Generalized graph clustering: recognizing (p, q)-cluster graphs*

Pinar Heggernes† Daniel Lokshtanov† Jesper Nederlof† Christophe Paul‡
Jan Arne Telle†

Abstract

Cluster Editing is a classical graph theoretic approach to tackle the problem of data set clustering: it consists of modifying a similarity graph into a disjoint union of cliques, i.e, clusters. As pointed out in a number of recent papers, the cluster editing model is too rigid to capture common features of real data sets. Several generalizations have thereby been proposed. In this paper, we introduce (p, q)-cluster graphs, where each cluster misses at most p edges to be a clique, and there are at most q edges between a cluster and other clusters. Our generalization is the first one that allows a large number of false positives and negatives in total, while bounding the number of these locally for each cluster by p and q. We show that recognizing (p, q)-cluster graphs is NP-complete when p and q are input. On the positive side, we show that $(0, q)$-cluster, $(p, 1)$-cluster, $(p, 2)$-cluster, and $(1, 3)$-cluster graphs can be recognized in polynomial time.

1 Introduction

Clustering is an optimization problem having applications in many fields ranging from bioinformatics [16, 1] to image processing [19], with various algorithmic approaches available [15]. The general idea of clustering is to partition a set of data items into subsets, called clusters, in such a way that highly similar items belong to the same cluster and items having low similarity belong to different clusters. The input typically consists of similarity values between pairs of items and in the graph-based approach to clustering the items correspond to vertices, with two vertices being adjacent if and only if their similarity value exceeds a fixed threshold θ [13]. In a perfect setting with no noise, an appropriate threshold yields a similarity graph whose connected components (or clusters) are cliques. However, in most cases there will be noise, both false positives (presence of an edge that should not have been present) and false negatives (missing edges). Shamir et al. [18] initiated a study of clustering in terms of graph modification problems with a focus on the Cluster Editing problem: modify a given graph by adding and deleting at most k edges to obtain a disjoint union of cliques.

Cluster Editing, parameterized by the number k of false positives and negatives, is FPT [3, 9, 10]. Furthermore, it has a polynomial-time 4-approximation algorithm but it does not admit a PTAS unless P = NP [14]. Several drawbacks of this model have been pointed out (see e.g. [4, 6]): for low values of the parameter k, it does not capture instances with a high number of

*This work is supported by the Research Council of Norway.

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway. {pinar.heggernes, daniel.lokshtanov, jesper.nederlof, jan.arne.telle}@ii.uib.no

‡CNRS, LIRMM, Université Montpellier 2, France. paul@lirrm.fr

false positives and negatives, nor does it allow overlap between clusters. As it has been observed that clusters do not always represent an equivalence relation (see [8, 17]), overlapping clusters have been considered [5, 7]. In addition, a weighted version of Cluster Editing has been considered to capture the fact that the costs of fixing false positives and of false negatives can differ [2]. Other variants to tackle data sets containing a large number of false negatives have been proposed [12, 11]. The p-Defective Clique Editing problem is introduced by Guo et al. [11]: modify a given graph by adding and deleting at most k edges to obtain a disjoint union of p-defective cliques, where a p-defective clique is a graph missing at most p edges from being a clique. An FPT algorithm, parameterized by p and k, is given for this problem [11]. Note that for low values of the parameters the p-Defective Clique Editing problem allows a high number of false negatives, as long as the noise is distributed among the clusters, but it does not allow a high number of false positives.

In this paper we present an alternative approach to graph clustering that allows a high total number of both false negatives and false positives, but little noise related to each cluster, by introducing what we call (p, q)-cluster graphs.

(p, q)-Cluster Graph Recognition
Input: A graph G and two integers p, q.
Question: Can the vertex set of G be partitioned into subsets with each subset missing at most p edges from being a clique (i.e. inducing a p-defective clique) and having at most q edges going to other subsets?

