Positive codegree thresholds in 3-graphs

Talk by Anna Halfpap

Notes by Sanjana Das

January 11, 2025

This is joint work with Van Magnan.

§1 Introduction

§1.1 Dirac's theorem

With extremal problems on graphs, we often have some structure that we forbid, and we think about what different graphs or hypergraphs can look like while avoiding that structure. In particular, often there's some parameter that we want to maximize, subject to the condition of not having the forbidden substructure.

Maybe one of the most famous examples is the following question:

> **Question 1.1.** How many edges can you have in an n-vertex graph with no triangle?

This is answered by Mantel's theorem, a classic result from 100 years ago. And it leads down a path of Turán-flavored problems in extremal graph theory.

But we'll be motivated by a question of slightly different flavor:

> **Question 1.2.** What's the largest possible minimum degree in a graph with no Hamiltonian cycle?

This is answered by Dirac's theorem, which is also a classic result in graph theory.

This is a question of a similar flavor in some sense — we want to make a certain parameter big while not seeing some subbstructure. But it's also a bit distinct — the subgraph we're forbidding is not fixed (like a triangle, which looks the same no matter what n is), but instead grows as n does. So the techniques we use are also different.

In this talk, we'll think about questions of this flavor. In particular, we'll consider *spanning* structures — subgraphs that contain all the vertices of our graph (e.g., Hamiltonian cycles, Hamiltonian paths, perfect matchings, and so on).

> **Question 1.3.** What conditions on G guarantee that you'll see some spanning structure of interest?

For example, for Hamiltonian cycles, there's Dirac's theorem (as mentioned earlier).

> **Theorem 1.4** (Dirac)
> If G has minimum degree $\delta(G) \geq n/2$, then it has a Hamiltonian cycle.

§1.2 Hypergraphs

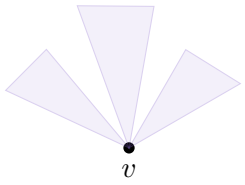
Instead of graphs, we'll be working with hypergraphs.

> **Question 1.5.** Can we prove Dirac-type theorems for hypergraphs?

First, there's multiple good notions of minimum degree in a hypergraph.

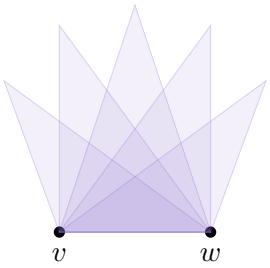
One thing you could do is look at the degrees of individual vertices — so we look at each vertex, count how many hyperedges it's in, and define the minimum degree of the hypergraph as the smallest such number.

> **Definition 1.6.** The minimum degree of a hypergraph H is defined as
>
> $$\delta(H) = \min_v \#(\text{edges containing } v).$$



But you can also look at *codegree*, where instead of just looking at one vertex, we consider a larger set of vertices. For ordinary graphs, we're looking at sets of $1 = 2 - 1$ vertices; so in r-graphs, we could generalize by looking at sets of $r - 1$ vertices. In the case $r = 3$, we're looking at a pair of vertices, and counting how many edges that pair is in.

> **Definition 1.7.** The minimum codegree of a 3-graph H is defined as
>
> $$\delta_2(H) = \min_{v \neq w} \#(\text{edges containing } v \text{ and } w).$$



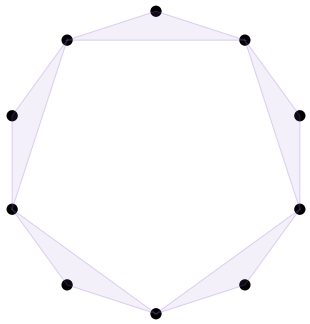
The notion we'll focus on is minimum *positive* codegree, where we ignore pairs which are not in any edges.

> **Definition 1.8.** The minimum positive codegree of a 3-graph H is defined as
>
> $$\delta_2^+(H) = \min \#(\text{edges containing } v \text{ and } w),$$
>
> where the minimum is only taken over pairs $\{v, w\}$ contained in at least one edge.

