Can we create large k-cores by adding few edges?

Chitnis, Rajesh; Talmon, Nimrod

DOI:
10.1007/978-3-319-90530-3_8

Document Version
Peer reviewed version

Citation for published version (Harvard):
Chitnis, R & Talmon, N 2018, Can we create large *k*-cores by adding few edges? in VV Podolskii & FV Fomin (eds), *Computer Science - Theory and Applications : 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6–10, 2018, Proceedings.* Lecture Notes in Computer Science , vol. 10846, Springer Verlag, pp. 78-89, 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russian Federation, 6/06/18. https://doi.org/10.1007/978-3-319-90530-3_8

Can We Create Large k-Cores by Adding Few Edges?

Rajesh Chitnis[1⋆] and Nimrod Talmon[2⋆⋆]

[1] Department of Computer Science, University of Warwick, UK.
Email: rajeshchitnis@gmail.com
[2] Ben-Gurion University of the Negev. Email: talmonn@bgu.ac.il

Abstract. The notion of a k-core, defined by Seidman ['83], has turned out to be useful in analyzing network structures. The k-core of a given simple and undirected graph is the maximal induced subgraph such that each vertex in it has degree at least k. Hence, finding a k-core helps to identify a (core) community where each entity is related to at least k other entities. One can find the k-core of a given graph in polynomial time, by iteratively deleting each vertex of degree less than k. Unfortunately, this iterative *dropping out* of vertices can sometimes lead to unraveling of the entire network; e.g., Schelling ['78] considered the extreme example of a path with $k = 2$, where indeed the whole network unravels.

In order to avoid this unraveling, we would like to edit the network in order to maximize the size of its k-core. Formally, we introduce the Edge k-Core problem (EKC): *given a graph G, a budget b, and a goal p, can at most b edges be added to G to obtain a k-core containing at least p vertices?* First we show the following dichotomy: EKC is polytime solvable for $k \leq 2$ and NP-hard for $k \geq 3$. Then, we show that EKC is W[1]-hard even when parameterized by $b + k + p$. In searching for an FPT algorithm, we consider the parameter "treewidth", and design an FPT algorithm for EKC which runs in time $(k + \mathbf{tw})^{O(\mathbf{tw}+b)} \cdot \text{poly}(n)$, where $\mathbf{tw}$ is the treewidth of the input graph. Even though an extension of Courcelle's theorem [Arnborg et al. , *J. Algorithms* '91] can be used to show FPT for EKC parameterized by $\mathbf{tw} + k + b$, we obtain a much faster running time as compared to Courcelle's theorem (which needs a tower of exponents) by designing a dynamic programming algorithm which needs to take into account the fact that newly added edges might have endpoints in different bags which cross the separator.

1 Introduction

Graphs are very useful for modeling networks which describe relationships and interactions between sets of people or entities, in various disciplines, such as so-

⋆ Supported by ERC grant CoG 647557 "Small Summaries for Big Data". Part of this work was done when the author was at the Weizmann Institute of Science and supported by Israel Science Foundation grant #897/13.

⋆⋆ Part of this work was done when the author was at the Weizmann Institute of Science.

cial sciences [17], life sciences [10], medicine [20], etc. Usually, we assign a vertex to each entity, and there is an edge between two entities if they are related or affect each other in some way. Analyzing graph structure has found applications in several important real-world problems such as targeted advertising [24], fraud detection [18], missing link prediction [16], locating functional modules of interacting proteins [14], etc.

An important problem in analyzing the structure of big networks is to detect large communities of vertices that are "related" to one another. This problem has been widely considered in various sub-areas of computer science [12, 13, 19]. A reasonable and well-studied notion for a vertex to be "related" within a community is to have a *large* number of neighbors within the community. Bhawalkar et al. [4] considered the following model of *user engagement* within a network: there is a single product and each individual has two options of "engaged" or "drop out". We assume that all individuals are initially engaged, and there is some given threshold parameter k such that a person finds it worthwhile to remain engaged if that person has at least k engaged friends.