Note that $(0, 0)$-cluster graphs are exactly cluster graphs. A (p, q)-cluster graph can have low values of p and q, while having a high total number of false negatives and false positives. In that case the similarity values and threshold θ satisfy the reasonable constraint that in each cluster C of similar items we find at most p pairs $u, v \in C$ with similarity less than θ, and at most q pairs $u \in C, w \notin C$ with similarity higher than θ. Observe also that tuning the p and q parameters independently is an alternative attempt of assigning different roles or importance to false positives and false negatives [2]. Moreover the transitivity constraint which has been criticized in Cluster Editing [8, 17] is relaxed. In this way the (p, q)-Cluster Graph Recognition problem, or its editing version, answers most of the drawbacks present in the Cluster Editing problem. Thus, as a first task, we want efficient algorithms for (p, q)-Cluster Graph Recognition. Not surprisingly, (p, q)-Cluster Graph Recognition is NP-complete, as we show in Theorem 3. However, as we summarize in Figure 1, there are various values of p and q for which (p, q)-Cluster Graph Recognition can be solved in polynomial time, some trivially and some by more complicated combinatorial arguments.

On the one hand, $(p, 0)$-Cluster Graph Recognition corresponds to the p-Defective Clique Editing problem allowing zero edge modifications and is therefore trivial since the answer is yes if and only if each connected component of the input graph is a p-defective clique. On the other hand, $(0, q)$-Cluster Graph Recognition is not at all simple, as there are many ways to partition the vertex set of a graph into a collection of cliques. In particular, similar problems like partitioning the vertex set of a graph into a minimum number of cliques (Partition Into Cliques), or into subsets of bounded size each having a bounded number of edges to other subsets (Minimum Degree Graph Partition), are both NP-hard. Hence it is surprising that $(0, q)$-Cluster Graph Recognition can be solved in polynomial time, as we prove in Theorem 10. We also show that $(p, 1)$-cluster and $(p, 2)$-cluster graphs can be recognized in polynomial time. Let us emphasize that the algorithms presented in this paper are

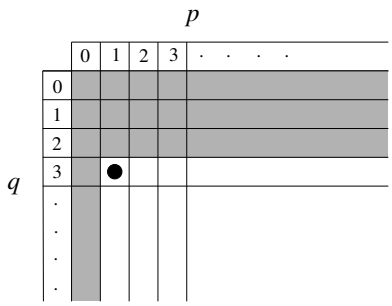


Figure 1: The shaded area and the dot indicate p and q values for which (p, q)-cluster graphs are polynomial-time recognizable.

polynomial in both p, q and the size of the graph. For example, the algorithm for $(0, q)$-Cluster Graph Recognition runs in time $O(n^3)$ and by binary search one can find the smallest q such that the input graph is a $(0, q)$-Cluster Graph.

The polynomial-time cases mentioned so far are summarized by the shaded area of the table given in Figure 1. The first interesting case outside of the shaded area is the recognition of $(1, 3)$-cluster graphs. With careful reduction rules and a computerized case analysis we are able to show that also $(1, 3)$-cluster graphs can be recognized in polynomial time (Theorem 14). We must leave as an open problem the complexity, up to P vs NP-complete, of (p, q)-Cluster Graph Recognition for all possible values of p and q (not to speak of its parameterized complexity or its edge modification variants).

2 Preliminaries

We consider undirected finite graphs with no loops or multiple edges. For a graph $G = (V, E)$, we denote its vertex and edge set by V and E, respectively, with $n = |V|$ and $m = |E|$. For $S \subseteq V$, the subgraph of G induced by S is denoted by $G[S]$.

The *neighborhood* of a vertex x of G is $N_G(x) = \{v \mid xv \in E\}$. The *closed neighborhood* of x is defined as $N_G[x] = N_G(x) \cup \{x\}$. If $S \subseteq V$, then the neighbors of S, denoted by $N_G(S)$, are given by $\bigcup_{x \in S} N_G(x) \setminus S$. The *degree* of a vertex x in G is $d_G(x) = |N_G(x)|$. We will omit the subscripts when there is no misunderstanding.

A *clique* is a set of vertices that are pairwise adjacent. A vertex x is called *simplicial* if $N(x)$ is a clique. If a vertex set C has exactly p pairs of non-adjacent vertices, we say that C *misses* p *edges*. We will call a vertex set that misses at most p edges a p*-group*. A p-group C such that there are at most q edges in G with exactly one endpoint in C is called a (p, q)*-group*.

For two non-negative integers p and q, a graph $G = (V, E)$ is a (p, q)*-cluster graph* if V can be partitioned into (p, q)-groups. Note that this condition is equivalent to the condition in the question of the (p, q)-Cluster Graph Recognition problem. As deleting vertices from G cannot disturb a partition into (p, q)-groups, (p, q)-cluster graphs are hereditary, i.e., being a (p, q)-cluster graph is preserved under taking induced subgraphs.