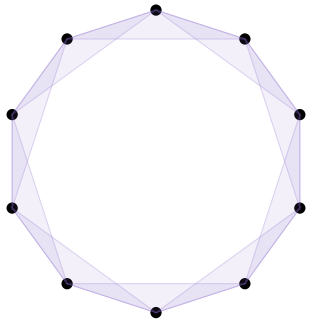
In other words, if we say a graph has minimum positive codegree at least δ, we're saying that once we know a pair of vertices is in at least one edge, then they're guaranteed to be in at least δ.

(For r-graphs, the minimum codegree and minimum positive codegree, denoted $\delta_{r-1}(H)$ and $\delta_{r-1}^+(H)$, are defined in the same way by looking at sets of $r - 1$ vertices.)

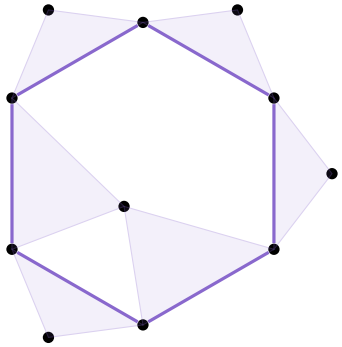
There's also several ways to generalize Hamiltonian cycles. In this talk, we'll focus on *loose* Hamiltonian cycles — this means we have a bunch of vertices in a circle and edges go around the circle, with consecutive edges intersecting in exactly one vertex.



One reason there's different notions of cycles is that you could have different levels of intersection between consecutive edges — in a *tight* cycle, consecutive edges intersect in $r-1$ vertices instead of one.



There's also Berge cycles, which are weird things that don't necessarily look that cyclic when you draw them, but do have some underlying cyclic structure keeping track of pairs of vertices with positive codegrees.



This means there's tons of different work people could do on Dirac-type theorems in hypergraphs, because there's all these different parameters and types of Hamiltonian cycles you could look at. To highlight one of these results (because its proof structure is relevant):

Theorem 1.9 (BHS)

For all ε, if n is sufficiently large and $\delta(H) \geq (7/16+\varepsilon)\binom{n}{2}$, then H contains a loose Hamiltonian cycle.

(We use n to denote the number of vertices of H.)

§1.3 Results

We want to prove versions of such results for minimum positive codegree. First, here's one for Berge cycles.

Theorem 1.10 (Halfpap–Magnan 2024+)
If H is a n-vertex r-graph with $\delta^+_{r-1}(H) \geq n/2 - r + 2$ and H does not have isolated vertices, then it has a Berge Hamiltonian cycle.

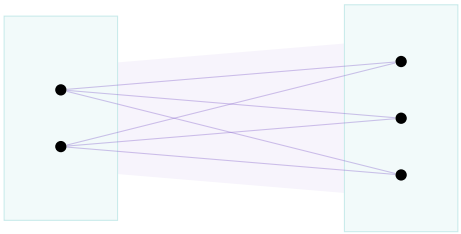
When $r = 2$, this bound becomes $n/2$, so we get back Dirac's theorem. You can prove Theorem 1.10 by induction on n, starting from Dirac's theorem — here things work out and you get a direct generalization.

Hre's a result for loose cycles (in uniformity 3).

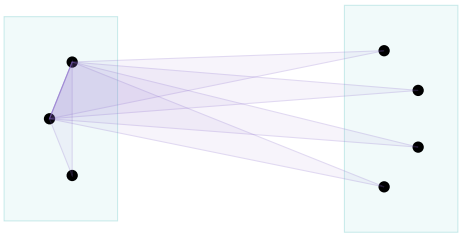
Theorem 1.11 (Halfpap–Magnan 2024+)
If a 3-graph H has no isolated vertices and $\delta^+_2(H) \geq (1/2 + \varepsilon)n$, then it has a loose Hamiltonian cycle.

