Skeleton-Based Clustering by Quasi-Threshold Editing

Skeleton-Based Clustering by Quasi-Threshold Editing

Ulrik Brandes[1], Michael Hamann[2(✉)], Luise Häuser[2], and Dorothea Wagner[2]

[1] ETH Zürich, Zürich, Switzerland
ubrandes@ethz.ch
[2] Karlsruhe Institute of Technology, Karlsruhe, Germany
michael@content-space.de, ufziw@student.kit.edu, dorothea.wagner@kit.edu

Abstract. We consider the problem of transforming a given graph into a quasi-threshold graph using a minimum number of edge additions and deletions. Building on the previously proposed heuristic Quasi-Threshold Mover (QTM), we present improvements both in terms of running time and quality. We propose a novel, linear-time algorithm that solves the inclusion-minimal variant of this problem, i.e., a set of edge edits such that no subset of them also transforms the given graph into a quasi-threshold graph. In an extensive experimental evaluation, we apply these algorithms to a large set of graphs from different applications and find that they lead QTM to find solutions with fewer edits. Although the inclusion-minimal algorithm needs significantly more edits on its own, it outperforms the initialization heuristic previously proposed for QTM.

Keywords: Quasi-threshold graph · Trivially perfect graph · Graph editing · Graph clustering · Community detection

1 Introduction

We consider the problem of clustering a graph by partitioning its nodes. Especially in the context of social networks, this problem is often referred to as community detection. The approach taken here is to view community detection as a graph modification problem. Specifically, Nastos and Gao [25] proposed to edit a given graph into a quasi-threshold graph and use its connected components to determine the clustering.

A quasi-threshold graph, also known as trivially perfect graph, is the transitive closure of a rooted forest [33], which can in turn be considered a skeleton of the graph. Figure 1 shows an example, and we provide a more detailed motivation for this particular approach in the next section.

As minimizing the number of edits is $\mathcal{NP}$-hard [25], the Quasi-Threshold Mover (QTM) heuristic [4 SPP] starts from some rooted forest on the nodes of the input graph and moves nodes within and between trees to reduce the edit distance between the input graph and the transitive closure of the forest.

Several improvements to QTM are proposed in this chapter. We reduce the running time of one round of node moves to linear and show that the edits incident to a single node can be minimized using an additional path sorting step. This ultimately

H. Bast et al. (Eds.): Algorithms for Big Data, LNCS 13201, pp. 134–151, 2022.
https://doi.org/10.1007/978-3-031-21534-6_7

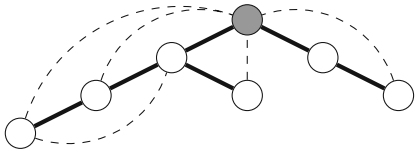

Fig. 1. Example quasi-threshold graph. The skeleton is denoted by thick edges, its transitive closure is dashed, the root is the gray node. (Color figure online)

leads to a linear-time algorithm for inclusion-minimal sets of edits. To also find smaller solutions, we propose a randomization of local moves. From an extensive experimental evaluation on empirical graphs we conclude that our modifications yields substantial improvements over the original QTM algorithm in terms of the size of the edit set.

2 Preliminaries

We consider simple undirected graphs $G = (V, E)$ consisting of $n := |V|$ nodes V that are connected by a set of $m := |E|$ edges $E \subset \binom{V}{2}$, i.e., without self-loops or multi-edges. By $N(u)$ we denote the set of neighbors of $u \in V$ and $\deg(u) := |N(u)|$ its degree. Further, let $N[u] := N(u) \cup \{u\}$ be the closed neighborhood of u. The subgraph induced by a set of nodes $X \subset V$ is denoted $G[X]$. With K_n, P_n, and C_n we denote the complete graph, path, and cycle on n nodes, respectively. These will be important as induced subgraphs, and we write, say, kK_n for k copies of K_n.

Quasi-threshold graphs are graphs that contain neither a P_4 nor a C_4 as node-induced subgraphs [34]. This is equivalent to an inductive construction in which the base case is a single node and there are two construction operators: either a universal node (adjacent to all previous nodes) is added, or the disjoint union of two quasi-threshold graphs is formed. The inductive construction of a quasi-threshold graph immediately gives rise to its skeleton forest referred to in the introduction.

2.1 Motivation

Many tasks in network analysis can be understood as first establishing an ideal, and then recovering that ideal from an empirical situation or at least determining a degree to which that ideal is met.

Take the most elementary notion, network density, as an example. The two idealized situations, polar opposites of one another, are graphs nK_1 of isolated nodes and cliques K_n[1]. The number of edges in a graph is a straightforward measure of distance from the ideal case of isolated nodes on an absolute scale of measurement. Since the number $\binom{n}{2}$ of edges in a clique varies with the number nodes n, *density* is often defined as the relative number $m/\binom{n}{2}$ of edges.

[1] It should not be lost on the reader that both names have social connotations [22].

A formulation of community detection using the same kind of reasoning can be developed as follows. Motivations for the vast majority of community detection methods generally state that the intention is to partition a graph into relatively dense subgraphs that are sparsely connected between them [17,28]. The idealized situation, with an undisputed partition into communities, is a *cluster graph* defined as disjoint unions of cliques or, equivalently, P_3-free graph. Each connected component is a clique, and these cliques are isolated from each other.

How far is a given graph from a cluster graph, and where are its communities? On an absolute scale, distance to the ideal situation is measured by counting the number of edges that need to be added or deleted to complete cliques and make those cliques independent. In *cluster editing*, a cluster graph of minimum edit distance is sought, and its cliques induce a clustering of the original graph. The normalized number of edges that do not have to be edited is known as *performance* [13], and cluster editing is a special case of *correlation clustering* [1]. Numerous other clustering approaches are based on objective functions that normalize the difference between a graph and the cluster graph ideal by taking additional factors such as the number of clusters, size of clusters, degree in clusters, etc. into account.

Like cluster graphs, quasi-threshold graphs represent an idealized situation, which we can think of as intersecting communities. To see this, we take two additional steps.