Clearly, a graph is a (p, q)-cluster graph if and only if every connected component of it is a (p, q)-cluster graph. Hence for the rest of the paper, we will assume that the input graph is connected. If not, we can run the presented algorithms on each connected component. As a

consequence, we can also restrict ourselves to identifying *connected* (p, q)-groups: a (p, q)-group C of a graph G might induce a disconnected subgraph, but then every connected component of $G[C]$ is also a (p, q)-group of G.

3 (p, q)-Cluster Graph Recognition is NP-complete

In this section we prove that, given as input a graph G and two integers p and q, it is NP-complete to decide whether G is a (p, q)-cluster graph. We use a reduction from the well known NP-complete problem Clique: Given a graph G and an integer k, $0 < k < n$, does G have a clique of size at least k?

Let $G_1 = (V_1, E_1)$ and k be input to Clique, where $|V_1| = n$. We show how to construct a graph G_2 and integers p and q such that G_1 has a clique of size at least k if and only if G_2 is a (p, q)-cluster graph. Let us first define

$$\alpha = nk - k^2 + 1 \qquad q = (n - k + 1)\alpha - 1 \qquad \beta = q - \alpha + 2 \qquad p = \beta k \ .$$

Note that $\alpha \geq n$ and $q \geq 2\alpha - 1$, since $0 < k < n$. We obtain G_2 from G_1 as follows:

1. Add a clique A of size α to G_1, and add edges between each vertex in A and each vertex in V_1. Call the resulting graph G_1'.
2. Add a clique B of size β to G_1', and add edges between each vertex in B and each vertex in A. Call the resulting graph G_2.

Lemma 1. *Let $G_1 = (V_1, E_1)$, G_2, p, and q be as described above. Then G_1 has a clique of size at least k if and only if there is a non-empty set $C \subseteq V_1$ such that $A \cup B \cup C$ is a (p, q)-group in G_2.*

Proof. Let C be a subset of V_1 and assume that $S = A \cup B \cup C$ is a (p, q)-group in G_2. Let $\ell = |C| \geq 1$. The number of edges in G_2 with exactly one endpoint in S is at least $(n - \ell)\alpha$. Since S is a (p, q)-group, $(n - \ell)\alpha \leq q = (n - k + 1)\alpha - 1$, which implies that $\ell \geq k$. Let j be the number of edges that C misses. Then S misses $\beta\ell + j$ edges. But since S is a (p, q)-group, $\beta\ell + j \leq p = \beta k$, and using $\ell \geq k$ gives $j = 0$ and $k = \ell$. Thus C is a clique of size k.

For the other direction, assume that G_1 has a clique of size at least k, and let C be a clique of size exactly k in G_1. Let $S = A \cup B \cup C$. The number edges that S misses is $\beta k = p$. The number of edges with exactly one endpoint in S is at most $(n - k)(k + \alpha)$.

$$\begin{aligned} nk - k^2 = \alpha - 1 \quad &\Leftrightarrow \quad (n - k)\alpha + nk - k^2 = (n - k)\alpha + \alpha - 1 \\ &\Leftrightarrow \quad (n - k)(k + \alpha) = (n - k + 1)\alpha - 1 = q \ . \end{aligned}$$

Thus, $(n - k)(k + \alpha) = q$ and S is a (p, q)-group in G_2. □

Lemma 2. *Let $G_1 = (V_1, E_1)$, G_2, p, and q be as described above. Then G_2 is a (p, q)-cluster graph if and only if there is a non-empty set $C \subseteq V_1$ such that $A \cup B \cup C$ is a (p, q)-group in G_2.*

Proof. Assume first that $G_2 = (V_2, E_2)$ is a (p, q)-cluster graph. We have to show that any (p, q)-group in G_2 that intersects with $A \cup B$ has to contain every vertex of $A \cup B$ and at least a vertex of V_1. Let $S \subseteq V_2$ be a (p, q)-group such that $S \cap (A \cup B) \neq \emptyset$. Observe first that S cannot be a proper subset of $A \cup B$, because any partition of $A \cup B$ into subsets results in more than q edges with an endpoint in each of the subsets, since $A \cup B$ is a clique of size $\alpha + \beta = q + 2$.

