Information Processing Letters 116 (2016) 682–688

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Hardness of peeling with stashes

Michael Mitzenmacher[a,1], Vikram Nathan*

[a] Harvard University, School of Engineering and Applied Sciences, United States

ARTICLE INFO

Article history:
Received 12 February 2015
Received in revised form 31 May 2016
Accepted 31 May 2016
Available online 4 June 2016
Communicated by R. Uehara

Keywords:
NP-complete
Graph algorithms
Peeling process
Stash
Hashing

ABSTRACT

The analysis of several algorithms and data structures can be framed as a *peeling process* on a random graph: vertices with degree less than k and their adjacent edges are removed until no vertices of degree less than k are left. Often the question is whether the remaining graph, the k-core, is empty or not. In some settings, it may be possible to remove either vertices or edges from the graph before peeling, at some cost. For example, in hashing applications where keys correspond to edges and buckets to vertices, one might use an additional side data structure, commonly referred to as a stash, to separately handle some keys in order to avoid collisions. The natural question in such cases is to find the minimum number of edges (or vertices) that need to be stashed in order to realize an empty k-core. We show that both these problems are NP-complete for all $k \geq 2$, with the sole exception being that the edge variant of stashing is solvable in polynomial time for $k = 2$ on standard (2-uniform) graphs.

1. Introduction

The following *peeling process* can be used to find the k-core of a hypergraph: vertices with degree less than k are repeatedly removed, together with their associated edges. The k-core is easily shown to be the maximal subgraph where each vertex has degree at least k; it is therefore uniquely defined and does not depend on the order vertices are removed in the peeling process. Peeling processes, and variations on it, have found applications in low-density parity-check codes [7,9], hash-based sketches [3,5,6], satisfiability of random boolean formulae [2,8,10], and cuckoo hashing [4,11]. Usually in the design of these algorithms the primary question is whether or not the k-core is empty, and an empty k-core corresponds to a success. We say that a hypergraph is k-*peelable* if it has an empty k-core.

If the k-core is not empty, a natural question to ask is how many edges or vertices need to be removed to yield an empty k-core. This question may have algorithmic implications. For example, consider a multiple-choice hash table of the following form. There are n keys and m buckets; each key has d possible buckets where it can be placed; and each bucket can hold at most $k-1$ keys. By associating buckets with vertices and keys with hyperedges (each key being an edge of its d possible buckets), we see that peeling can naturally provide an assignment of keys to buckets satisfying the constraints. When a vertex is removed, the bucket obtains all the keys corresponding to adjacent edges; if the peeling yields an empty hypergraph, all keys have been placed. In this setting, removing an edge from the hypergraph before peeling can correspond to placing a key into a separate structure, often referred to as a stash. If a suitably sized stash can be implemented, peeling can efficiently find an assignment, leading to the

* Corresponding author.
E-mail address: nathan.vikram@gmail.com (V. Nathan).
[1] Supported in part by NSF grants CCF-0915922, IIS-0964473, and CNS-1011840.

http://dx.doi.org/10.1016/j.ipl.2016.05.006
0020-0190/© 2016 Elsevier B.V. All rights reserved.

question of how many edges need to be removed so the remaining hypergraph is k-peelable.[2] Even without such algorithmic implications, the minimum number of vertices or edges to remove to obtain a k-peelable graph appears a natural and interesting graph theoretic question.

Here we prove that determining the minimum number of vertices, or the minimum number of edges, that need to be stashed (removed from the graph) so that a d-regular hypergraph is k-peelable is NP-complete for any $k, d \geq 2$. The only exception is that, in the case of edges, determining the minimal stash is solvable in polynomial time for $k = d = 2$; that is, for 2-cores on standard graphs. Although we believe the case of $k = d = 2$ is well understood, we briefly review it Section 3. Given the connection to stashes for cuckoo hash tables, we call this class of problems *stash problems*.

We note that a similar problem was recently considered in [1]. In their variation, they look at the *anchoring problem*: given a budget b, find the subset B of b vertices such that peeling the graph of vertices from $V - B$ of degree less than k yields the maximum number of remaining edges. That is, the chosen subset B cannot be peeled, and serves as an anchor for the k-core. They show that when $k = 2$, the problem is solvable in linear time, and when $k \geq 3$ the problem is NP-hard and further is NP-hard to approximate within an $O(n^{1-\epsilon})$ factor for any constant ϵ. One way of viewing the anchoring problem is that it *adds* to the graph; for example, by replacing a vertex in B by a clique of size at least k (and appropriately connecting edges), one can guarantee that vertex is anchored. In contrast, our goal in stash problems is to *remove* vertices or edges from the graph.

