H-freeness Testing in Bounded Admissibility Graphs

By Oded Lachish

Joint Work with

Christine Awofeso, Patrick Greaves and Felix Reidl

Content

- H-freeness?
- H-freeness Testing?
- Why is *H-freeness* Testing easy in bounded degree graphs?
- Why is it not at all easy in bounded average degree graphs?
- *2-admissible graph*?
- Why is testing *C_4-freeness* easy in bounded *2-admissibility graphs*?

Open questions?

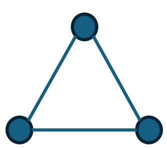
H-freeness (family of graphs)

Let *H* be a graph

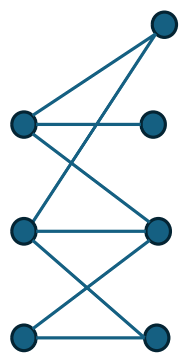
A graph *G* is *H-free*, if it does not have a subgraph *H'* that is isomorphic to *H*

For example,

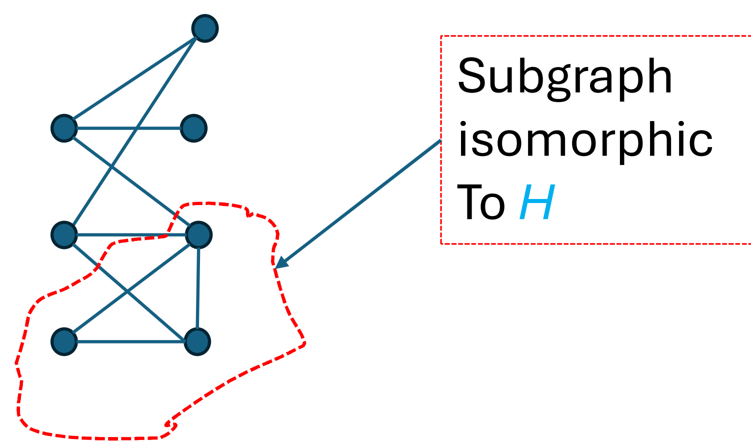
H



H-free graph



A graph that
is not *H-free*



Why is H-freeness important?

- It is a fundamental problem in Computer Science
- It has applications in Bioinformatics, Network Science etc

H-freeness in property testing?

- O. Goldreich and D. Ron. *Property testing in bounded degree graphs*. STOC'97
- A. Czumaj and C. Sohler. *A characterization of graph properties testable for general planar graphs with one-sided error (it's all about forbidden subgraphs*. FOCS'19
- T. Eden, R. Levi, and D. Ron. *Testing c_k-freeness in bounded-arboricity graphs*. ICALP'24

H-freeness in Property Testing

What if you had to pay for access to the input graph?

- The input is the size of a graph $G = ([n],E)$ and oracle access to G

Oracle access: access to an oracle that answers these queries:

- What is the degree of a vertex k ?
- What is the i'th neighbour of a vertex k ?
- Is the edge $\{i,j\}$ in the graph?

Each answer costs a pound (computation is for free)

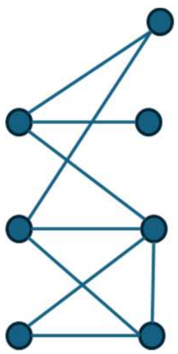
You barely have any money

H-freeness in Property Testing

Oracle access: access to an oracle that answers these queries:

- What is the degree of a vertex *k* ?
- What is the *i*'th neighbour of a vertex *k* ?
- Is the edge *{i,j}* in the graph?

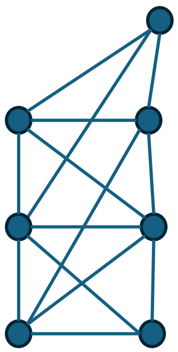
There is a good chance that the money will run out way before an *H-subgraph* (subgraph of *G* isomorphic to *H*) !



H-freeness in Property Testing

Oracle access: access to an oracle that answers these queries:

- What is the degree of a vertex *k* ?
- What is the *i*'th neighbour of a vertex *k* ?
- Is the edge *{i,j}* in the graph?