Furthermore, S cannot be equal to $A \cup B$, since the number of edges between A and V_1 is αn and $q \leq \alpha n - 1$. Consequently, S must contain whole $A \cup B$ and at least a vertex of V_1.

For the other direction, assume that $S = A \cup B \cup C$ is a (p, q)-group for a non-empty set $C \subseteq V_1$. Observe that for any $v \in V_1 \setminus C$, the set $\{v\}$ is a (p, q)-group, since the degree of v is at most $n + \alpha - 1$, and $q \geq 2\alpha - 1 \geq n + \alpha - 1$. Hence S and the collection the single vertex groups for each vertex in $V_1 \setminus C$ defines a partition of V_2 into (p, q)-groups and consequently G_2 is a (p, q)-cluster graph. □

Theorem 3. *Given a graph G and integers p and q, it is NP-complete to decide whether G is a (p, q)-cluster graph.*

Proof. Combining Lemmas 1 and 2, we conclude that G_1 has a clique of size at least k if and only if G_2 is a (p, q)-cluster graph. Since G_2 can be constructed from G_1 in polynomial time, the theorem follows. □

4 Polynomial-time recognizable (p, q)-cluster graphs

In this section we show that for p and q values that correspond to the shaded area and the black dot in the table in Figure 1, (p, q)-cluster graphs can be recognized in polynomial time. Recall that we can assume the input graph to be connected.

As mentioned in the introduction, recognizing $(p, 0)$-cluster graphs is trivial for every integer p, as it is equivalent to checking whether the input graph misses at most p edges.

For recognizing connected $(p, 1)$-cluster graphs, note that the vertex set of such a graph is either a p-group or consists of two connected p-groups with a single edge between them. Hence we can first check whether the input graph is a $(p, 0)$-cluster graph. If not, we can check for each bridge in the graph, whether the removal of this bridge results in two connected components each of which is a p-group. This can clearly be done in polynomial time.

4.1 Polynomial-time recognition of $(p, 2)$-cluster graphs

Assume that a given connected graph $G = (V, E)$ is a $(p, 2)$-cluster graph. Then V has a partition into $(p, 2)$-groups $V_1, V_2, \ldots, V_k$. For convenience, in this subsection we call a $(p, 2)$-group simply a *group*. Let us define a graph H which has vertices $v_1, v_2, \ldots, v_k$ and edges $v_i v_j$ if G has an edge with an endpoint in V_i and an endpoint in V_j. Note that for each edge $v_i v_j$ of H, there can be at most one edge with an endpoint in V_i and an endpoint in V_j in G (except the case where the $(p, 2)$-partition consists of only two groups). Clearly H is a connected graph of maximum degree 2, which means that it is a path or a cycle. Furthermore, the removal of any two edges of H is equivalent to the removal of exactly two edges of G (except the case where H has only two vertices). We will use this property to decide whether a given graph G is a $(p, 2)$-cluster graph. For this purpose we describe a dynamic programming algorithm.

For every pair of edges $e_1 = u_1 v_1$ and $e_2 = u_2 v_2$ of G, we check whether u_1 and v_1 appear in different connected components of $G' = (V, E \setminus \{e_1, e_2\})$, and u_2 and v_2 appear in different connected components of G'. If so, then we say that $\{e_1, e_2\}$ is a *cut* of G. Let $L(e_1, e_2, u_1, u_2)$ be the disjoint union of all connected components of G' containing u_1 or u_2. One can think of the cut $\{e_1, e_2\}$ having two "sides", $L(e_1, e_2, u_1, u_2)$ and $L(e_1, e_2, v_1, v_2)$ respectively. We define a function $T(e_1, e_2, u_1, u_2)$ that is true if and only if $\{e_1, e_2\}$ is a cut and $L(e_1, e_2, u_1, u_2)$ can be partitioned into groups. Then the following recurrence holds for T.

- If $\{e_1, e_2\}$ is not a cut then $T(e_1, e_2, x, y)$ is `False`, for all x, y.
- Otherwise, if every connected component of $L(e_1, e_2, u_1, u_2)$ is a group then $T(e_1, e_2, u_1, u_2)$ is `True`.
- Otherwise $T(e_1, e_2, u_1, u_2)$ is `True` if and only if there is an edge $e = uv \in E$ with $u, v \in L(e_1, e_2, u_1, u_2)$ such that
  - every connected component of $L(e, e_1, u, u_1)$ is a group and $T(e, e_2, v, u_2)$ is true, *or*
  - every connected component of $L(e, e_2, v, u_2)$ is a group and $T(e, e_1, u, u_1)$ is true.