We start from *split graphs*, which are defined as those graphs that have a partition $V = C \uplus P$ into a clique $G[C]$ and an independent set $G[P]$, or, equivalently, $(2K_2, C_4, C_5)$-free graphs. They represent the ideal case of a core-periphery structure [3], and are characterized by their degrees: if $n > d_1 \geq \cdots \geq d_n \geq 0$ is the degree sequence of a graph, then it is a split graph if and only if the kth Erdős-Gallai inequality $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1}^{n} \min\{k, d_j\}$ is actually an equality for the *corrected Durfee number* $h = \max\{k : d_k \geq k-1\}$. In this case, h nodes of highest degree induce the clique and the others an independent set. The minimum number of edges that need to be edited to turn a graph into a split graph, known as its *splittance* [19], is half the difference between the two sides of the defining inequality, $\frac{1}{2}(\sum_{j=h+1}^{n} d_j - \sum_{i=1}^{h} d_i)$. These edits can be chosen so that h nodes of highest degree induce the clique and the remaining nodes the independent set. The computationally easy problem of split editing becomes intractable, for instance, if adapted for density [5] instead of edge numbers or for multiple cores [6].

By distinguishing a core from a periphery, split graphs also distinguish nodes that are central (as members of the core) from others that are not (as members of the periphery). Every node in the periphery is adjacent only to nodes in the core, and every node in the core is adjacent to all other core nodes. Hence, the neighborhood of any periphery node $u \in P$ is a subset of the closed neighborhood of any core node $v \in C$, $N(u) \subseteq N[v]$. This binary classification can be refined by comparing all pairs of nodes according to this neighborhood inclusion property, known as the vicinal preorder [16]. Schoch and Brandes [30] have shown retrospectively that this is actually the common ground of standard notions of centrality.

Graphs characterized by a total vicinal preorder are called *threshold* or *nested graphs* [23,24] and therefore the ideal structure of an undisputed ranking of nodes by centrality. Threshold editing is intractable, even if the input is a split graph [14], and for

a number of reasons centrality has been defined via an abundance of indices rather than the node ranking in a closest threshold graph.

Threshold graphs are the $(2K_2, C_4, P_4)$-free graphs and therefore a subclass of quasi-threshold graphs. They can be constructed by adding one node at a time, either as a universal or isolated node, so that they have a skeleton that is a caterpillar. Each connected component of a quasi-threshold graph, in turn, can be seen as a group of nested graphs that intersect at their cores but may branch out into different peripheries.

We have thus motivated quasi-threshold graphs as idealized structures of (disjoint groups of) intersecting communities. Quasi-threshold editing yields a partition into communities and in addition for each of them a centralized nesting structure represented by their skeleton tree.

2.2 Related Work

Quasi-threshold graphs can be recognized in linear time [8,34,4 SPP]. While the first algorithm [34] computes a skeleton forest if G is a quasi-threshold graph, the other [8, 4 SPP] additionally computes a forbidden subgraph if G is not.

As mentioned, quasi-threshold editing is $\mathcal{NP}$-hard [25]. Due to its characterization via a finite set of finite forbidden subgraphs, it is fixed-parameter tractable in the number of edits k [7]. In combination with the certifying recognition in linear time, this yields a simple $\mathcal{O}(6^k \times (n+m))$ time algorithm. For the related problem of quasi-threshold deletion, where edges may be deleted but not added, improved branching rules have been proposed, reducing the running time from $\mathcal{O}(4^k \times (n+m))$ to $\mathcal{O}(2.42^k \times (n+m))$ [21]. Further, ordered enumeration of solutions is also possible with FPT delay [10]. A polynomial kernel of $\mathcal{O}(k^7)$ nodes has been introduced by Drange and Pilipczuk [15], who also show that the problem cannot be solved in time $2^{ok} \times n^{\mathcal{O}(1)}$ unless the Exponential Time Hypothesis fails.

The first editing heuristic has been proposed by Nastos and Gao [25]. With Quasi-Threshold Mover [4 SPP], the first editing heuristic with a running time close to linear has been proposed. Recently, a study on techniques for computing exact solutions has been published [18 SPP].

For the superclass of cographs, or P_4-free graphs [9], the problem of inclusion-minimal editing has recently been considered [11]. Instead of asking for a set of edge edits of minimum cardinality, it asks for a set of edge edits such that no proper subset yields a cograph. While cograph editing is also $\mathcal{NP}$-hard [20], inclusion-minimal cograph editing can be solved in linear time [11].

3 Quasi-Threshold Mover (QTM)

The Quasi-Threshold Mover algorithm, short QTM, iteratively improves the skeleton forest to heuristically minimize the number of induced edits. It starts with a given skeleton forest, this may be the trivial skeleton where every node is a root which implies that all edges are deleted. In each round, it iterates over all nodes u in a random order and possibly moves u to a new position in the forest if this decreases the number of induced edits. For this, it considers every node $v \in V \setminus \{u\}$ as a parent for u. Further, a subset of

the children of the new parent v may be adopted, i.e., moved below u. In the induced quasi-threshold graph, u is then connected to v and all its ancestors as well as to all adopted children and their descendants. Every neighbor x of u in this set of nodes saves deleting the edges $\{u,x\}$ but every non-neighbor y of u implies inserting the edge $\{u,y\}$. Therefore, we select the parent v and the children to adopt such that the number of u-neighbors minus non-neighbors is maximized. Given a potential parent v, we always adopt children whose subtrees contain more neighbors than non-neighbors of u. We call those children *close children*. Using a DFS, we could determine for every node how many neighbors and non-neighbors are above/below that node and thus allowing the selection of the best parent and which children are close. However, this gives a quadratic running time per round. Instead, QTM starts limited local searches starting from the neighbors of u. They only visit one or two non-neighbors per neighbor. The idea is that whenever a subtree contains more neighbors than non-neighbors, it will be fully visited. Thus, the algorithm is able to determine all close children. Similarly, the best parent is determined by propagating information upwards in the skeleton. As QTM uses a priority queue to manage nodes during this bottom-up search, the running time per round is $\mathcal{O}(n+m\log\Delta)$, where Δ is the maximum degree.

In the following, we present several novel improvements for QTM. In Sect. 3.1, we show how to reduce the running time per round to linear in the number of nodes and edges. Further, in Sect. 3.2, we present an additional path sorting step that modifies the skeleton forest before every local move of a node u and yields a move that is optimal with respect to the edits incident to u. This local optimality directly gives us an inclusion-minimal algorithm as we show in Sect. 3.3. The last improvement is randomization, in Sect. 3.4, we show how we can select uniformly at random among all possible sets of edits incident to the moved node u.