In this model of *user engagement* all individuals with less than k friends will drop out immediately. Unfortunately, this can propagate and even those individuals who initially had more than k friends in the network may end up dropping out. An extreme example of this was given by Schelling [22, page 214]: consider a path on n vertices and let $k = 2$. Note that, while $n-2$ vertices initially have degree two in the network, there will be a *cascade of iterated withdrawals* since each endpoint has degree one, thus it drops out and now its neighbor in the path has only one friend in the network and it drops out as well; eventually, the whole network drops out. Indeed, at the end of the iterated withdrawals process the remaining engaged individuals form a unique maximal induced subgraph whose minimum degree is at least k. This subgraph is called the *k-core* and is a well-known concept in the theory of networks; it was introduced by Seidman [23] and also been studied in various social sciences literature [8, 9]. The concept of k-core decompositions (where for each vertex v we find the max k such that v belongs to the k-core in G) has been used in the analysis and visualization of large scale networks [1–3].

A Game-Theoretic Model for k-Core: Consider the following game-theoretical model from [4]: each user in a graph $G = (V, E)$ pays a cost of k to remain engaged, and she receives a profit of 1 from every neighbor who is engaged. If an individual is not engaged, then she receives a payoff of zero. Hence, she remains engaged if she has non-negative payoff, i.e., she has at least k neighbors who are engaged. Then the k-core can be viewed as the unique maximal equilibrium in this setting. Assuming that all the players make decisions simultaneously the model can be viewed as a simultaneous-move game where each individual has two strategies viz. remaining engaged or dropping out. For every strategy profile $\delta \in \{0,1\}^{|V|}$ let $S_\delta = \{i \ : \ \delta_i = 1\}$ denote the set of players who remain engaged. We can easily characterize the set of pure Nash equilibria for this game: a strategy profile δ is a Nash equilibrium if and only if the following two conditions hold:

– No engaged player wants to drop out, i.e., minimum degree of the induced graph $G[S_\delta]$ is $\geq k$
– No player who has dropped out wants to become engaged, i.e., no $v \in V(G) \setminus S_\delta$ has $\geq k$ neighbors in S_δ

In general there can be many Nash equilibria. For example, if G itself has minimum degree $\geq k$ then $S_\delta = \emptyset$ and $S_\delta = V(G)$ are two equilibria (and there may be more). Owing to the fact that it is a maximal equilibrium, the k-core has the special property that it is beneficial to both parties: it maximizes the payoff of every user, while also maximizing the size of the network. Chwe [8, 9] and Sääskilahti [21] suggest that one can reasonably expect this maximal equilibrium even in real-life implementations of this game.

The Anchored k-Core problem (AKC): The unraveling described above in Schelling's example of a path might be highly undesirable if the goal is to keep as many people engaged as possible. One possibility of preventing this is by "buying" the two end-point players into being engaged. This ensures that the whole path remains engaged. Correspondingly, Bhawalkar et al. [4] formalized this notion and defined the Anchored k-Core problem (AKC). In AKC, they overcome the issue of unraveling by allowing some "anchors": these are vertices that remain engaged irrespective of their degree. This can be achieved by giving them extra incentives or discounts. The question in AKC is the following: given three integers b, k, and p, can one use at most b *anchors* and ensure that there is an anchored k-core of size at least p?

Besides defining the AKC problem, Bhawalkar et al. [4] showed that AKC is solvable in polynomial time for $k \leq 2$ but NP-hard for $k \geq 3$. Also it is NP-hard to approximate the approximate the size of the optimal k-core to within an $O(n^{1-\epsilon})$ factor[3] for any $\epsilon > 0$. From the viewpoint of parameterized complexity, they showed that for every fixed $k \geq 3$ the p-AKC problem is W[2]-hard with respect to b, and, on the positive side, they gave a polynomial-time algorithm for graphs of bounded treewidth. In a follow-up work, Chitnis et al. [7] showed that it remains NP-hard on planar graphs for all $k \geq 3$, even if the maximum degree of the graph is $k + 2$; that it becomes FPT on planar graphs (unlike on general graphs) parameterized by b for all $k \geq 7$; and, strengthening the intractability result of Bhawalkar et al. [4], they showed W[1]-hardness of it with respect to p (which is always greater than or equal to b).

The Edge k-Core problem (EKC): In this paper we consider an alternative way to maximize the size of the k-core. For example, in Schelling's example, instead of anchoring the two end-point players, one could also *add an edge* between these two vertices; this again ensures that the whole path remains engaged. While this changes the structure of the network, it has the desirable property of ensuring a "pure" k-core, i.e., where *each* vertex has degree at least k. Correspondingly, we ask the following question: can we add "few" edges to the given network and obtain a "large" k-core? Formally, the problem we study in this paper is as follows.