For every pair of edges e_1 and e_2, we compute $T(e_1, e_2, u_1, u_2)$, $T(e_1, e_2, u_1, v_2)$, $T(e_1, e_2, v_1, u_2)$, and $T(e_1, e_2, v_1, v_2)$, using the above formula. After all this has been computed, we check for every pair of edges e_1 and e_2 whether they can be two consecutive edges of H. To do this, we simply check whether $T(e_1, e_2, x, y)$ is true and $G[V \setminus L(e_1, e_2, x, y)]$ is a connected group for some edges e_1 and e_2 and endpoints x of e_1 and y of e_2. If such edges e_1, e_2 and endpoints x, y exist we conclude that G is a $(p, 2)$-cluster graph. Otherwise it is not a $(p, 2)$-cluster graph. The necessary computations can be done in a straight forward way in time $O(m^3)$. With a few extra reduction rules and more clever dynamic programming it is possible to reduce this running time considerably.

4.2 Polynomial-time recognition of $(0, q)$-cluster graphs

Deciding whether a given graph $G = (V, E)$ is a $(0, q)$-cluster graph is deciding whether V can be partitioned into $(0, q)$-groups. This is equivalent to partitioning V into cliques, such that for each of these cliques G has at most q edges with exactly one endpoint in that clique. Analogous to previous subsection, in this subsection we will call a $(0, q)$-group simply a *group*. Also, we call a vertex of v of G a *high degree vertex* if $d(v) \geq q + 1$. We start with some observations on $(0, q)$-cluster graphs, the proofs of which are given in the appendix.

Observation 4. *Let $G = (V, E)$ be a $(0, q)$-cluster graph. Then there is a partition of V into groups such that every group either consists of a single vertex or contains a high degree vertex.*

Observation 5. *Let $G = (V, E)$ be a $(0, q)$-cluster graph. Then there is a partition of V into groups such that every group C with at least two vertices contains a vertex v with $N[v] = C$.*

Note that Observation 5 is equivalent to saying that C contains a simplicial vertex. By the above observations, we can restrict our search to groups that are either singletons or contain a simplicial vertex and a high degree vertex.

Definition 6. A group C is a *good group* if $C = N[v]$ for some simplicial vertex v, C contains a high degree vertex, and there are at most q edges in the graph with exactly one endpoint in C (the last condition is implicit from the definition of group).

Lemma 7. *A graph is a $(0, q)$-cluster graph if and only if its high degree vertices can be covered by non-overlapping good groups.*

Proof. Let $G = (V, E)$ be a graph, and assume that G has a set of good groups $C_1, ..., C_\ell$, such that $C_i \cap C_j = \emptyset$ for every pair $i \neq j$ between 1 and ℓ, and every high degree vertex of G belongs to some C_i for $1 \leq i \leq \ell$. Since these good groups do not overlap, each high degree vertex belongs to a unique good group. Each of these groups has at most q edges leaving the group. Every vertex of G that does not appear in one of these groups, has degree at most q and is a

$(0, q)$-group on its own. Let $v_1, ..., v_t$ be such vertices of G. Then $C_1, ..., C_\ell, \{v_1\}, ..., \{v_t\}$ is a partition of V into clusters, and hence G is a $(0, q)$-cluster graph.

The other direction follows from the fact that a good group containing a high degree vertex is of size at least 2 and Observation 5. □

For the next lemma, we say that a good group is *maximal* if its set of high degree vertices is not a proper subset of the set of high degree vertices of another good group.

Lemma 8. *Let G be a graph and let C_1, C_2 be two maximal good groups of G, such that C_1 has a high degree vertex not in C_2, and C_2 has a high degree vertex not in C_1. Then $C_1 \cap C_2 = \emptyset$.*

Proof. Let $C_1 \cap C_2 = X$. Let v_1 be a high degree vertex of C_1 not in C_2, and let v_2 be a high degree vertex of C_2 not in C_1. Hence $v_1, v_2 \notin X$. Observe first that $|X| \leq q$ because otherwise, since C_2 is a clique, there would be more than q edges from C_1 to v_2, contradicting that C_1 is a good group. If v_1 has a neighbor outside of C_1 then $N[v_1] \neq C_1$, hence C_1 has another vertex v_1' such that $N[v_1'] = C_1$, and $v_1' \notin X$. If v_1 has no neighbor outside of C_1, then since $d(v_1) \geq q + 1$ and $|X| \leq q$, again C_1 has a vertex $v_1' \neq v_1$ such that $v_1' \notin X$. Hence $|C_1| \geq |X| + 2$. With the same arguments, C_2 has a vertex $v_2' \neq v_2$ such that $v_2' \notin X$, and thus also $|C_2| \geq |X| + 2$.