3.1 Linear Running Time

To realize its bottom-up search, QTM needs to process nodes ordered by depth in the forest. While it is straightforward to use a bucket per level in the forest, this has the problem that this yields a running time linear in the depth of the deepest neighbor of u. It turns out, though, that we do not need to consider deep neighbors. More precisely, we show that we can ignore neighbors at a depth of more than $2\times\deg(u)$. Consider a node $v\in N(u)$ with depth larger than $2\times\deg(u)$. If u is connected to v in the edited graph, this implies that u is also connected to all at least $2\times\deg(u)$ ancestors of u. Among these, there can be at most the remaining $\deg(u)-1$ neighbors and thus at least $\deg(u)+1$ non-neighbors. Thus, this implies $\deg(u)+1$ edge insertions. Making u a root in the forest, i.e., deleting all edges incident to u, causes just $\deg(u)$ edits and is thus better. Therefore, we can ignore neighbors with depth larger than $2\times\deg(u)$. We can thus use a bucket per depth of the remaining neighbors which eliminates the log-factor of the running time.

3.2 Sorting Simple Paths

QTM minimizes edits with respect to the choice of a parent and adopted children of that parent. Here we show that an additional sorting step minimizes the edits incident to u

in the edited graph independently of the chosen skeleton forest. For this, we consider *simple paths*, which we define as a maximal path in the skeleton forest in which each node has exactly one child except for the lowest node. Every node is thus part of exactly one simple path, which may only consist of the node itself. A crucial observation is that reordering nodes in simple paths is the only way the skeleton forest can be modified without affecting the induced quasi-threshold graph.

Lemma 1. *Let G be a graph and T a corresponding skeleton forest. It holds that $N[u] = N[v]$ if and only if u and v are on the same simple path.*

Proof. If $N[u] = N[v]$, then u and v are on the same simple path:

Assume otherwise, i.e., that u and v are not on the same simple path in T. Consider the path P_{uv} between u and v in T. As it is not simple, it contains a node that is not its lowest node and has at least a child x that is not on P_{uv}. As x is not on P_{uv}, either $\{u,x\} \in E$ and $\{v,x\} \notin E$ or vice-versa, depending on whether u is an ancestor of v or v an ancestor of u. This is a contradiction to $N[u] = N[v]$, thus u and v must be on the same simple path.

If u and v are on the same simple path, $N[u] = N[v]$:

An edge $\{u,v\}$ exists if and only if u and v are in an ancestor-descendant relationship in the skeleton T. Consider a node u. All ancestors/descendants of u apart from its simple path are also ancestors/descendants of all other nodes in its simple path. Further, the nodes in its simple path form a clique. Therefore, $N[u] = N[v]$. □

Lemma 2. *Let T, T' be two different skeletons that induce the same quasi-threshold graph G. Then the simple paths of u in T and T' consist of the same nodes.*

Proof. Assume otherwise, i.e., that the simple paths of u in T and T' differ. Then there is a node x that is on the simple path of u in T but not in T' (or vice-versa, but assume w.l.o.g. that it is in T). As x and u are on the same simple path in T, $N[u] = N[x]$ by Lemma 1. Lemma 1 also implies that u and x must be on the same simple path in T', which is a contradiction to the existence of x and thus our assumption. Thus, the simple paths of u must consist of the same nodes in T and T'. □

Lemma 3. *Let T, T' be two skeletons that induce the same quasi-threshold graph G. Then the only difference between T and T' is the reordering of simple paths.*

Proof. Assume otherwise, i.e., that there were two skeletons T, T' that imply the same quasi-threshold graph G but differ more than just reordering of simple paths. A forest is uniquely determined by specifying the set of ancestors of every node. Thus there must be a node u such that the ancestors of u in T are different from the ancestors in T'. As a consequence, there is a node v that is an ancestor of u in T or T', but not in both. Assume w.l.o.g. that v is an ancestor of u in T. Due to T, $\{u,v\} \in E$. As $\{u,v\} \in E$ if and only if v is an ancestor of u or v is a descendant of u, v must be a descendant of u in T'. As u is an ancestor of v in T, $N[u] \supseteq N[v]$. As v is an ancestor of u in T', $N[v] \supseteq N[u]$ and thus $N[u] = N[v]$. Due to Lemma 1, this implies that u and v are together on a simple path in both T and T'.

By Lemma 2, the simple path of u must consist of the same nodes in T and T'. Therefore, we can replace the simple path in T by the simple path in T' without altering the resulting graph, and then search for a new pair u, v as described above. This reordering of the simple path does not change any other simple path. Therefore, if we apply this procedure repeatedly, it cannot find the same nodes again. Thus, this procedure terminates after at most n steps. As every step just reorders a simple path, the only difference between T and T' was reordering of simple paths. □

The main idea of path sorting is, before moving a node u, to move all its neighbors to the top of their respective simple paths. Since it might unify simple paths and thus enable reordering, u is first removed from the graph. This reordering makes it possible to choose the lowest neighbor of a simple path as parent without needing to insert edges to other non-neighbors in it. Note that the order in simple paths does not play a role when adopting a node c as a child because all nodes in its path become neighbors of u anyway. We show that this minimizes the number of edits incident to u by considering an optimal set of edits and its skeleton forest and showing that our forest with reordered simple paths does not yield more edits.

Lemma 4. *Consider a graph $G = (V, E)$, a node $u \in V$ and a skeleton forest T. Applying QTM to u on T^- which is T with u removed and simple paths reordered such that neighbors of u are at the top of their simple paths minimizes the number of edits incident to u.*

Proof. Let Q be the quasi-threshold graph with minimum edits incident to u and T_Q a skeleton forest of Q. Let T_Q^- be T_Q without u, children of u attached to u's parent. This keeps all ancestor-descendant-relationships between all nodes except u and thus all remaining edges. The reverse of this operation is exactly what QTM does: choosing a parent and potentially adopting some of its children. Thus, QTM can find an optimal set of edits incident to u in T_Q^-. Since, by Lemma 3, T^- and T_Q^- differ only in the order of nodes on simple paths, we show that the orderings of T^- and T_Q^- are equally good.