[3] That is, distinguishing whether the size of the optimal k-core is $O(b)$ or $\Omega(n)$

The Edge k-Core Problem (EKC)
Input : A simple, undirected graph $G = (V, E)$ and integers b, k, and p.
Question: Is there a set of vertices $H \subseteq V$ of size $\geq p$ such that there is a set $B \subseteq (\binom{V}{2} \setminus E)$ with $|B| \leq b$ and every $v \in H$ satisfies $\deg_{G'[H]}(v) \geq k$, where $G' = (V, E \cup B)$?

EKC arises naturally in several scenarios concerning network resiliency such as:

- **Peer-to-Peer (P2P) networks:** In P2P networks, users share common resources (bandwidth, disk storage, etc.). For any user to benefit from the network they should be connected to at least a certain threshold k number of other users. EKC then tells the parent company which connections shall be added between the users so that a large number of users can successfully use the P2P network.
- **Distributed networks:** Suppose there is an existing network of computers, connected by some topology. Over time the complexity of the tasks to be executed increases, and one may need more computers to perform the task in a distributed fashion. The Edge k-Core problem then guides us on how to edit the network (which connections to add between computers) assuming that we know the threshold k number of computers needed for any task.

Our Results. Besides introducing EKC, we first, in Section 2, describe a polynomial-time algorithm for $k \leq 2$ and show NP-hardness for $k \geq 3$, thus providing a complexity dichotomy. Then, in Section 3, we begin by showing that EKC is W[1]-hard parameterized by $b + p$ even when $k = 3$. This tells us that we need to consider more parameters if we seek fixed-parameter tractability. As a natural network parameter, we consider the treewidth **tw** and design a dynamic program to show that EKC is fixed-parameter tractable with respect to $\mathbf{tw} + k + b$. In our view, this is the most technical part of the paper.

Comparing Anchored k-Core and Edge k-Core. While AKC has been studied before [4, 7, 6], in this paper we introduce and study the EKC problem. Below we briefly show that these two problems are unrelated in the following sense: there are examples of graphs where for the same values of p and k we need very different number of anchored vertices or edge additions to achieve a k-core of size at least p.

- **Edge Additions > Anchored Vertices**: Let G be a disjoint union of two components G_1 and G_2, where $G_1 = K_{z_1}$ (i.e., a clique on z_1 vertices) and G_2 is a z_2-regular graph on n_2 vertices. Choose $z_1 \ll z_2 \ll n_2$. If $k = z_2$ and $p = z_1 + n_2$, then the number of vertices which need to be anchored for an *anchored* k-core of size p is $b_v = z_1$. However, to get a *pure* k-core of size p we need to add $z_2 - (z_1 - 1)$ edges on each vertex of G_1, and hence $b_e \geq \frac{z_1 \cdot (z_2 - z_1 + 1)}{2} \gg z_1 = b_v$. This example is asymptotically tight since adding $(k - \deg)$ new edges to a vertex of degree k simulates anchoring the vertex.

– **Edge Additions < Anchored Vertices**: We build a graph G as follows: let $G_1 = K_{2n}$, let G_2 be K_{2n} with a perfect matching removed, and add a matching of size $2n$ between G_1 and G_2. If $k = 2n$ and $p = 4n$, then the number of vertices which need to be anchored to obtain an *anchored* k-core of size p is $b_v = 2n$. However, to get a *pure* k-core of size p it is enough to add the n edges of the perfect matching which were removed from G_2. Hence $b_e = n < 2n = b_v$. This example is strictly tight since anchoring two endpoints of an edge simulates adding the edge.

2 Classical Complexity: Polytime algorithms and NP-hardness

In this section we first describe a polynomial-time algorithm which solves Edge k-Core whenever $k \leq 2$ (Theorem 1). Then, we show that this result is tight with respect to k, by showing that Edge k-Core is NP-hard whenever $k \geq 3$ (Theorem 2).

Theorem 1. *[⋆][4] Edge k-Core is polynomial-time solvable for $k \leq 2$.*

Theorem 2 (⋆). *Edge k-Core is NP-hard for $k \geq 3$.*

3 Parameterized Complexity: W[1]-hardness and FPT algorithms

In this section, we analyze EKC via the framework of parameterized complexity. EKC is para-NP-hard parameterized by k since Theorem 2 shows that EKC is NP-hard for $k = 3$. In fact, taking a closer look at the proof of Theorem 2, we observe the following.