Since $d(v_1) \geq q + 1$, v_1 has at least $q + 1 - (|C_1| - 1)$ neighbors outside of C_1. In addition, there are $|X|(|C_2| - |X|)$ edges between C_1 and $C_2 \setminus X$. Since C_1 is a good group, we must thus have: $q + 1 - (|C_1| - 1) + |X|(|C_2| - |X|) \leq q$. Symmetrically, and with the same arguments, we conclude that: $q + 1 - (|C_2| - 1) + |X|(|C_1| - |X|) \leq q$. Adding up these two inequalities and simplifying, we get:

$$4 + (|X| - 1)|C_1| + (|X| - 1)|C_2| - 2|X|^2 \leq 0$$

Recall that $|C_1| \geq |X| + 2$ and $|C_2| \geq |X| + 2$. Hence we can conclude:

$$4 + (|X| - 1)(|X| + 2) + (|X| - 1)(|X| + 2) - 2|X|^2 \leq 0$$

Doing the arithmetic, we see that the above inequality reduces to $2|X|^2 \leq 0$, and hence we can conclude that $|X| = 0$, which completes the proof. □

Consequently, maximal good groups with different sets of high degree vertices do not overlap. With this, we reach the desired characterization.

Theorem 9. *A graph is a $(0, q)$-cluster graph if and only if every high degree vertex belongs to a good group.*

Proof. Let G be a graph. If G has a high degree vertex v that does not belong to any good group, then G is clearly not a $(0, q)$-cluster graph, due to Lemma 7 and since v cannot define a good group on its own due to its high degree. For the other direction, assume that every high degree vertex of G belongs to a good group. Repeatedly take a maximal good group containing uncovered high degree vertices, and call $\mathcal{C}$ the resulting set of good groups. $\mathcal{C}$ covers all the high degree vertices of G, and the good groups of $\mathcal{C}$ pairwise have different sets of high degree vertices. Thus by Lemma 8, they are pairwise non-overlapping. Consequently, by Lemma 7, G is a $(0, q)$-cluster graph. □

Theorem 10. *Given a graph G and an integer q, it can be decided in polynomial time whether G is a $(0, q)$-cluster graph.*

Proof. Note first that finding the good groups of any graph $G = (V, E)$ can be done in polynomial time, as we only need to check whether $N[v]$ is a clique, contains a high degree vertex, and G has at most q edges with exactly one endpoint in $N[v]$, for each vertex $v \in V$. Now, by Theorem 9, it simply remains to check whether every high degree vertex appears in a good group, which can be done by the procedure described in the proof of Theorem 9. A straight forward implementation gives a total running time of $O(n^3)$, which can probably be improved. □

4.3 Polynomial-time recognition of $(1, 3)$-cluster graphs

Each polynomial-time algorithm that we have given so far has corresponded to a whole row or a whole column of the table given in Figure 1. In fact, with the algorithms that we have given, we have now proved that (p, q)-graphs are recognizable in polynomial time for values of p and q that correspond to the whole shaded area in that table. From here on, we see that the first natural case to study, with respect to NP-completeness versus polynomial-time computability, for a single value of p and a single value of q is the recognition of $(1, 3)$-cluster graphs. This corresponds to the dot in the table. In this subsection, we show that $(1, 3)$-cluster graphs can be recognized in polynomial time.

Analogous to previous subsections, we will refer to a $(1, 3)$-group simply as a *group* in this subsection. A *minimal group* is a group that cannot be partitioned into smaller groups. If there is a partition of the vertex set of a graph into groups then there is also a partition into minimal groups. A *high degree vertex* is now a vertex of degree at least 4. A 1-*group* is a clique missing at most one edge, according to the definitions in Section 2. (Note that a 1-group is not necessarily a group.) A *maximal* 1-group is a 1-group that is not a proper subset of another 1-group. (Note the difference from the definition of maximality in the previous subsection.)