In what follows, we define terms and briefly consider the case $k = d = 2$. We then show the problem of finding the optimal stash size when stashing vertices is NP-complete for $k \geq 2$ by a reduction from Vertex Cover. We then reduce the problem of finding the optimal stash size for vertices to the problem for edges.

2. Notation and definitions

Recall that we say that a hypergraph is k-peelable if it has an empty k-core. A *k-vertex-stash* of a hypergraph is a subset of vertices $V \subset G$ such that $G - V$ is k-peelable. (Of course removing a vertex also removes all adjacent edges.) Similarly, a *k-edge-stash* of a hypergraph is a subset of edges $E \subset G$ such that $G - E$ is k-peelable. For fixed values k and d, the decision problem k-VERTEX-STASH(G_d, s) (respectively k-EDGE-STASH(G_d, s)) asks whether the d-regular hypergraph G_d has a minimal k-vertex-stash (respectively k-edge-stash) of size at most s. We use k-VERTEX-STASH and k-EDGE-STASH where the meaning is clear. We note that we could also consider non-regular hypergraphs in this framework as well, but since stashing on these graphs can easily be reduced to stashing on regular hypergraphs, we consider only regular hypergraphs here. We refer to 2-regular hypergraphs as standard graphs for clarity and convenience. When we say PROBLEM1 $\leq_P$ PROBLEM2, we mean that there is a polynomial time reduction from PROBLEM1 to PROBLEM2. In most proofs, we do not explicitly say the reductions can be done in polynomial time since their implementations are easily seen to be linear.

[2] We note that peeling does not completely solve the problem of assigning keys to buckets; for example, work on cuckoo hashing shows that one can do substantially better after peeling on random graphs to match additional keys to buckets. However, peeling provides a quick way to create an assignment, and for cuckoo hashing, peeling first maintains optimality in terms of the number of keys assigned. Because of this, understanding the limits of peeling in this context appears worthwhile.

3. 2-EDGE-STASH

For standard graphs, the 2-core is empty precisely when the graph has no cycles. It follows readily that for 2-EDGE-STASH the minimum number of edges that need to be removed equals the minimum number of edges that need to be removed so that the graph has no cycles; this well-known quantity is the *cyclomatic number* of the graph, $h(G) = |E| - |V| + (\text{\# connected components in } G)$. We note $h(G)$ is easily computed in polynomial time by starting with an empty graph, inserting the edges of G one at a time in any order, and setting aside any edge that forms a cycle, incrementing $h(G)$ accordingly. (Standard union-find algorithms can be used to test for cycles.)

As we show later, the corresponding vertex stash problem for standard graphs is actually NP-complete.

4. *k*-VERTEX-STASH is NP-complete

To start, it is clear that k-VERTEX-STASH is in NP for any d-regular hypergraph as the certificate is just the k-vertex-stash. Also, the standard NP-complete Vertex Cover problem is the degenerate problem 1-VERTEX-STASH. That is, since a graph has an empty 1-core if and only if it consists of a collection of isolated vertices, the smallest number of vertices to remove for 1-VERTEX-STASH is the size of the minimum vertex cover. This connection suggests a reduction from VERTEX COVER.

Definition 1. A vertex cover of G is a set of vertices V such that all edges in G are adjacent to at least one vertex in V. The NP-complete decision problem VERTEX-COVER(G, s) asks if G has a vertex cover of size at most s.

Theorem 1. *VERTEX-COVER $\leq_P$ k-VERTEX-STASH for $k \geq 2$.*

Proof. Given a hypergraph G we construct a hypergraph G' by adding vertices and edges to G such that V^* is a minimal vertex cover of G if and only if V^* is a minimal k-vertex-stash of G'. To create G', we replace every edge (u, v) by a subgraph $C_k(u, v)$ with the following properties.

1. $u, v \in C_k(u, v)$.
2. Each vertex $w \in C_k(u, v)$ has degree at least k.
3. If either u or v is removed, $C_k(u, v)$ has an empty k-core.