Corollary 1. *EKC is NP-hard for $k = 3$, even on planar graphs of max degree 5.*

Now we show that EKC admits a simple XP algorithm parameterized by b.

Observation 3 *The EKC problem admits an XP algorithm parameterized by b.*

Proof. Since any graph on n vertices can have at most $\binom{n}{2}$ edges (recall that we do not allow parallel edges or self-loops), we can try all possible subsets of non-edges (which are not already present) to be added. This gives an $\binom{\binom{n}{2}}{b} = n^{O(b)}$ algorithm.

[4] Proofs of results marked with [⋆] are deferred to the full version of the paper due to lack of space.

Consider a yes-instance I of EKC for which $p \leq k$. Since the minimum degree of any subgraph containing at most k vertices is at most $k-1$, it follows that the size of the k-core created in the solution of I is at least $k+1$ (recall the definition of EKC, which asks for a k-core containing **at least** p vertices). Thus, we assume henceforth that instances of EKC satisfy $p > k$. Next, we show that Theorem 3 translates to an XP algorithm for EKC parameterized by p.

Proposition 1. *The EKC problem has an XP algorithm parameterized by p.*

Proof. We can convert any set of p vertices (assuming w.l.o.g. that $p > k$) into a k-core by adding at most $\binom{p}{2}$ edges. Hence, if $b \geq \binom{p}{2}$, then we can answer YES; otherwise, i.e., if $b < \binom{p}{2}$, then the $n^{O(b)}$ algorithm from Theorem 3 is also an $n^{O(p^2)}$ algorithm for the EKC problem. □

Next we show that if one wants to design an FPT algorithm for the EKC problem, then even combining the three parameters p, k, and b is not enough.

Theorem 4. *The* Edge k-Core *problem is W[1]-hard parameterized by $p+b$, for $k=3$.*

Proof. We reduce from the W[1]-hard Clique problem [11] which, given a graph G and an integer ℓ, asks for the existence of ℓ pairwise adjacent vertices in G. Consider an instance $(G=(V,E),\ell)$ of Clique where $V=(v^1,v^2,\ldots,v^n)$ and construct a new graph $G'=(V',E')$ as follows.

For each $1 \leq i \neq j \leq \ell$ make a copy G_{ij} of the vertex set V (do not add any edges). Each of these vertices is *black*. Let the vertex v^r in the copy G_{ij} be labeled v^r_{ij}. Add the following edges to G' (we use the notation $[n]=\{1,2,\ldots,n\}$):

- For each $1 \leq i \neq j \leq \ell$ and $r,s \in [n]$ we add an edge between v^r_{ij} and v^s_{ji} if and only if $v^r v^s \in E$. Subdivide each such edge twice by adding two new **green** vertices x^{rs}_{ij} and x^{sr}_{ji}. We refer to x^{rs}_{ij} as a **brother** of x^{sr}_{ji} and vice-versa.
- For each $i \in [\ell], r \in [n]$ add the cycle $v^r_{i1}-v^r_{i2}-\ldots v^r_{i,i-1}-v^r_{i,i+1}-\ldots-v^r_{i\ell}-v^r_{i1}$. Let us denote this cycle by C^r_i.

This completes the construction of G'. Let $k=3$, $b=\binom{\ell}{2}$ and $p=4b$. In the full version of the paper, we show the correctness of the reduction.

3.1 FPT Algorithm Parameterized by $\mathbf{tw}+k+b$

Next we sketch the proof of our main technical result, namely that EKC is fixed-parameter tractable for $\mathbf{tw}+k+b$. Notice that Theorem 4, which shows that EKC is W[1]-hard even for $k+p+b$, indeed motivates studying further parameters.

Let T be a tree and $B : V(T) \to 2^{V(G)}$. The pair (T,B) is called as a *valid tree decomposition* of an undirected graph G, if T is a tree in which every vertex $x \in V(T)$ has an assigned set of vertices $B_x \subseteq V(G)$ (called a bag) such that the following properties are satisfied:

– **(P1)**: $\bigcup_{x \in V(T)} B_x = V(G)$.
– **(P2)**: For each $u - v \in E(G)$, there exists an $x \in V(T)$ such that $u, v \in B_x$.
– **(P3)**: For each $v \in V(G)$, the set of vertices of T whose bags contain v induces a connected subtree of T.

The *width* of the tree decomposition (T, B) is $\max_{x \in V(T)} |B_x| - 1$. The treewidth of a graph G, usually denoted by $\mathbf{tw}(G)$, is the minimum width over all valid tree decompositions of G.