We start this subsection with some reduction rules given in Definition 11. The first steps of our algorithm will be to apply these rules until they cannot be applied anymore, to obtain a *reduced* graph. For these rules, we define the concept of *penalizing* a vertex x as follows. Let u be a vertex of G, and let $G' = G[V \setminus \{u\}]$. Let x be a vertex of $N_G(u)$. Then $d_{G'}(x) = d_G(x) - 1$. When we *penalize* x, we keep the degree of x unchanged. Hence we let $d_{G'}(v) = d_G(v)$ and keep it artificially high.

Definition 11. We say that a graph G is *reduced* if the following reduction rules cannot be applied to it:

1. If, for an edge uv, there is no group that contains both u and v, then delete edge uv and penalize u and v.
2. If G contains a maximal 1-group C of size at least 5, then delete C and penalize the vertices of $N_G(C)$ accordingly.
3. If Rule 2 cannot be applied and G contains a clique C of size at least 4, then delete C and penalize the vertices of $N_G(C)$ accordingly.
4. If Rules 2 and 3 cannot be applied and G contains a 1-group C of size 4, such that C has 3 vertices with neighbors outside of C, then delete C, and penalize $N_G(C)$ accordingly.

The proof of the following two lemmas can be found in Appendix.

Lemma 12. *Given a graph G, the reduced graph G' obtained from G by applying the reduction rules in Definition 11 can be computed in polynomial time. Moreover, G is a $(1, 3)$-cluster graph if and only if G' is a $(1, 3)$-cluster graph.*

Consequently, from now on we can assume that our input graph G is reduced. In particular, G has no groups of size larger than 4. We will call a group C a *leaf group* if at most one vertex of C has neighbors outside of C. Since we assume G to be connected, this is equivalent to C having exactly one vertex with neighbors outside of C.

Lemma 13. *In a reduced graph with more than 42 vertices, every high degree vertex appears in at most 2 minimal non-leaf groups.*

The above lemma enables us to show the main result of this subsection, stated in the following theorem.

Theorem 14. $(1,3)$*-cluster graphs can be recognized in polynomial time.*

Proof. Given a graph H, we can compute a reduced graph G in polynomial time by Lemma 12. If G has at most 42 vertices, we can solve the problem in constant time. Assume that G has $n > 42$ vertices.

First we show that there is a polynomial time reduction from $(1,3)$-Cluster Graph Recognition to Sat. Given a reduced graph G, we describe an instance of Sat obtained from G, as follows. For every minimal group, we make a variable x. For every high degree vertex v, we make a clause $(x_1 \vee x_2 \vee ... \vee x_t)$, where $x_1, ..., x_t$ are the variables corresponding to the t minimal groups containing v. For every pair of overlapping minimal groups with corresponding variables x and y, we make a clause $(\bar{x} \vee \bar{y})$. Clearly, G is a $(1,3)$-cluster graph if and only if the created formula is satisfiable. By Lemma 12, the same is true for H. In G, there are at most n^4 groups, since every group is of size at most 4. Consequently, the construction of the formula from the given graph takes polynomial time.

Due to Lemma 13, in the constructed Sat formula, every clause X contains variables $x_1, ..., x_t$ corresponding to leaf groups and at most two variables a and b corresponding to minimal non-leaf groups. We can safely set $x_2, ..., x_t$ to be false, as we will let x_1 ensure the TRUE value of this clause. Every other clause that contains one of $x_2, ..., x_t$, contains it in the negated form, and hence will be true. x_1 appears in at most two other clauses: $(\bar{x_1} \vee \bar{a})$ and $(\bar{x_1} \vee \bar{b})$. Hence, according to the truth-value of a and b, we can assign TRUE or FALSE to x_1, and clause X will be true. Consequently, we can remove clauses involving all leaf groups, as they are not decisive for the satisfiability of the whole formula. The remaining clauses all have two literals, and hence we have a 2-Sat instance that can be solved in polynomial time.