We'll spend most of the time talking about the proof of this result.

First, the constant 1/2 is tight. To show that Dirac's theorem (for ordinary graphs) is tight, you take a slightly unbalanced complete bipartite graph — then a Hamiltonian cycle would force you to bounce back and forth between the two parts, but you can't do that and cover all vertices.



We can use the same idea here (for 3-graphs) — we take two sets U on the left and V on the right, where V is a tiny bit bigger than U. We make U *universal*, meaning that we take all edges inside U and all edges with two endpoints in U and one in V. Meanwhile, we make V *strongly independent*, meaning that we don't include any edges with more than one endpoint in V.



This can't have a Hamiltonian Berge cycle or loose cycle — if it did have one, then that cycle would give a cyclic ordering of the vertices, where any time two vertices are next to each other in the cycle, they have to be in an edge together. But more than half our vertices are in V, so there are going to be two vertices in V which are next to each other.

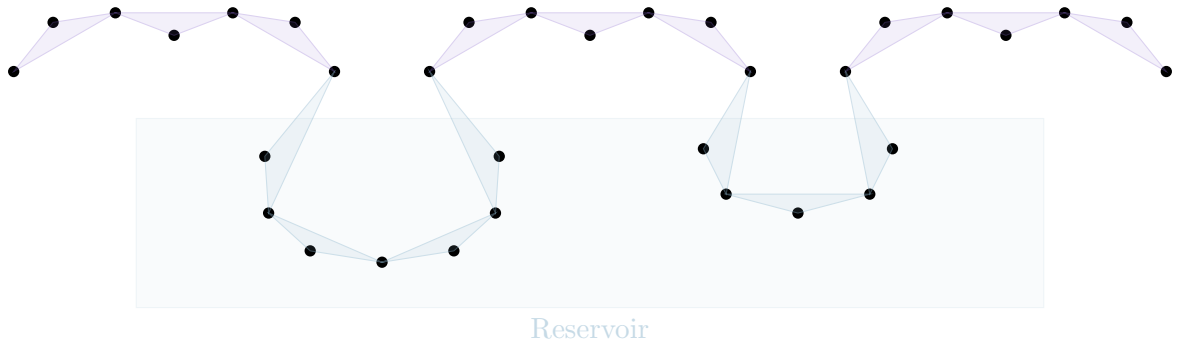
This is a nice thing about positive codegrees — not only do we get the same *answer* as for ordinary graphs (the constant 1/2), but the *constructions* also look very similar. Meanwhile, the story for ordinary codegree is quite different (for both the answer and the construction); this graph has minimum codegree 0.

§2 The absorbing method

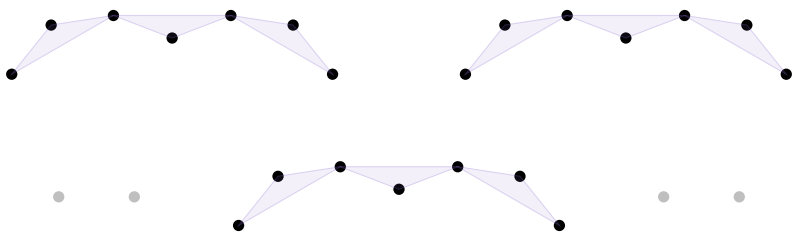
To prove Theorem 1.11, we're going to use machinery from the absorbing method. There's lots of technical detail that we won't talk about, but here's some intuition about how the flavor of the absorbing method helps us find a spanning structure.

There's three things we want to find in our graph, which will come together to make a loose Hamiltonian cycle. First, we want to find two special small structures. The first is an *absorber* — this is a small loose path with the special property that any time I take a small extra set of vertices, I can tuck them in between the vertices of this path to get a slightly longer loose path. So the absorber can 'eat' any small set of vertices (making a longer path with the same endpoints).

