we find that compared to the bipartite graph representation, the HDAG representation reduces the degree of vertex nodes and increases the degree of hyperedge nodes in the underlying representation, as Figure 4 illustrates. The HDAG representation might therefore be preferable if vertex computations are relatively heavy and dependent on the vertex degrees.

4. MODELING & CHARACTERIZATION

The move beyond graphs toward hypergraphs brings with it new challenges in terms of programming models, application-level modeling, and understanding the characteristics of hypergraphs. Though we only briefly discuss each of these areas, they represent significant opportunities on their own.

Programming models: The restrictive programming models afforded by popular systems such as MapReduce [3], Pregel [11], and Spark [13] have proven highly effective for encoding several data analysis problems. A restricted but still very effective programming model must be developed for hypergraph analysis systems as well, because such restricted models leave room for optimization at the system level while providing sufficient expressibility to the programmers. This may, however, prove to be a significant challenge. In the graph context, system developers had the benefit of a mature and extensive body of efficient graph analysis algorithms from which to make design decisions while developing these programming models. The area of hypergraph algorithms is much less mature, both due to a lack of systems for hypergraph analysis, and also due to application modeling issues (see below). As a result, it is more difficult to identify the common elements that a programming model must support.

Application-level modeling: Though we have argued that hypergraphs are a powerful structure for modeling groups in many domains, such modeling itself may be difficult. In fact, even the question of what constitutes a group can be a difficult one based on the context. As an example, in the DBLP dataset, the concept of a group is rather obvious: individuals that collaborate to co-author a paper have almost certainly engaged in a meaningful group interaction. On the other hand, the concept of a group is more nebulous in a social networking context such as Facebook. Here, it is unclear if a group should be defined as the set of all friends for a user, or all people who interact closely via common wall postings or photos, or perhaps people who are explicitly identified as belonging to a Facebook group. Even if we restrict the choice of a group to those people who interact with each other, we may want to distinguish between those who have interacted recently or often, against those who may have interacted a long time ago or only rarely. The Apache dataset illustrates similar challenges; there we treated all committers to a common file as forming a group, though they may



(a) DBLP (b) Apache

Figure 4: Degree distribution for nodes in the underlying representation. Note that for the bipartite representation, these correspond to vertex degree and hyperedge size, but this no longer holds in the HDAG representation.

have worked on the file at different points of time. Of course some of these modeling issues arise in the dyadic graph setting as well, but the added generality of hypergraphs only magnifies the challenge, and many of these modeling questions are domain-specific.

Hypergraph characterization: The distinctive structure of natural graphs (such as those modeling Internet topology or the structure of the Web) has been the subject of a great deal of research, and is relatively well understood. For example, we know that many natural graphs exhibit power-law degree distributions [5], and that graphs modeling social phenomena tend to have small diameters that shrink rather than grow as vertices and edges are added to the graph [7]. What we do not know, however, is whether these observations apply equally to hypergraphs, and also what other attributes might be important for characterizing hypergraphs. Further, it is possible that natural hypergraphs may display unique characteristics not observed in or relevant to graphs. For instance, the vertices in a social network-based hypergraph may have limited degree due to the limited capacity of humans to engage in social interactions, even though some hyperedges are very large.

From our own early experiments, we suspect that hypergraph instances are likely to differ significantly from one another based on the particular domain that they model. For example, Figure 4 shows the degree distributions of nodes in the underlying representation for both the DBLP and Apache hypergraphs. We can easily see that both hypergraphs exhibit highly skewed distributions, but the particular shapes of these distributions are quite different. Additionally, the impact of using the HDAG representation is also very different in these two hypergraphs. Specifically, compared to the DBLP hypergraph, using the HDAG representation for the Apache hypergraph yields a much greater reduction in vertex degree, but also a much greater increase in hyperedge degrees (in the underlying representation). In fact, for the Apache hypergraph, the HDAG representation actually increases the total space requirement due to the large number of subset relationships but lack of a clear hierarchical structure to these relationships. These results suggest that hypergraphs need to be characterized and classified based on their properties to drive some of the system optimizations.

5. CONCLUSION

Graph theory has been a powerful tool for modeling several phenomena in domains ranging from biology to social network analysis, but we argue that *group* interactions are better modeled by *hypergraphs*. Several major challenges and opportunities arise as we work toward scalable hypergraph analysis systems; we highlight a sample of those challenges here. We consider the need for efficient representations of hypergraphs, and show that in some cases it is possible to exploit the specific structure of a hypergraph to reduce storage overhead. We discuss several challenges in distributing computation on hypergraphs, including the need for more general partitioning approaches. Finally, we discuss several other problems that arise as we move from graphs to hypergraphs, including designing programming models, using hypergraphs to model real-world groups, and the need for a better understanding of the structural characteristics of hypergraphs.

6. REFERENCES

[1] C. Berge. *Graphs and hypergraphs*, volume 6. Elsevier, 1976.

[2] M. Costanzo et al. The genetic landscape of a cell. *Science*, 327(5964):425–431, 2010.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. of OSDI*, pages 137–150, 2004.

[4] E. Estrada and J. Rodriguez-Velazquez. Complex networks as hypergraphs. *Arxiv preprint physics/0505137*, 2005.

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. of SIGCOMM*, pages 251–262, 1999.

[6] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. PowerGraph: Distributed graph-parallel computation on natural graphs. In *Proc. of OSDI*, pages 17–30, 2012.

[7] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Proc. of ICDM*, pages 229–238, 2009.

[8] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE Transactions on VLSI Systems*, 7(1):69–79, 1999.

[9] D. Lazer et al. Computational social science. *Science*, 323(5915):721–723, 6 February 2009.

[10] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *PVLDB*, 5(8):716–727, Apr. 2012.

[11] G. Malewicz et al. Pregel: A system for large-scale graph processing. In *Proc. of SIGMOD*, pages 135–146, 2010.

[12] L. G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33(8):103–111, 1990.

[13] M. Zaharia et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proc. of NSDI*, pages 15–28, 2012.