Hypergraphs, Approximation Algorithms, and the Densest Subgraph Problem

Rebecca Kempe

June 24, 2024

1 The Densest Subgraph Problem

In some sense, the Densest Subgraph Problem (DSP) can be seen as a much looser version of the Maximum Clique Problem. In the Clique problem, we are trying to find the largest subgraph that is complete. If we consider our vertices to be people, and edges to represent a friendship relationship between two people, in the Clique problem we are trying to find the largest friend group in a community.

More formally, if we have a graph $G = (V, E)$ where $|V| = n, |E| = m$, here is the goal of the maximum clique problem:

$$\text{Find the largest } K_c \subseteq G, \text{ where } c \leq n.$$

(Recall that a complete graph on n nodes is denoted K_n. In a complete graph every node is adjacent to every other node, and thus every node has degree $n - 1$.)

The DSP is a little bit different: we are trying to find the subgraph with the highest average degree. We want a group of people where on average, each person is friends with a large number of people. You could also see it as the group where "the most people know each other", or the group that maximizes the ratio of friendships to people.

We define the density $\rho(G)$ of a graph to be the number of edges in the graph divided by the number of vertices. Here is the goal of the DSP:

$$\text{Find some } G' \subseteq G \text{ such that } \rho(G') = \frac{|E'|}{|V'|} \text{ is maximum.}$$

We call the subgraph with the highest density the **optimal subgraph** and denote it G^*. The density of G^* is the **optimal density**, and we denote it ρ^*.

How do we find these subgraphs? Let's take some subset S of V, and define $E(S)$ to be the set of edges with both endpoints in S. We maximize the ratio of edges in $E(S)$ to edges in S. Then,

$$\rho^* = \max_{S \subseteq V} \frac{|E(S)|}{|S|}.$$

We can then denote the subset that maximizes the density by S^*. Therefore, the optimal subgraph is $G^* = (S^*, E(S^*))$ with density $\rho(G^*) = \rho^*$.

2 Hypergraphs

An informal introduction

Before we start, I would like to note that hypergraph theory is not as mature as graph theory, and therefore there is a lot less agreement on things like definitions and notations. I will be presenting one way to look at hypergraphs; it is not the only way to do so, however. Moreover, there are many ways to generalize concepts from graph theory to hypergraphs, as you might discover. For example, I have come across at least two definitions of a directed hypergraph.

Graphs can be seen as a way to represent pairwise relationships between objects. With graphs, we have one object type and one relationship type. For example, in many of the social networking examples we've seen, our object is people, and our pairwise relationship is friendship.

Hypergraphs are easier to look at as a way to represent one to many relationships between objects. Here, we can consider the relationships to involve a primary object (the "one") and a secondary object (the "many"). For example, if we consider our primary object to be people, and our secondary object to be movies, then we can consider our relationship to be a person liking some group of movies. Each primary object is related to an arbitrary number of secondary objects.

(As you might have noticed, this informal definition of a hypergraph allows empty edges (someone might dislike all movies), and it also allows **multiple** or **repeated edges** (someone might like exactly the same set of movies as someone else). The definition of a hypergraph does not allow repeated edges, and typically we don't consider empty edges either.)

It can be helpful to visualize a hypergraph as a pegboard with elastics around different groups of pegs. In that case the primary objects are the elastics, the secondary objects are the pegs, and an elastic is related to a group of pegs if it is wrapped around them.