We will use a special type of tree decompositions called *nice* tree decompositions.

Definition 1. *A tree-decomposition (T, B) of G is said to be* nice *if T is a rooted binary tree such that each vertex $t \in T$ is one of the following four types:*

– ***Leaf Node**: t is a leaf in T and $B_t = \{v\}$ for some $v \in G$.*
– ***Introduce Node**: t has exactly one child t' and $B_t = B_{t'} \cup \{v\}$ for some $v \notin B_{t'}$.*
– ***Forget Node**: t has exactly one child t' and $B_t = B_{t'} \setminus \{v\}$ for some vertex $v \in B_{t'}$.*
– ***Join Node**: t has exactly two children t', t'' such that $B_{t'} = B_t = B_{t''}$.*

The advantage of nice tree-decompositions is that when writing a dynamic program we only need to handle four types of nodes. It is known [5, 15] that a general tree decomposition (T, B) (of treewidth $\mathbf{tw}$) can be converted, in linear time, into a *nice tree decomposition* (T', B') of the same width such that $|T'| = O(\mathbf{tw} \cdot n)$.

Our main result in this section, and what we believe is the most technically-involved result in this paper, is a dynamic programming based FPT algorithm for EKC parameterized by $\mathbf{tw} + k + b$. We remark here that the fixed-parameter tractability of EKC parameterized by $\mathbf{tw} + k + b$ can be shown to follow from Courcelle's theorem, albeit with much worse running time (i.e., tower of exponentials); further, we argue that our dynamic programming is interesting because the operation of adding edges is an "inter-bag" operation: given a bag which separates the graph into two parts, a new edge might have one endpoint in each part. Usually, FPT algorithms parameterized by treewidth are for "inter-bag" operations, where each recursive call in the dynamic program is confined to its subtree, while in our case the structure is more involved. In the interest of space, we provide here a description with some intuition for the proof of Theorem 5. The complete formal proof of correctness and the analysis of the running time is deferred to the full version.

Theorem 5. *The* Edge k-Core *problem can be solved in $(k + \mathbf{tw})^{O(\mathbf{tw}+b)} \cdot poly(n))$ time.*

Proof (Sketch). Given a nice tree decomposition of a given graph (notice that finding a tree decomposition, or at least an approximation of it, can be done in FPT time) we design a dynamic program for solving EKC on it. First we

fix an (arbitrary) ordering on the vertices of V, say $\phi = (v_1, v_2, \ldots, v_n)$. We will view the vertices in this order when we consider them in a bag of the tree-decomposition; to this end, for a node t in the tree decomposition which is a bag containing x vertices, let us arbitrarily order those x vertices (while fixing this ordering) and denote them by $[v_1, \ldots, v_x]$. For a node $t \in T$ let T_t denote the subtree of T which is rooted at t. Also let V_t denote the union of vertices in all bags of the nodes in the subtree of T rooted at t, i.e., $V_t = \cup_{t \in T_t} B_t$. For $i \in \mathbb{N}$ let $\mathbf{0}^i, \mathbf{1}^i$ denote the multiset which has i zeroes and i ones respectively. For $q \geq s$ let $\mathcal{H}(q,s) = \{\mathbf{z} \in \{0,1\}^q \ : \ \mathbf{z} \text{ has Hamming weight exactly s}\}$. Similarly, $\mathcal{H}^*(q,s) = \{\mathbf{z} \in \{0,1,\ldots,k\}^q \ : \ \mathbf{z} \text{ has at most s non-zero entries}\}$. Finally, for $\mathbf{z} \in \mathcal{H}(q,s)$ we define the set $\mathcal{H}^*_z(q,s) = \{\mathbf{y} \in \mathcal{H}^*(q,s) \ : \ \mathbf{y}[i] \neq 0 \Rightarrow \mathbf{z}[i] \neq 0\}$.