Consider the parent p and children C of u in T_Q. If p is the lower end of its simple path in T_Q^-, we obtain the same ancestor from the lowest node of p's simple path in T^-. Similarly, for an adopted child $c \in C$, adoption of the highest node of c's simple path in T^- yields the same descendants. If p is not the lower end of its simple path in T_Q^-, we distinguish two cases: u adopted p's only child or u is a leaf node in T_Q. In the first case, neither the position in p's simple path nor its node order matters as any position and node order gives the same neighbors and thus edits. If u is a leaf node in T_Q and p is not the lower end of its simple path, the node order matters as u is only connected to p and p's ancestors but not the nodes below p on p's simple path P_p in T_Q^-. By Lemma 3, P_p also exists in T_Q^-. Every non-neighbor of u among p and its ancestors in P_p causes an edge insertion while every neighbor of u below p causes an edge deletion. By moving all neighbors of u to the top of P_p and choosing the lowest neighbor of u on P_p as parent, we do not get any edits incident to nodes of P_p and thus minimize the edits among all possible orderings of P_p. This shows that QTM finds a parent and children to adopt on T^- that minimize the number of edits incident to u. □

What remains to show is that maintaining and sorting all simple paths does not increase the asymptotic running time of QTM. Simple paths are maintained explicitly

in a dynamic array, every node stores its simple path and position in it. This allows us to swap neighbors of the node to move u in constant time to the position of the first non-neighbor in its simple path, we also store this position. Moving nodes can cause simple paths to be split or joined. We store simple paths ordered from lowest to highest node. Whenever simple paths are split or merged, u is adjacent in the edited graph to the upper part of the path either before or after the move. In a split, we remove the upper part of the path from its end. In a merge, we add the nodes of the upper path to the lower path. Both operations are thus linear in the number of neighbors of u before or after the move. The running time analysis of QTM already accounts for running time linear in the number of neighbors of u in the edited graph both before and after the move. Thus, path sorting does not increase the asymptotic running time of QTM.

3.3 Inclusion-Minimal Editing

With the local moving routine of QTM, we can incrementally insert the nodes of a graph G into an initially empty graph. Due to Lemma 4, this minimizes the number of edits in each step. Overall, this yields an inclusion-minimal editing of G, as it has also been shown, e.g., for interval graphs [26]. The basic idea is that if there was a set of superfluous edits, these edits could have been omitted already at the steps where they were introduced, violating the local minimality guaranteed by Lemma 4.

This inclusion-minimal editing algorithm can also be considered a one-pass streaming algorithm. To add a node, we need the skeleton of the already seen nodes, which can be stored in $\mathcal{O}(\log(n))$ bits per node. We only consider the edges of every node once, the only constraint is that when we encounter a node u in the stream, we also need to get all incident edges that are incident to the already seen nodes.

3.4 Randomized Choices

To accelerate convergence, the original QTM algorithm moves a node u only if this reduces the number of edits and there is no rule for breaking ties between moves. The algorithm also never adopts children whose subtrees contain an equal number of neighbors and non-neighbors, as this only swaps edge deletions for insertions. We call such children *indifferent children*. We now propose to break ties by choosing uniformly at random from the best options for u, even if this does not lead to an improvement. The rationale is that on a plateau of equally good solutions only some may lead to better solutions in the next move. The same technique can also be applied to the inclusion-minimal editing, where a more diverse set of solutions can be obtained.

This poses two challenges: we need to find all options, and we may count each of them only once. In particular, different choices of a parent p and children C to adopt might actually yield the same quasi-threshold graph and thus only one of them should be considered. For instance, choosing a parent x without adopting any children is the same as choosing x's parent p as parent and adopting x. But we also cannot disregard p, because adopting a second child of p would yield a different quasi-threshold graph.

Since, according to Lemma 2, the set of simple paths is unique, we can resolve the ambiguity by ensuring that a node u that is moved is inserted at the bottom of its new simple path. The lowest node of a simple path does not have exactly one child,

for otherwise the path would not end there. Accordingly, we ignore positions where u adopts exactly one child.

Thus, if a potential parent p has exactly one close child and no indifferent children, we disregard it. If p has one close child, we must choose at least one indifferent child and thus get $2^{c_i} - 1$ possibilities to choose among the c_i indifferent children. If p has at least two close children, we can choose an arbitrary subset of the indifferent children and get 2^{c_i} possibilities. If p has no close children and at most one indifferent child, we have only the single option of not adopting the child. If p has no close children and $c_i \geq 2$ indifferent children, we have $2^{c_i} - c_i$ possible choices among the indifferent children that do not lead to exactly one child.

In our algorithm, we propagate the number of choices upwards in our bottom-up search together with the minimum number of required edits and the best parent. When processing a node that is a suitable parent that achieves the same number of edits, we choose it with a probability that is proportional to its number of choices for adopting children divided by the total number of choices aggregated so far. As the number of choices is exponential in the number of indifferent children, we store the logarithm of the number of choices to avoid overflows or dealing with huge integers. While this introduces rounding errors when adding numbers that are of different orders of magnitude, in these cases the chances of choosing one parent instead of the other are vanishingly small anyway.

QTM already guarantees that we discover nodes whose subtree contains as many neighbors as non-neighbors, so it is easy to select them. There are some cases though, where QTM needs to be modified to propagate information about equally good parents. In particular, this is the case if the current candidate so far shows no benefit over isolating the node to move. In that case, QTM does not propagate any information as there is always an ancestor of the current node that is at least as good. We adapt QTM to also propagate information about equally good parents even if the number of saved edits is 0. We however do not insert the parent p into the priority queue unless there is an actual improvement over isolating the node to move. The reason for this is that if p is a non-neighbor, it causes an additional edit that leads to -1 saved edits. This cannot be compensated further up in the tree, as otherwise the path above p to the root contained more u-neighbors than non-neighbors and choosing the parent of p as parent of u and not adopting any children was better.

4 Experimental Evaluation

We added our extensions to the original QTM implementation in C++ as part of NetworKit [31 SPP][2]. All experiments were performed on an Intel Core i7-2600K CPU with 32GB RAM. Each algorithms was executed ten times with ten different seeds and randomly permuted node ids. By *instance*, we denote a combination of seed and (permuted) graph.

[2] Our implementation is available at https://github.com/michitux/networkit/tree/upstream/qtm-linear.