Relaxed problem:

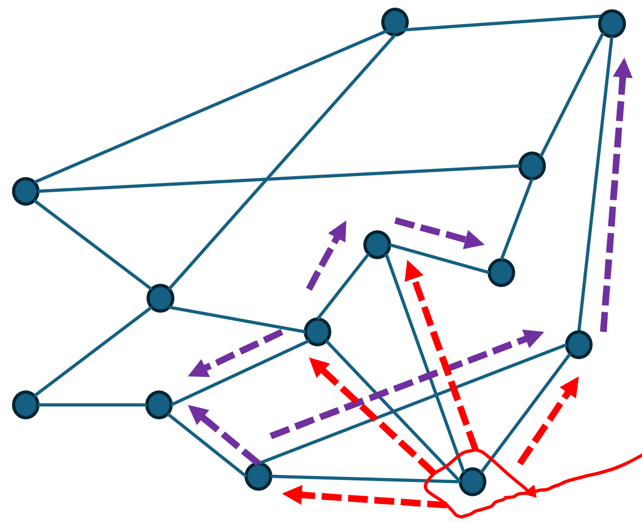
- Your algorithm can use randomness and is only required to find an *H-subgraph* if *G* is far from *H-freeness*, and even then, only with probability at least *2/3*

What if the graph is not far from *H-freeness*?

Then there is no requirements on its behaviour !

Testing triangle freeness in <u>**bounded degree**</u> graphs

Bounded degree: there exists a fixed constant *d* such that the degree of every vertex in the graph is at most *d*.



- Pick a random vertex
- Find its neighbours
- Find its neighbour's neighbours

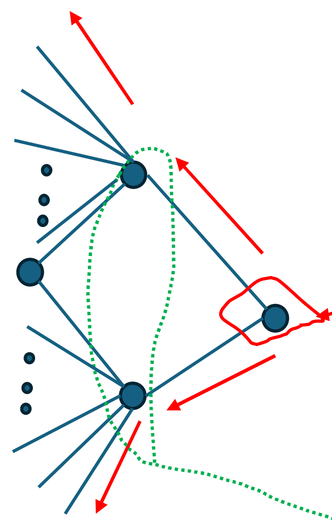
(repeat a few times)

The total number of queries is $O(d^2)$, which is independent of the graph size (cheap)

Testing C_4 freeness in bounded **average** degree graphs

- Pick a random vertex
- Find its neighbours
- Find its neighbour’s neighbours

(repeat a few times)



Two vertices with a very high degree

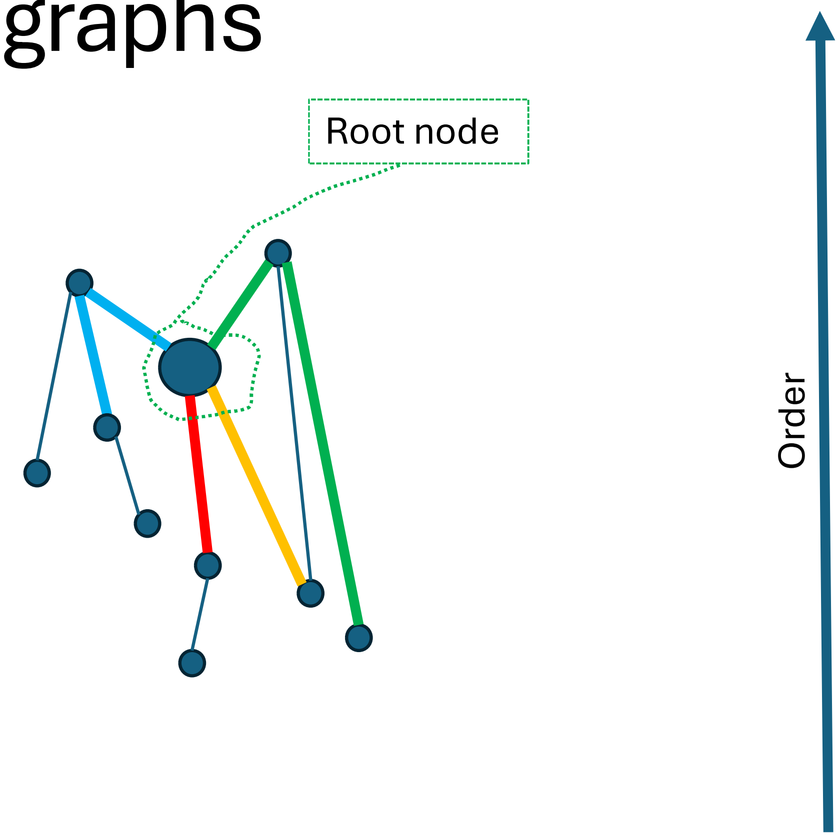
For C_4 freeness this problem is inherent even if the graph has bounded degeneracy (arboricity, 1-admissibility). There is a lower bound (nice question for advanced students)