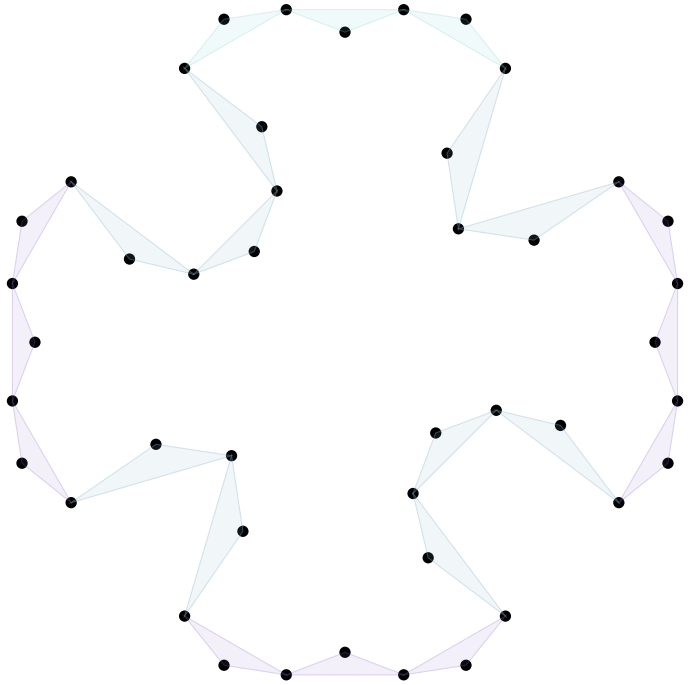
The second special structure is a *reservoir*. This is a small set of vertices that has a connecting path where if I have some bounded number of loose paths and want to stitch them together (i.e., to bridge the end of one path to the beginning of another), then I can do this stitching using edges from my reservoir (as long as I don't have too many of these paths).



For almost everything else (the vertices not in the absorber or reservoir), we want a tiling consisting of not too many loose paths. This tiling doesn't have to be perfect, but it should cover *almost* everything else.



Here's how these pieces come together: First, we take the reservoir and use it to stitch together the absorber and all the loose paths in our almost tiling (we can do this because we don't have too many paths in the tiling). Now we have a loose cycle — a bunch of loose paths stitched together.



This loose cycle is not quite Hamiltonian, because there might be a few vertices that were missed by our almost tiling, or some that were in the reservoir but weren't used. But we're not missing too many vertices — this leftover is small enough that we can absorb it into the absorber, in order to get a Hamiltonian cycle.

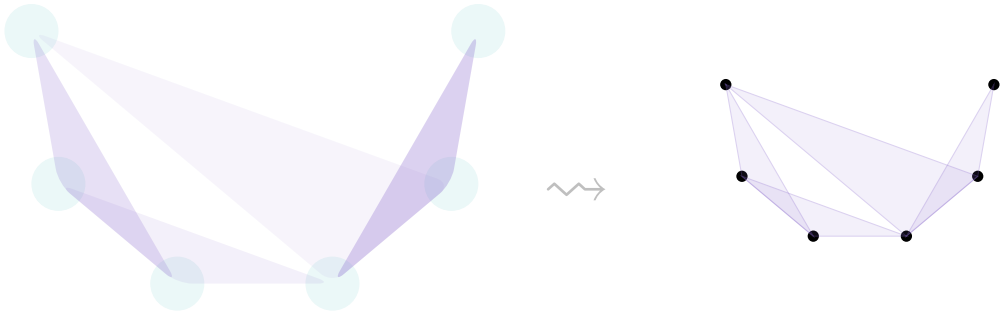
So if we can find things with these magical properties (i.e., the absorber, reservoir, and almost tiling), this seems like a great idea for building a Hamiltonian cycle. But how do we find these things?

We have 7 minutes, so we can't talk about all the details. For the absorber and reservoir, there are some lemmas from the BHS result (Theorem 1.9) that we could use directly — the condition on minimum positive codegree also implies some condition on minimum degree. For many of the parts that the authors had to do from scratch, you could use fairly standard techniques (e.g., probabilistic tools). In the remaining time, we'll mostly focus on constructing the almost tiling.