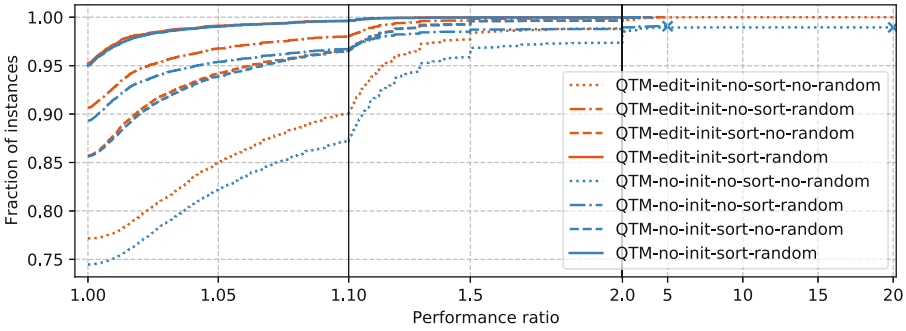

Fig. 2. Comparison of the different variants of QTM on the COQ protein similarity dataset with either no initialization or the initialization heuristic. Lines ending with a "x" are algorithms that need edits for instances that are quasi-threshold graphs and are thus infinitely worse than the best algorithm.

Our algorithms are evaluated on two datasets. The first consists of 3964 connected components of the COG protein similarity data [2,27]. Each connected component consists of a symmetric matrix of similarities, and we construct an unweighted graph from its non-negative entries. Even though the dataset does not include fully connected components (i.e., cliques), 1666 components remain that are quasi-threshold graphs and do not require any edits. We restrict parts of our analysis to the 716 graphs that require at least 20 edits. As a second dataset we use 100 social networks of Facebook friendships at US universities and colleges [32].

Unless noted otherwise, QTM is run for a maximum of 400 iterations. We stop early if an iteration does not result in a node movement. With randomization enabled, however, we do continue for up to 50 iterations without improvements if nodes had more than one option.

We use so-called *performance profiles* [12] to compare the number of edits achieved by different algorithms. A performance profile indicates the fraction of instances on which an algorithm performed within a specified percentage of the best algorithm with the best seed on that graph. For readability, we sometimes divide the plots by vertical lines indicating intervals of the x-axis with different linear scales.

4.1 Sorting Paths and Randomization

We first examine the impact of sorting paths and randomization on the number of edits. In a $2 \times 2 \times 2$-design, we combine no initialization (a spanning forest of isolated nodes) and the previous initialization heuristic [4 SPP] with iterations that make or do not make use of path sorting and randomization.

Figure 2 shows the results for the full COG protein similarity dataset. Despite the many instances that are, or are almost, quasi-threshold graphs, clear differences arise, with the old variants performing the worst. As is to be expected, quasi-threshold graphs are not always recognized without initialization. The variants with just sorting follow with some margin. Here, the difference between the two initialization algorithms is

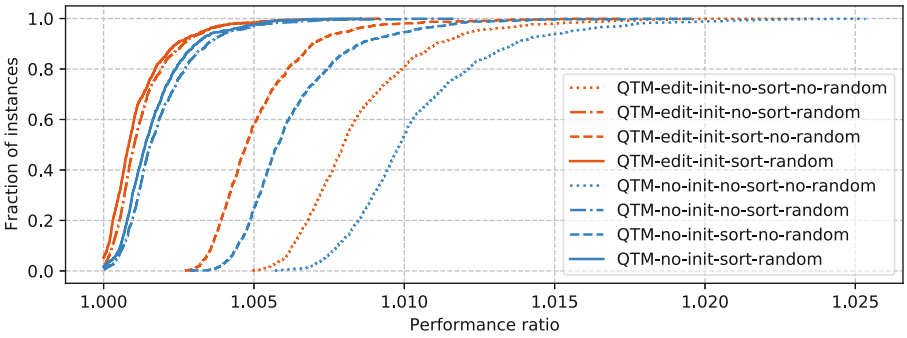

Fig. 3. Comparison of the different variants of QTM on the Facebook 100 dataset with either no initialization or the initialization heuristic.

almost gone and no quasi-threshold graph requires any edits. This is not the case right after the first iteration, though, so we can rule out that one iteration of this algorithm is an alternative inclusion-minimal algorithm.

The versions with just randomization performs even better than with just sorting paths. However, here, some graphs are not recognized as quasi-threshold graphs and a clear gap between the two initializations remains. With path sorting and randomization, the performance is even better, regardless of the initialization 95% of the instances are as good as the best algorithm and seed, and almost all instances are within 10% of the best solution.

For the Facebook 100 dataset, the results that are shown in Fig. 3 are slightly different. First, the instances are much more challenging with even the smallest requiring more than ten thousand edits. There are slight differences between the different solutions which mean that usually there is just one seed and algorithm that achieves the best result on a graph, explaining why no algorithm has the best solution for more than 10% of the instances. Also, we are no longer talking about 10% differences in the number of edits, but at most 2.5%. Still, there are clear differences between the algorithm variants. The original two variants need at least 0.5% more edits than the best solutions on almost all instances while the variants with path sorting and randomization need at most 0.5% more edits than the best solutions on almost all instances. The variants with path sorting give a good improvement, but unlike in the COG protein similarity dataset, the differences between the initializations remain. With randomization, the difference between the two initializations is even larger than the difference between using just randomization and using both path sorting and randomization.

Overall, we can conclude that using path sorting and randomization significantly improve the quality of the solutions. However, on the Facebook 100 dataset, initialization still seems to make a difference, indicating that even with these improvements we are not able to escape all local minima. Next, we consider the inclusion-minimal editing as initialization.