For a node $t \in T$ in the tree decomposition, we define the following boolean quantity

$$\texttt{BOOL}[t, b_{\texttt{in}}, b_{\texttt{out}}, p_{\texttt{in}}, p_{\texttt{out}}, \mathbf{y}, \mathbf{z}, q_0, Q]$$

for each choice of

- $b_{\texttt{in}}, b_{\texttt{out}}, p_{\texttt{in}}, p_{\texttt{out}} \geq 0$
- $b_{\texttt{in}} + b_{\texttt{out}} \leq b$
- $p_{\texttt{in}} \leq |B_t|$ and $p_{\texttt{out}} \leq |V_t \setminus B_t|$
- $\mathbf{y} \in \mathcal{H}(|B_t|, p_{\texttt{in}})$ and $\mathbf{z} \in \mathcal{H}^*_y(|B_t|, p_{\texttt{in}})$
- $0 \leq q_0 \leq p_{\texttt{out}}$ (actually, $p_{\texttt{out}} - b \leq q_0 \leq |Q|$)
- $Q = \{q_1, q_2, \ldots, q_{p_{\texttt{out}} - q_0}\}$ is a (multi)set of size $p_{\texttt{out}} - q_0$ such that each element in Q is positive and at most k, i.e., $1 \leq q_i \leq k$ for each $i \in [p_{\texttt{out}} - q_0]$

We set $\texttt{BOOL}[t, b_{\texttt{in}}, b_{\texttt{out}}, p_{\texttt{in}}, p_{\texttt{out}}, \mathbf{y}, \mathbf{z}, q_0, Q] = 1$ if and only if there exist sets $H_t \subseteq V_t$ and $E_t \subseteq H_t \times H_t$ such that the following conditions hold:

- $|H_t \cap B_t| = p_{\texttt{in}}$
- $|H_t \cap (V_t \setminus B_t)| = p_{\texttt{out}}$
- Number of edges of E_t which have both endpoints in B_t is $b_{\texttt{in}}$
- Number of edges of E_t which have at most one endpoint in B_t is $b_{\texttt{out}}$
- $|E_t| \leq b_{in} + b_{out}$
- $\mathbf{y}[v] = 1$ if and only if $v \in B_t \cap H_t$
- If $v \in B_t \cap H_t$ then the degree of v in the graph $G^*[H_t] = G[H_t] \cup E_t$ is $\geq \mathbf{z}[v]$
- There is a bijection $\phi_t : ((V_t \setminus B_t) \cap H_t) \to (Q \cup \mathbf{0}^{q_0})$ such that for every $w \in (V_t \setminus B_t) \cap H_t$ we have $\deg_{G^*[H_t]}(w) + \phi_t(w) \geq k$

We say that (H_t, E_t, ϕ_t) is a **witness** for $\texttt{BOOL}[t, b_{\texttt{in}}, b_{\texttt{out}}, p_{\texttt{in}}, p_{\texttt{out}}, \mathbf{y}, \mathbf{z}, q_0, Q] = 1$.

Intuition: Instead of solving the Edge k-Core problem in recursion, we design a dynamic program which solves a more general problem. Specifically, this general problem is such that (1) we specify more concretely how this given budget is to be used and which structure the k-core shall have; and (2) we allow some "help" from the "outside world". Let us mention that, for (1), we specify how

the budget b shall be split into b_{in} (budget to be used solely in the bag vertices) and b_{out} (budget to be used not solely in the bag vertices); how the p vertices of the k-core shall be split between p_{in} (k-core vertices in the bag vertices) and p_{out} (k-core vertices not in the bag vertices); where we even specify, by the 1-entries of the vector $\mathbf{y}$, exactly which vertices of the bag shall be k-core vertices. Now, recall that in the Edge k-Core problem, each vertex in the k-core must have degree at least k; for (2), we allow some "help" from the "outside world", by relaxing this "at least k" requirement for some of the vertices; specifically, we specify the needed degrees for the vertices of the k-core in the bag vertices (those p_{in} vertices whose corresponding y values are 1), since we require for them to have only degrees as specified by the $\mathbf{z}$ vector; and, for the vertices of the k-core which are not in the bag vertices, we use two multisets Q and $\mathbf{0}^{q_0}$ (which are together of size p_{out}). We view this as some "degree help" that we allow those vertices to use (in order to make their degree $\geq k$): the exact way by which we specify the amount of "degree help" that the vertices of the k-core could use is defined by the bijection ϕ. The multiset Q corresponds to those vertices which actually need some "degree help" (and we maintain all such numbers), while the number q_0 corresponds to the number of vertices of $H_t \cap (V_t \setminus B_t)$ which do not need any help at all.

The crucial idea is that although $p_{\texttt{out}}$ can be as large as n, we have that $|Q| \leq b$ since we only have b edges in the budget to help. (In fact, even $\sum_{x \in Q} x \leq b$ holds.)