§3 The almost tiling

To construct the almost tiling, the idea is that we use weak hypergraph regularity to get a regularity partition, and this partition indicates some places where it's easier to find almost-spanning loose paths.

First, here's what regularity looks like: If you have a really big hypergraph, then you can magically partition it into not-too-many classes with 'nice' properties (where they have the same size, and the distribution of edges between almost all triples of classes is really nice in some ways). You might have some flaws — there's a few edges that live inside a class or that intersect a class in two places, and a few triples of classes where the edges between them are not too nice — but we can delete those flaws and get a subgraph with *really* nice edge distribution. Then we shrink to the *cluster graph* where we contract each class into a single vertex, and draw an edge if there was a positive density of edges between the original classes.



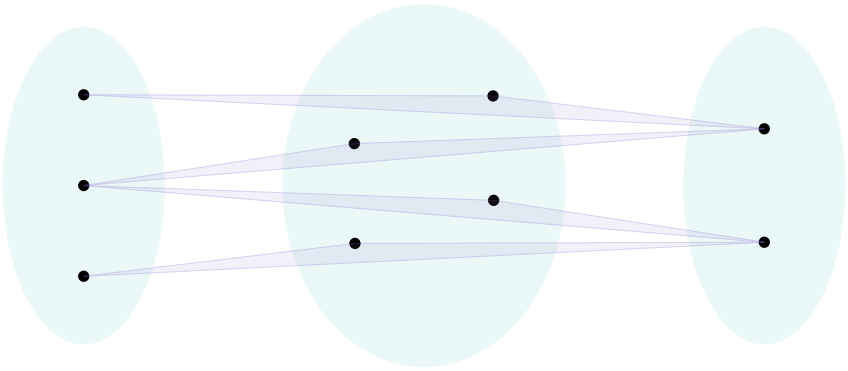
There are some technical details we're sweeping under the rug about how this interacts with the minimum positive codegree. Our eventual goal is to use this cluster hypergraph to find some structure that tells us how to find loose paths tiling our big hypergraph. And for this, we need a minimum positive codegree condition on the cluster hypergraph, so we need the minimum positive codegree to not get wrecked during this process. This could potentially be a problem — the cleanup step involves deleting $o(n^3)$ edges, but codegrees are on the order of n, so they could get messed up. But there are some tricks that make things work out (e.g., we can fix these issues by deleting a few more edges), and we can ensure that this cluster hypergraph satisfies the same minimum positive codegree condition — if it has m vertices, it has minimum positive codegree at least $(1/2 + \varepsilon)m$ (for a modified ε).

§3.1 Tiling nice configurations of classes

First suppose that we can find four vertices in the cluster graph with two edges between them. This corresponds to four classes in the original graph which have two nicely distributed sets of edges; we'll show that we can tile these classes nicely with loose paths.

First, we can cut the two middle classes in half; then we have two triples of classes where each consists of one big part and two small parts, with regularly distributed edges between them.

Then there's a fairly standard proposition that we can find an almost spanning loose path on these three parts — we essentially start in one small part, bounce to the big part and then to the next small part, then bounce back to the big part and then to the first small part, and so on; and regularity guarantees that we can cover almost everything.



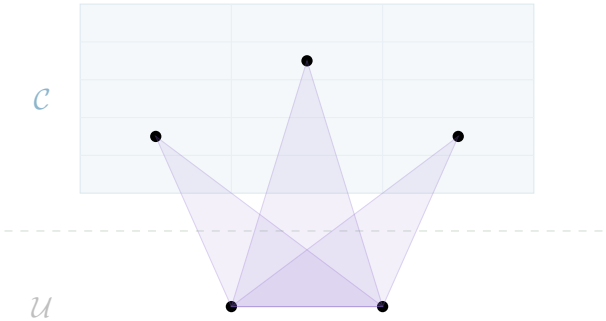
Now it's enough to figure out how to split our regularity-partitioned graph into pieces of this shape; in other words, we want to tile our cluster graph by pictures that look like the following.