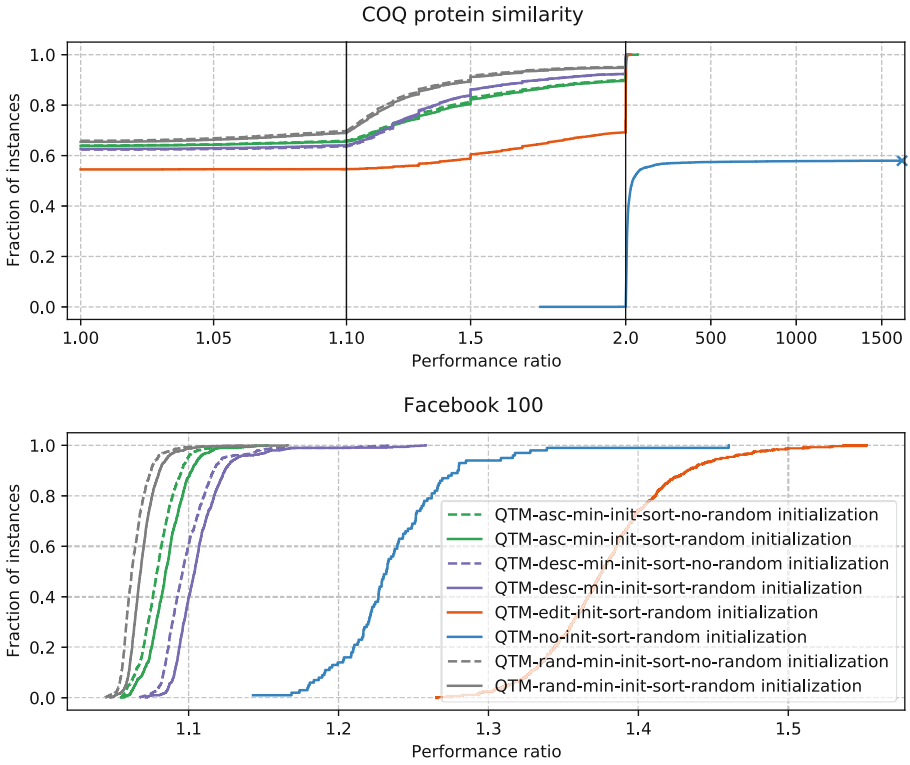

Fig. 4. Comparison of the different initializations of QTM on the COQ protein similarity dataset (top) and the Facebook 100 dataset (bottom).

4.2 Initialization and Convergence

Apart from the two original initialization methods, we consider three variants of the inclusion-minimal editing that differ based on the order in which nodes are inserted. We consider a random order and descending or ascending by degree. For the inclusion-minimal initialization, we also consider randomization of the chosen position in the skeleton.

First, we consider just the initialization itself in Fig. 4 for both datasets. Both plots use as "best algorithm" the algorithm runs with the up to 400 iterations. For the COQ protein similarity dataset, we can see that even just the initialization algorithms also match some of the best solutions, which is to be expected as some of them require no edits. No initialization corresponds to just deleting all edges, and we can see that for some graphs this is very far from an optimal solution. The inclusion-minimal variants clearly need less edits than the initialization heuristic, with the randomized order being best and a not so clear distinction between ascending and descending order. Interestingly, the variants without additional randomization seem to perform slightly better.

On the Facebook 100 dataset, a large fraction of the edges is edited, such that even just deleting all edges is less than 50% worse than what the best algorithm achieves.

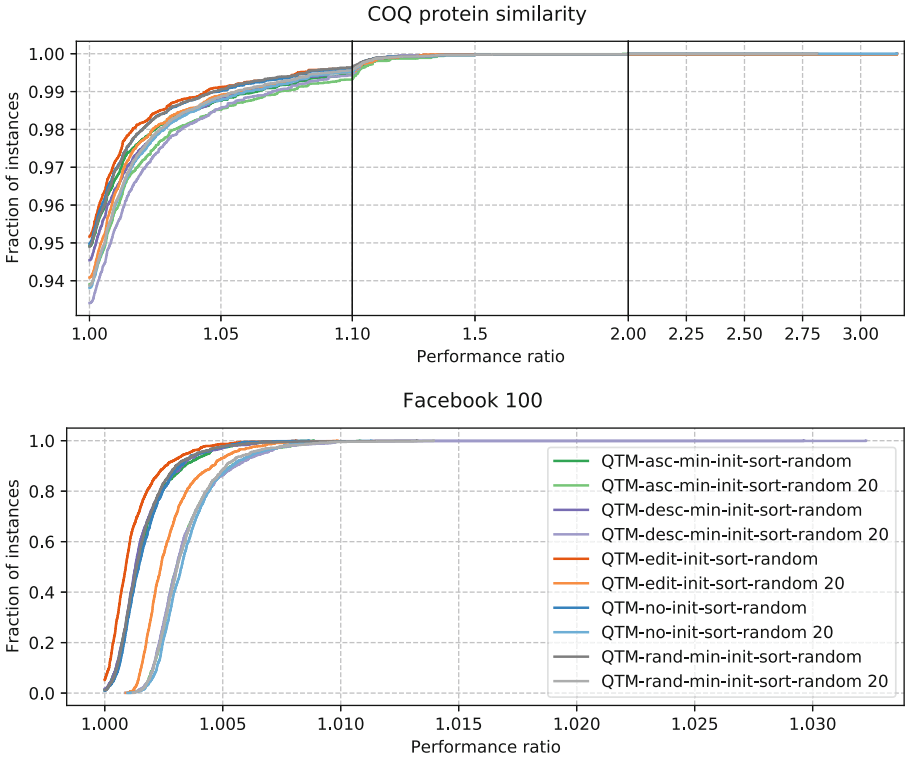

Fig. 5. Comparison of the different initialization variants of QTM on the COQ protein similarity dataset (top) and the Facebook 100 dataset (bottom) with both path sorting and randomization after up to 400 iterations and after 20 iterations.

The initialization heuristic actually needs more edits than just deleting all edges, an observation already made by Brandes et al. [4 SPP]. The inclusion-minimal initialization algorithms perform much better than that, even though they do not match any best results. Again, the randomized order is best, following by ascending and then descending degree order. We can also clearly see again that not randomizing the choices is slightly better. This indicates that there might be potential for further optimizing choices in the inclusion-minimal editing algorithm.

Next, we consider how the choice of the initialization algorithm influences the result after 20 iterations or up to 400 iterations with both path sorting and randomization enabled. Figure 5 compares the results for both datasets. For the COQ protein similarity dataset, the results are very close. The initialization heuristic wins both after 20 and 400 iterations, the inclusion-minimal editing with randomized order comes second. The remaining variants follow, with the descending degree ordering being last. The differences are small, though, and in some cases the initialization seems to be more important than the number of iterations.