Let r be the root of T. Next, we will show how to recursively compute the values of the boolean quantity BOOL; for now, let us mention that we will decide that the given instance (G, b, k, p) of Edge k-Core is a yes-instance if and only if the following holds:

$$\bigvee_{\substack{b_{\texttt{in}}+b_{\texttt{out}} \leq b \\ 0 \leq p_{\texttt{in}} \leq |B_r| \\ 0 \leq p_{\texttt{out}} \leq |V_r \setminus B_r| \\ p_{\texttt{in}}+p_{\texttt{out}} \geq p \\ \mathbf{y} \in \mathcal{H}(|B_r|, p_{\texttt{in}})}} \texttt{BOOL}[r, b_{\texttt{in}}, b_{\texttt{out}}, p_{\texttt{in}}, p_{\texttt{out}}, \mathbf{y}, \mathbf{z} = k^{|B_r|}, q_0 = p_{\texttt{out}}, Q = \emptyset] = 1$$

Below we briefly give some intuition on how to recursively compute the values of BOOL for each type of node in the nice tree decomposition. We defer the formal recurrence, proof of correctness, and analysis of the running time to the full version.

3.1.1 Leaf node

Intuition: For leaf nodes, there are no further recursive calls; thus, it is enough to check the "sanity" of the given values.

3.1.2 Forget node

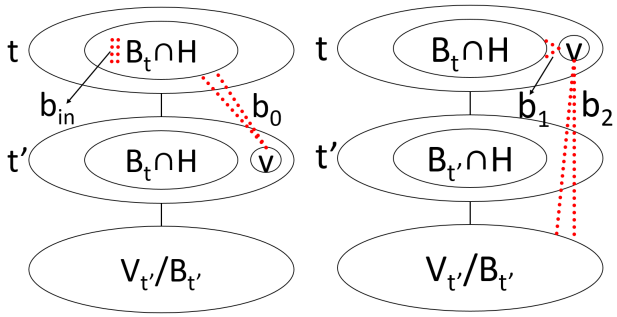

Fig. 1. Recursion on a forget node (left) and on an introduce node (right).

Intuition: Refer to Figure 1. There are two possibilities for forget node, namely whether the forgotten vertex v is part of the k-core or not. If it is not, then we can call the child with almost exactly the same values; if it is in the k-core, then we guess the exact connections that the forgotten node will have to other vertices in the bag. Given those guesses, we can issue a recursive call almost without worrying about the forgotten node; notice that we guess whether or not v receives non-zero "help" from the outside, since, in the child, v is a bag vertex, and thus does not have a corresponding Q-value or is counted in q_0.

3.1.3 Introduce node

Intuition: Refer to Figure 1. Let t' be the child of t such that $B_t = B_{t'} \cup \{v\}$. There are two cases to consider, namely whether v is in the k-core or not. If v is not in the k-core then we can safely issue a recursive call to the child t'. Otherwise, we can fully guess the "new" connections between v to the other vertices in the bag B_t, and then call the child with different z-values, since their degrees will be increased by v. By the definition of a tree-decomposition there cannot be any edges already present between v and any vertex of $V_{t'} \setminus B_{t'}$. However, while staying within FPT time we cannot guess the exact set of vertices from $V_{t'} \setminus B_{t'}$ which get a "help" edge from v. Instead we just guess the number of such edges, and issue the recursive call for t' with the appropriate changes in some Q-values.

3.1.4 Join node

Intuition: Refer to Figure 2. First we can guess how the $b_{\mathtt{in}}$ edges that will be introduced in the bag of t so we can then call t and t' with $b_{\mathtt{in}=0}$. Then we guess the partition of $b_{\mathtt{out}}$ into three parts: two parts correspond to the $b_{\mathtt{out}}$ edges for t', t'' respectively and the third part corresponds to edges between $V_{t'} \setminus B_{t'}$ and $V_{t''} \setminus B_{t''}$. Note that by the properties of tree decompositions, it follows that there are no edges already present between $V_{t'} \setminus B_{t'}$ and $V_{t''} \setminus B_{t''}$.

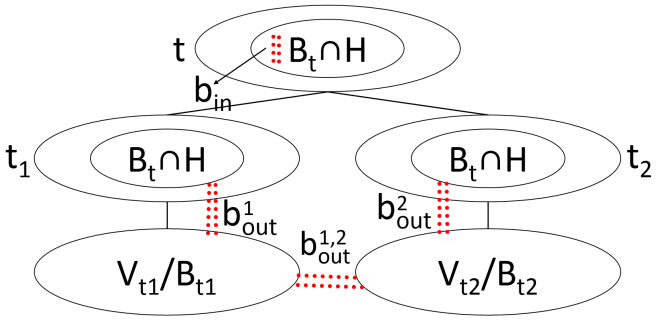

Fig. 2. Recursion on a join node.