§3.2 Tiling the cluster graph

At this point, we can forget all the regularity stuff and just focus on the cluster graph. We've done some things to ensure that the cluster graph inherits the minimum codegree condition (i.e., its minimum positive codegree is a $(1/2+\varepsilon)$-fraction of its number of vertices). And we want to find tiles of the above shape that cover almost all the vertices.

This is a nice averaging argument. Let's assume that we can't do this — so we take a maximal tiling and assume that the number of vertices it leaves uncovered is large (i.e., more than an α-fraction, where α is a very small but fixed constant). Our goal is to get a contradiction by somehow constructing a bigger tiling.

There's a couple of averaging steps; the first uses the idea of a *switching configuration*. Suppose that we have our tiling $\mathcal{C}$ and a set of uncovered vertices $\mathcal{U}$, and suppose there are some pairs of vertices in $\mathcal{U}$ with positive codegree. Then those pairs will have a bunch of edges coming off them, and most of those edges are going to go into our tiling $\mathcal{C}$.

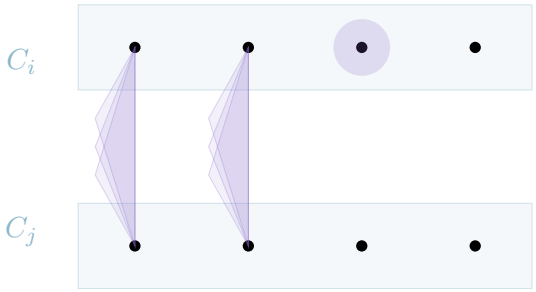


Suppose that two of these edges go into the *same* tile C. Then we can make a tiling that's no smaller by taking these two vertices, taking out C, and adding back the tile consisting of these two uncovered vertices and the two vertices in C that they had edges to.

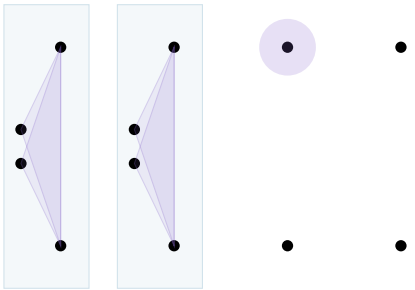
And if we have *two* of these switching configurations that involve the same tile C and are disjoint, then we can take out C and add in *two* tiles, and that contradicts the maximality of our tiling.

So we do an averaging argument to say how often we see these switching configurations; and we can show that if we have three pairwise disjoint positive codegree cherries in $\mathcal{U}$ (i.e., pairs of vertices in $\mathcal{U}$ with positive codegree), then we're able to switch to a *bigger* tiling.

Now we know not only that $\mathcal{C}$ is a maximal tiling, but also that $\mathcal{U}$ can't have three disjoint positive codegree cherries — this means it has some property of being very sparse. For the last step, we do some more averaging to show that we can then switch from $\mathcal{C}$ to a tiling $\mathcal{C}'$ that has the same size, but where the uncovered set $\mathcal{U}'$ now *does* have three disjoint positive codegree cherries (which would contradict the maximality of $\mathcal{C}'$). The idea for this is to find pairs of tiles C_i and C_j which have a bunch of matched pairs in U — specifically, we have two vertices in each which form pairs with lots of neighbors in U, and we have one leftover vertex which also has high degree with vertices in U.



Then we can switch out C_i and C_j for two tiles formed from the pairs on the left.



When we do this switching, our one leftover vertex ends up going into $\mathcal{U}$; and since this leftover vertex had lots of neighbors in $\mathcal{U}$, we can use this to manufacture these positive codegree cherries.

So this is how we get our tiling of the cluster graph with pieces of the desired form; and once we get that, we lift this back to the original graph and get the almost tiling by loose paths that we needed.