C_4 – cycle of length 4

Bounded *2-admissibility* graphs

Any graph for which there exists a special complete order on its vertices that satisfies:

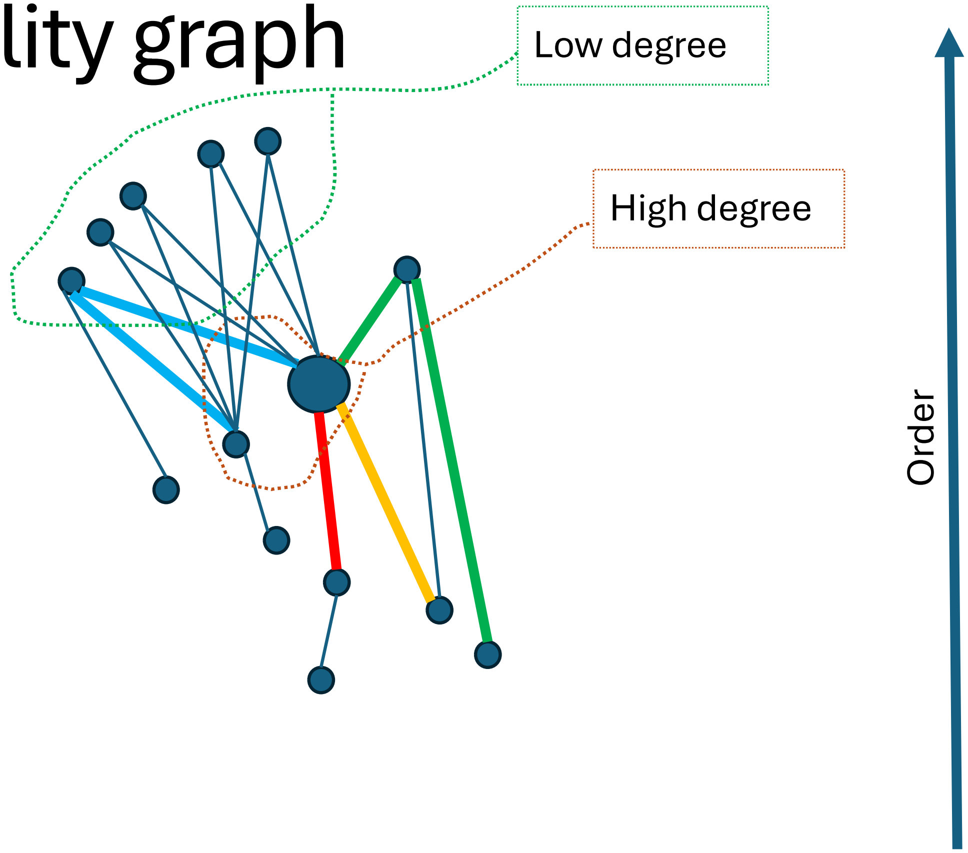
There exists *p* such that every vertex in the graph has a maximum *2-admissible packing* of size (number of paths) at most *p*



Every monochromatic path in the graph is a 2 admissible path, all the paths in packing share only their first vertex.

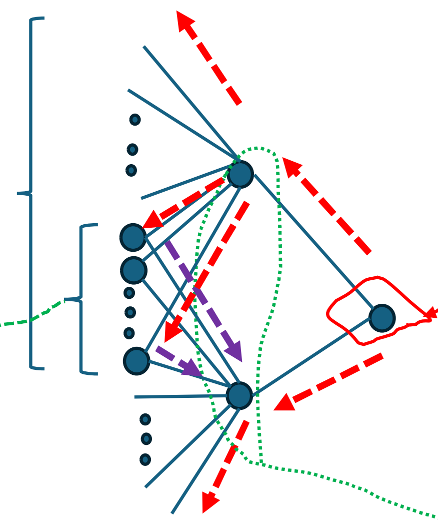
Bounded 2-admissibility graph

If two high degree vertices do not share a large portion of their neighbours, C_4-subgraphs including both vertices do not contribute much to the distance of the graph from C_4-freeness



We can also assume that high degree vertices are not neighbours!

New algorithm (pseudo-BFS)



Low degree

Two vertices with a very high degree

- Pick a random vertex
- Find its neighbours find a random subset of α its neighbours
- For each vertex found find a random subset of α its neighbours
- Repeat the previous

(repeat a few times)

This is the main idea, there is still much work to do to get it to work

Open Questions?

- Properties that are not subgraph freeness