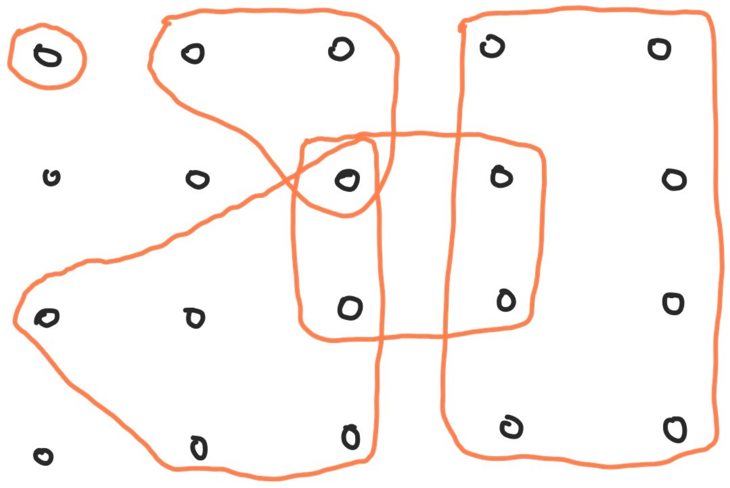

Figure 1: Here, the black circles are the pegs (movies), and the orange curves are the elastics (people). Notice how elastics can be wrapped around any number of pegs.

Some definitions

More formally, a hypergraph is a tuple $H = (V, E)$ where V is a set and E is a family of subsets of V ($E \subseteq \mathcal{P}(V)$). The number of vertices $|V| = n$ is called the **order** of a hypergraph, and the number of edges $|E| = m$ is called the **size** of a hypergraph.

A hypergraph H is called **k-regular** if every vertex in H has degree k, and **k-uniform** if every edge in H contains k vertices.

Note that under this definition, graphs are 2-uniform hypergraphs since their edge set can be written as $E \subseteq \{e \in \mathcal{P}(V) : |e| = 2\}$.

The **rank** $r(H)$ of a hypergraph is the number of nodes in the edge with the most nodes, or the maximum cardinality of an edge in H.

The **maximum degree** of a hypergraph H is denoted $\Delta(H)$, and it is the degree of the vertex with the highest degree. (The degree of a vertex is the number of hyperedges that contain it).

Hypergraph representations

The easiest way to represent a hypergraph is to draw the nodes as labelled points, and draw curves around them, which represent edges. For exmple, here is a drawing of $H = (V, E)$ where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_1 : \{v_1, v_2\}, e_2 : \{v_1, v_2, v_3\}, e_3 : \{v_4\}\}$.

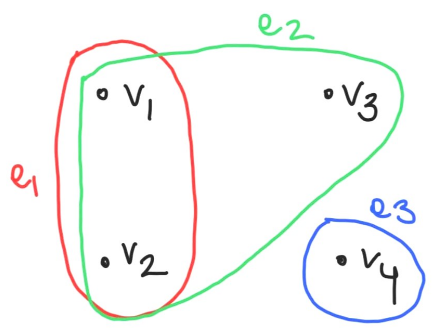

Figure 2: A representation of a hypergraph as a set of circled nodes.

Hypergraphs can also be represented using incidence matrices and incidence graphs. In an incidence graph, we represent both vertices and hyperedges as nodes in the new graph, and put an edge between an edge node and a vertex node if the edge contains that vertex.

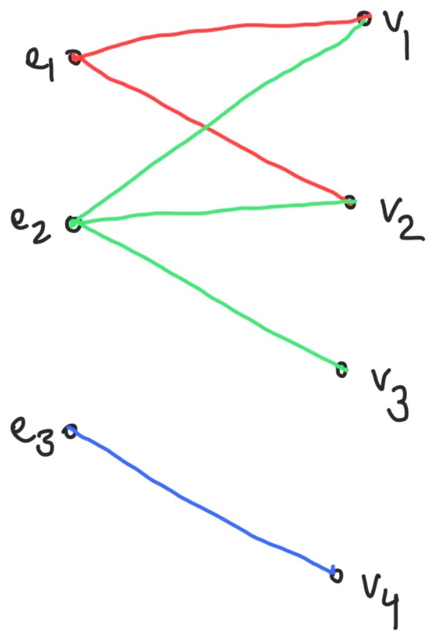

Figure 3: An incidence graph.

In an incidence matrix, each row represents an edge and each column represents a vertex, giving us an $m \times n$ matrix. We enter the number 1 in row i, column j of the matrix if e_i contains v_j, and enter 0 otherwise. This representation is very useful because it allows us to do things like computing the cardinalities of edges (row sums) and the degrees of vertices (column sums) much more systematically.

$$\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline e_1 & 1 & 1 & 0 & 0 \\ e_2 & 1 & 1 & 1 & 0 \\ e_3 & 0 & 0 & 0 & 1 \end{array}$$

Figure 4: An incidence matrix.