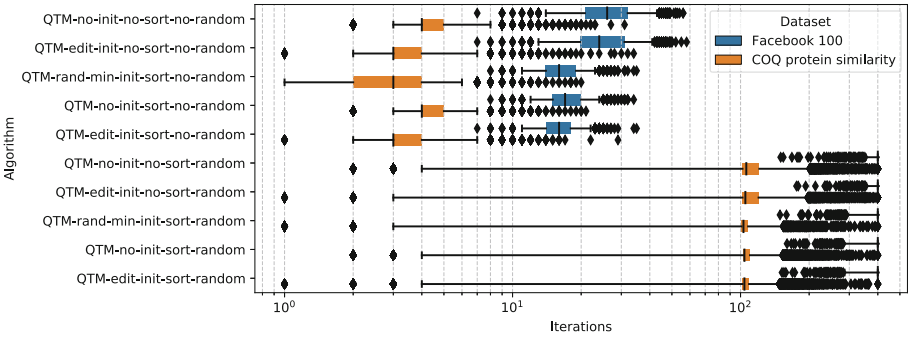

Fig. 6. Number of iterations used by QTM. The COQ protein similarity dataset only includes graphs that require at least 20 edits. Whiskers extend to the 5th and 95th percentile.

For the Facebook 100 dataset, the difference between 20 and 400 iterations is clearly visible. While the initialization heuristic clearly wins, the number of iterations seems more important than the initialization. This can be explained by the much larger and more difficult graphs that also require more iterations as shown in Fig. 6. Here, we exclude the ascending and descending degree ordered initialization to improve readability.

Without randomization, most instances of the COQ protein similarity dataset converge within 10 iterations. On the Facebook 100 dataset, those algorithms require up to 40 iterations for most instances to converge. Enabling path sorting decreases the number of required iterations. As the initialization is not counted as an iteration, it is natural that variants without initialization take an iteration longer in the median on the COQ protein similarity dataset. The difference between the initialization heuristic and the inclusion-minimal editing as initialization is small. With randomization enabled, most instances of the Facebook 100 dataset use all 400 iterations that we allowed. With path sorting enabled, some more instances converge earlier, i.e., either no move was possible – which is unlikely here – or no improvement has been found for 50 iterations. For the COQ protein similarity dataset, most instances finish in a bit more than 100 iterations. Again, this is less with path sorting.

We conclude that the initialization heuristic introduced by Brandes et al. [4 SPP] is still unmatched in results even though it is initially worse than the new inclusion-minimal variants. For the inclusion-minimal editing, a random node order seems to perform best. Path sorting leads not only to better results of QTM, but also faster convergence. Randomization leads to a much larger number of iterations that yield some improvements. Here, limiting the number of iterations is required to achieve reasonable running times but still even with 20 iterations, randomization improves results.

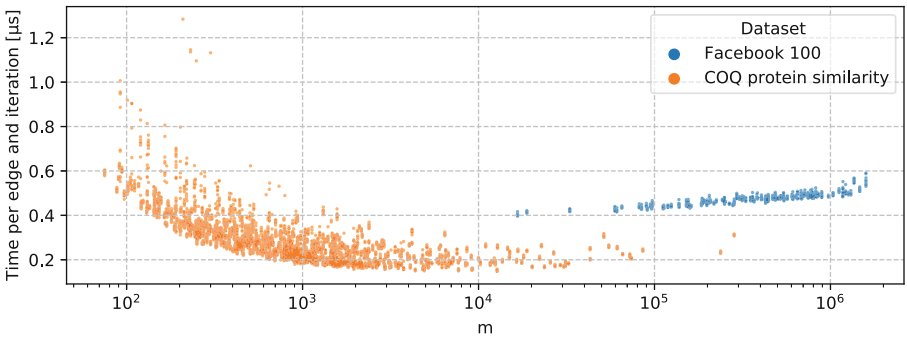

Fig. 7. Running time per edge and iteration vs. number of edges of QTM with initialization heuristic, path sorting and randomization on graphs of the two datasets requiring at least 20 edits.

4.3 Running Time

Figure 7 shows the running time per edge and iteration in microseconds for QTM with initialization heuristic, randomization and path sorting. Although this includes the time for initialization, we normalized by the number of subsequent iterations. Since initialization time is dominated by the iterations, which in turn are linear in the number of edges, this normalized running time should be roughly constant. For the COQ protein similarity dataset, it actually decreases with increasing graph size. Given that this happens in the range where these graphs have only hundreds of edges, initialization overheads might play a role. For the Facebook 100 dataset, running times actually increase slightly with graph size. Between the smallest and the largest graph, we see an increase from around 0.4µs to 0.6µs. We examined CPU statistics and found increased cache misses to be a likely explanation. The percentage of cache misses increases while the number of instructions per edge and iteration is almost constant across the Facebook 100 dataset.

5 Conclusion

We have extended the fast quasi-threshold editing heuristic QTM with new path sorting and randomization components. We have shown that path sorting both provides new local optimality guarantees in theory and better results in practice. Our experimental results indicate that randomization indeed helps escaping local optima, but convergence needs much longer, in particular for large graphs. Still, even with few iterations, results are improved in practice. We also modified QTM into a linear-time algorithm for inclusion-minimal edit sets, which serve well as initialization for QTM. While it reduces the number of edits compared to the previous initialization heuristic, the final result after convergence are slightly worse.

Therefore, it would be interesting to investigate further ways to escape local minima, e.g., by moving several nodes at once by some form of contraction. A recent master's thesis [29] extends QTM to the weighted quasi-threshold editing problem where every node pair has a cost and the goal is to find a set of edits with minimum total cost. It

shows that with non-uniform edit costs, QTM seems to get stuck in local minima and investigates moving whole subtrees as a remedy. While moving subtrees helps, it also significantly increases the running time.

Acknowledgement. This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants BR 2158/11-2 and WA 654/22-2 within the Priority Programme 1736 *Algorithms for Big Data*.