4 Conclusions and Future Directions

In this paper, we introduced the Edge k-Core problem (EKC), where the goal is to create a "large" k-core by adding only "few" edges, and provided several hardness and algorithmic results. Specifically, we showed that EKC is polynomial-time solvable for $k \leq 2$ but NP-hard for $k \geq 3$; further, we showed that EKC is W[1]-hard for $k+p+b$, but fixed-parameter tractable for $\mathbf{tw}+k+b$.

For future research, one might look at EKC for directed graphs: similar work was done by Chitnis et al. [6] for the AKC problem. Another direction is to study the (in)approximability of EKC. Finally, one can consider a version which combines AKC with EKC: in it, a "large" anchored k-core would be created by anchoring at most b_v vertices and adding at most b_e edges.

Acknowledgments

The authors would like to thank Fedor Fomin, Petr Golovach, and Bart M.P. Jansen for helpful discussions.

References

1. J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NIPS 2005*, pages 41–50.
2. V. Batagelj, A. Mrvar, and M. Zaversnik. Partitioning approach to visualization of large graphs. In *Graph Drawing, 7th International Symposium, GD'99, Stirín Castle, Czech Republic, September 1999, Proceedings*, pages 90–97, 1999.
3. M. Baur, U. Brandes, M. Gaertler, and D. Wagner. Drawing the AS graph in 2.5 dimensions. In *Graph Drawing, 12th International Symposium, GD 2004, New York, NY, USA, September 29 - October 2, 2004,*, pages 43–48, 2004.
4. K. Bhawalkar, J. M. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing Unraveling in Social Networks: The Anchored k-Core Problem. *SIAM J. Discrete Math.*, 29(3):1452–1475, 2015.

5. H. L. Bodlaender and A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.
6. R. Chitnis, F. V. Fomin, and P. A. Golovach. Parameterized complexity of the anchored k-core problem for directed graphs. *Inf. Comput.*, 247:11–22, 2016.
7. R. H. Chitnis, F. V. Fomin, and P. A. Golovach. Preventing unraveling in social networks gets harder. In *Proceedings of AAAI '13*, 2013.
8. M. Chwe. Structure and Strategy in Collective Action 1. *American Journal of Sociology*, 105(1):128–156, 1999.
9. M. Chwe. Communication and Coordination in Social Networks. *The Review of Economic Studies*, 67(1):1–16, 2000.
10. Z. Dezső and A. Barabási. Halting Viruses in Scale-free Networks. *Physical Review E*, 65(5):055103, 2002.
11. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
12. N. Du, B. Wu, X. Pei, B. Wang, and L. Xu. Community detection in large-scale social networks. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 16–25. ACM, 2007.
13. S. Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.
14. T. Gutiérrez-Bunster, U. Stege, A. Thomo, and J. Taylor. How do biological networks differ from social networks? (an experimental study). In *ASONAM*, pages 744–751, 2014.
15. T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
16. N. Korovaiko and A. Thomo. Trust prediction from user-item ratings. *Social Network Analysis and Mining*, 3(3):749–759, 2013.
17. R. T. Mikolajczyk and M. Kretzschmar. Collecting social contact data in the context of disease transmission: Prospective and retrospective study designs. *Social Networks*, 30(2):127 – 135, 2008.
18. S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*, pages 201–210. ACM, 2007.
19. S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos. Community detection in social media. *Data Mining & Knowledge Discovery*, 24(3):515–554, 2012.
20. R. Pastor-Satorras and A. Vespignani. Epidemic Spreading in Scale-free Networks. *Physical review letters*, 86(14):3200–3203, 2001.
21. P. Sääskilahti. Monopoly Pricing of Social Goods. *Technical Report. University Librray of Munich*, 2007.
22. T. Schelling. *Micromotives and Macrobehavior*. WW Norton, 1978.
23. S. Seidman. Network Structure and Minimum Degree. *Social networks*, 5(3):269–287, 1983.
24. W.-S. Yang and J.-B. Dia. Discovering cohesive subgroups from social networks for targeted advertising. *Expert Systems with Applications*, 34(3):2029–2038, 2008.