Multi-hypergraphs

A further generalization of a hypergraph is a multi-hypergraph, where we allow multiple edges to share the same vertex set. These are defined as a triple $H = (V, E, \mathrm{supp})$ where V is a set of vertices, E is a set of edges, and the **support function** $\mathrm{supp} : E \to \mathcal{P}(V)$ maps individual edges to subsets of the vertex set.

This definition becomes very natural from the point of linear algebra, because then we can look at an edge as being a row in an $m \times n$ matrix where all entries are either 0 or 1 (and any matrix over $\{0,1\}$ can be represented in this way).

This definition also allows us to model one to many relationships more effectively.

Generalizing the DSP for hypergraphs

While the maximum clique problem does not generalize very nicely for hypergraphs (we would most likely want to redefine it for k-uniform hypergraphs instead), the densest subgraph problem generalizes very nicely. In fact, we can almost copy everything we said about graphs word for word, replacing every instance of a graph with a hypergraph. However, we would have to redefine the set $E(S)$ slightly, in the following way: $E(S) = \{e \in E | e \in \mathcal{P}(S)\}$. Defining the optimal density for multi-hypergraphs is also slightly messier:

$$\rho^* = \max_{S \subseteq V} \frac{|\{e \in E : \mathrm{supp}(e) \subseteq S\}|}{|S|}$$

Another thing to note is that since hypergraphs do not have the same assumption of uniformity as graphs, the concept of the densest subgraph having the "highest average degree" might also not translate very well.

3 Approximation Algorithms

Until now, we have spent a lot of time discussing what the DSP is, but not how to actually find the densest subgraph. Unlike the maximum clique problem, which is NP-Complete, the

DSP is known to be solvable in polynomial time. There are several known algorithms which compute the exact value of the density of the optimal subgraph. However, when viewing algorithmic complexity in real-world contexts, things like constants can actually matter. It is also possible for constants to get so large that the algorithm is functionally unusable. (See this page on "galactic algorithms".) Most of the known exact algorithms for solving the DSP are technically polynomial time algorithms, but they break down on real world graphs due to requiring too much time or too much memory.

When exact algorithms prove too cumbersome, approximation algorithms can be an alternate method for attempting to solve problems. Instead of designing an algorithm that always produces the optimal answer, one can design an algorithm that always gets "close enough" to the optimal answer. While there is a tradeoff in terms of accuracy, there are generally huge gains in terms of speed or space complexity.

Formally, we define approximation algorithms in the following way: Let P be a problem (this means P is the space of all possible inputs, for example, the set of all graphs in this case), and let $I \in P$ be an instance to the problem (I is a particular input, such as a specific graph G). Let $OPT(I)$ denote the value of the optimal solution for the problem (this would be ρ^* in this case). An algorithm $\mathcal{A}$ is called an α-approximation for P if:

For a maximization problem: $\forall I \in P,\ \alpha \in (0,1],\ \alpha \cdot OPT(I) \leq \mathcal{A}(I) \leq OPT(I)$ (lower bound).
For a minimization problem: $\forall I \in P,\ \alpha \geq 1,\ OPT(I) \leq \mathcal{A}(I) \leq \alpha \cdot OPT(I)$ (upper bound).

Note: there is an alternative definition for max problems, which redefines α as being a number ≥ 1 and $\mathcal{A}(I)$ as being greater than or equal to $\frac{1}{\alpha} \cdot OPT(I)$. This is to use the use of the same (integer) constants for both max problems and min-problems. While this notation is also in use, I don't like it and therefore will not be using it.

If I have a $\frac{1}{5}$-approximation for a maximization problem, that means the algorithm always returns a value that is at least 20% of the optimal value. Approximation algorithms always return a value that is within the factor of approximation (can I actually call it that?).