References

1. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Mach. Learn. **56**(1–3), 89–113 (2004). https://doi.org/10.1023/B:MACH.0000033116.57574.95
2. Böcker, S., Briesemeister, S., Bui, Q.B.A., Truß, A.: A fixed-parameter approach for weighted cluster editing. In: APBC, pp. 211–220. Imperial College Press (2008). http://www.comp.nus.edu.sg/%7Ewongls/psZ/apbc2008/apbc050a.pdf
3. Borgatti, S.P., Everett, M.G.: Models of core/periphery structures. Soc. Netw. **21**(4), 375–395 (2000). https://doi.org/10.1016/S0378-8733(99)00019-2
4 SPP. Brandes, U., Hamann, M., Strasser, B., Wagner, D.: Fast quasi-threshold editing. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 251–262. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_22
5. Brandes, U., Holm, E., Karrenbauer, A.: Cliques in regular graphs and the core-periphery problem in social networks. In: Chan, T.-H.H., Li, M., Wang, L. (eds.) COCOA 2016. LNCS, vol. 10043, pp. 175–186. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48749-6_13
6. Bruckner, S., Hüffner, F., Komusiewicz, C.: A graph modification approach for finding core-periphery structures in protein interaction networks. Algorithms Mol. Biol. **10**, 16 (2015). https://doi.org/10.1186/s13015-015-0043-7
7. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inf. Process. Lett. **58**(4), 171–176 (1996). https://doi.org/10.1016/0020-0190(96)00050-6
8. Chu, F.P.M.: A simple linear time certifying LBFS-based algorithm for recognizing trivially perfect graphs and their complements. Inf. Process. Lett. **107**(1), 7–12 (2008). https://doi.org/10.1016/j.ipl.2007.12.009
9. Corneil, D.G., Lerchs, H., Burlingham, L.S.: Complement reducible graphs. Discret. Appl. Math. **3**(3), 163–174 (1981). https://doi.org/10.1016/0166-218X(81)90013-5
10. Creignou, N., Ktari, R., Meier, A., Müller, J., Olive, F., Vollmer, H.: Parameterised enumeration for modification problems. Algorithms **12**(9), 189 (2019). https://doi.org/10.3390/a12090189
11. Crespelle, C.: Linear-time minimal cograph editing (2019). https://perso.ens-lyon.fr/christophe.crespelle/publications/SUB_minimal-cograph-editing.pdf
12. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002). https://doi.org/10.1007/s101070100263
13. van Dongen, S.M.: Graph Clustering by Flow Simulation. Ph.D. thesis, University of Utrecht (2000)
14. Drange, P.G., Dregi, M.S., Lokshtanov, D., Sullivan, B.D.: On the threshold of intractability. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 411–423. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_35
15. Drange, P.G., Pilipczuk, M.: A polynomial kernel for trivially perfect editing. Algorithmica **80**(12), 3481–3524 (2017). https://doi.org/10.1007/s00453-017-0401-6

16. Foldes, S., Hammer, P.L.: The Dilworth number of a graph. Ann. Discrete Math. **2**, 211–219 (1978). https://doi.org/10.1016/S0167-5060(08)70334-0
17. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3–5), 75–174 (2010). https://doi.org/10.1016/j.physrep.2009.11.002
18 SPP. Gottesbüren, L., Hamann, M., Schoch, P., Strasser, B., Wagner, D., Zühlsdorf, S.: Engineering exact quasi-threshold editing. In: SEA, pp. 10:1–10:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.SEA.2020.10
19. Hammer, P.L., Simeone, B.: The splittance of a graph. Combinatorica **1**(3), 275–284 (1981). https://doi.org/10.1007/BF02579333
20. Liu, Y., Wang, J., Guo, J., Chen, J.: Complexity and parameterized algorithms for cograph editing. Theor. Comput. Sci. **461**, 45–54 (2012). https://doi.org/10.1016/j.tcs.2011.11.040
21. Liu, Y., Wang, J., You, J., Chen, J., Cao, Y.: Edge deletion problems: branching facilitated by modular decomposition. Theor. Comput. Sci. **573**, 63–70 (2015). https://doi.org/10.1016/j.tcs.2015.01.049
22. Luce, R.D., Perry, A.: A method of matrix analysis of group structure. Psychometrika **14**, 95–116 (1949). https://doi.org/10.1007/BF02289146
23. Mahadev, N.V., Peled, U.N.: Threshold Graphs and Related Topics. Ann. Discrete Math. **56**. Elsevier (1995)
24. Mariani, M.S., Ren, Z.M., Bascompte, J., Tessone, C.J.: Nestedness in complex networks: observation, emergence, and implications. Phys. Rep. **813**, 1–90 (2019). https://doi.org/10.1016/j.physrep.2019.04.001
25. Nastos, J., Gao, Y.: Familial groups in social networks. Soc. Netw. **35**(3), 439–450 (2013). https://doi.org/10.1016/j.socnet.2013.05.001
26. Ohtsuki, T., Mori, H., Kashiwabara, T., Fujisawa, T.: On minimal augmentation of a graph to obtain an interval graph. J. Comput. Syst. Sci. **22**(1), 60–97 (1981). https://doi.org/10.1016/0022-0000(81)90022-2
27. Rahmann, S., Wittkop, T., Baumbach, J., Martin, M., Truß, A., Böcker, S.: Exact and Heuristic Algorithms for Weighted Cluster Editing. In: CSB, pp. 391–401 (2007). https://doi.org/10.1142/9781860948732_0040
28. Schaeffer, S.E.: Graph clustering. Comput. Sci. Rev. **1**(1), 27–64 (2007). https://doi.org/10.1016/j.cosrev.2007.05.001
29. Schmitt, D.: Engineering Heuristic Quasi-Threshold Editing. Master's thesis, Karlsruhe Institute of Technology (2021). https://i11www.iti.kit.edu/_media/teaching/theses/ma-schmitt-21.pdf
30. Schoch, D., Brandes, U.: Re-conceptualizing centrality in social networks. Eur. J. Appl. Math. **27**(6), 971–985 (2016). https://doi.org/10.1017/S0956792516000401
31 SPP. Staudt, C.L., Sazonovs, A., Meyerhenke, H.: Networkit: a tool suite for large-scale complex network analysis. Netw. Sci. **4**(4), 508–530 (2016). https://doi.org/10.1017/nws.2016.20
32. Traud, A.L., Mucha, P.J., Porter, M.A.: Social structure of Facebook networks. Phys. A: Stat. Mech. Appl. **391**(16), 4165–4180 (2012). https://doi.org/10.1016/j.physa.2011.12.021
33. Wolk, E.S.: A note on "The comparability graph of a tree". Proc. AMS **16**(1), 17–20 (1965). https://doi.org/10.2307/2033992
34. Yan, J., Chen, J., Chang, G.J.: Quasi-threshold graphs. Discret. Appl. Math. **69**(3), 247–255 (1996). https://doi.org/10.1016/0166-218X(96)00094-7