The "gold standard" for an approximation algorithm is a $(1-\epsilon)$-approximation algorithm. These are algorithms which allow you to specify how much error (ϵ) you're willing to tolerate, at the cost of the number of iterations or runtime of the algorithm depending on ϵ in some way (usually some power of ϵ is in the denominator).

4 The Peeling Algorithm(s)

One of the most famous papers about the DSP was written by Charikar in 2000, and it details a $\frac{1}{2}$-approximation algorithm for solving the DSP. This procedure is called "peeling", and is as follows:

1. Remove the vertex of lowest degree and all of edges which contain it; keep track of which vertices were removed (in that order).
2. If a vertex has degree zero, remove it and add it to the ordering of removed vertices.
3. After the removal of each vertex, note the density of the remaining graph.
4. Once all vertices have been removed, find out which vertex removal left behind the subgraph of highest degree. Return the graph induced by the remaining vertices.

We call this peeling because we "peel off" the vertices of lowest degree at each step. This algorithm is so useful because a single iteration of this algorithm guarantees a subgraph that has at least 50% of the optimal density; in practice, on real-world graphs, it is closer to 80% of the optimal density on most graphs.

In 2019, Boob et. al published a paper detailing an interative version of greedy peeling that they conjectured was a $(1 - \epsilon)$-approximation for the DSP. They called this algorithm **Greedy++** (because computer scientists are great at naming things), and relied on the concept of updating "loads" over time. In 2022, Charikar, Quanrud, and Torres published a paper in which they validated this claim, and presented a version of this algorithm which works for all supermodular functions (creatively named **SuperGreedy++**).

5 References, Resources, Further Reading

Getting Involved in Research at Carleton!

Information about DSRI, for incoming first years: https://science.carleton.ca/students/undergraduate-student-summer-research-opportunities/deans-summer-research-internships/

Black and Indigenous Summer Research Internships: https://science.carleton.ca/students/undergraduate-student-summer-research-opportunities/black-and-indigenous-summer-research-internships/

What are NSERC Undergraduate Student Research Awards?: https://www.nserc-crsng.gc.ca/students-etudiants/ug-pc/usra-brpc_eng.asp

How to apply for a USRA in the Faculty of Science: https://science.carleton.ca/students/undergraduate-student-summer-research-opportunities/undergraduate-research-awards-program/

More Detailed Introductions

Chekuri's Talk at the Tutte Combinatorics and Optimization Seminar (Waterloo), about the DSP and Supermodularity (basically, he introduces the DSP and talks about his paper):

https://www.youtube.com/watch?v=DO8iNCwmp0c

A YouTube video series on hypergraph theory, by Vital Sine: https://youtube.com/playlist?list=PLZ2xtht8y2-IRjvXJJpka2GIJ3wn7u4g2 (pretty accessible)

One of the only books on hypergraph theory that currently exists (it might be a lot to try to read right now though...): Hypergraph Theory: An Introduction by Alain Bretto

Chandra Chekuri's introductory lecture on Approximation Algorithms:
https://courses.grainger.illinois.edu/cs583/sp2018/Notes/intro.pdf

Lalla Mouatadid's notes on Approximation Algorithms: http://www.cs.toronto.edu/~lalla/373s16/notes/Intro2Approx.pdf

Important Papers

Yuly Billig (my advisor) wrote a paper detailing an exact algorithm for solving the DSP for hypergraphs: Dense Clusters in Hypergraphs. This algorithm is different from the others in that it heavily relies on techniques from linear algebra.

The original paper on the greedy peeling algorithm for the DSP, by Moses Charikar (very influential): Greedy Approximation Algorithms for Finding Dense Components in a Graph

A paper conjecturing that we can approximate the DSP for graphs using iterative peeling (Greedy++): Flowless: Extracting Densest Subgraphs Without Flow Computations

Extending the iterative peeling algorithm to supermodular functions, and proving that it's a $(1-\epsilon)$-approximation for the DSP (SuperGreedy++): Densest Subgraph: Supermodularity, Iterative Peeling, and Flow

A paper about different variations of the DSP: A Survey on the Densest Subgraph Problem and Its Variants