Hence we have given a polynomial-time reduction from $(1,3)$-Cluster Graph Recognition to 2-Sat, which means that $(1,3)$-cluster graphs can be recognized in polynomial time. □

5 Concluding remarks

We have introduced the (p,q)-Cluster Graph Recognition problem and proved that it is NP-complete. We have shown that $(p,0)$-cluster, $(p,1)$-cluster, $(p,2)$-cluster, $(0,q)$-cluster, and $(1,3)$-cluster graphs can be recognized in polynomial time. In fact, with a careful implementation we believe that $(p,2)$-cluster and $(1,3)$-cluster graphs can be recognized in linear time. Many interesting questions arise from these results. Some of the most obvious are: Is (p,q)-Cluster Graph Recognition FPT when parameterized by either p or q, meaning that there is an algorithm with running time $f(p) \cdot poly(n,q)$ or $f(q) \cdot poly(n,p)$? Is it FPT when parameterized by both p and q, meaning that there is an algorithm with running time $f(p,q) \cdot poly(n)$? Can (p,q)-cluster graphs be recognized in polynomial time for every pair of fixed p and q, meaning

that there is an algorithm with running time $n^{f(p,q)}$? If not, what is the smallest p and smallest q for which it is NP-complete to recognize (p, q)-cluster graphs?

In fact, there is a natural extension of (p, q)-Cluster Graph Recognition to (p, q, k)-Cluster Graph Editing. In the latter, we ask whether a (p, q)-cluster graph can be obtained by adding or removing in total at most k edges in a given graph. Hence $(p, q, 0)$-Cluster Graph Editing is equivalent to (p, q)-Cluster Graph Recognition. The editing version of the problem opens a whole range of questions of the above type involving k in addition.

References

[1] Z. Y. A. Ben-Dor, R.Shamir. Clustering gene expression patterns. *J.Comput. Biol.*, 6(3/4):281–292, 1999.

[2] S. Böcker, S. Briesemeister, Q. A. Bui, and A. Truß. Going weighted: Parameterized algorithms for cluster editing. In *COCOA 2008*, *LNCS* 5165: 1–12, 2008.

[3] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

[4] E. Chesler, L. Lu, S. Shou, Y. Qu, J. Gu, J. Wang, H. Hsu, J. Mountz, N. Baldwin, M. Langston, D. Threadgill, K. Manly, and R. Williams. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nature Genetics*, 37(3):233–242, 2005.

[5] P. Damaschke. Fixed-parameter enumeratbility of cluster editing and related problems. *Theory of Computing Systems*, 2009.

[6] F. Dehne, M. Langston, X. Luo, S. Pitre, P. Shaw, and Y. Zhang. The cluster editing problem: implementations and experiments. In *IWPEC 2006*, *LNCS* 4169: 13–24, 2006.

[7] M. Fellows, J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Graph-based data clustering with overlaps. In *COCOON 2009*, *LNCS* 5609: 516–526, 2009.

[8] H. Frigui and O. Nasraoui. Simultaneous clustering and dynamic key-word weighting for text documents. *M. Berry (ed.), Survey of Text Mining, Springer*, pages 45–70, 2004.

[9] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39:321–347, 2004.

[10] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: fixed-parameter algorithm for clique generation. *Theory of Computing Systems*, 38:373–392, 2005.

[11] J. Guo, I. A. Kanj, C. Komusiewicz, and J. Uhlmann. Editing graphs into disjoint unions of dense clusters. In *ISAAC 2009*, *LNCS*: 583–593, 2009.

[12] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. A more relaxed model for graph-based data clustering: s-plex. In *AAIM 2009*, *LNCS* 5564: 226–239, 2009.

[13] J. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.

[14] A. M. Charikar, V.Guruswami. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71:360–383, 2005.

[15] D. W. R. Xu. Survey of clustering algorithms. *IEEE Transactions onNeural Networks*, 16(3):645–678, 2005.

[16] R. R.Sharan, A. Maron-Katz. Click and expander: a system for clustering and visualizing gene expression data. *Bioinformatics*, 19(14):1787–1799, 2003.

[17] D. Scholtens, M. Vidal, and R. Gentlemand. Local modeling of global interactome networks. *Bioinformatics*, 21:3548?3557, 2005.

[18] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[19] R. L. Z. Wu. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

Appendix

Proof of Observation 4

Proof. Assume that G has a partition into groups $C_1, C_2, \ldots, C_k$. Let let C_i be a group containing at least two vertices, such that every vertex of C_i has degree at most q. In this case each vertex of C_i defines a group consisting only of itself. Furthermore, after dividing C_i into singletons, every other set C_j with $j \neq i$ is still a group. □

Proof of Observation 5

Proof. By Observation 4 we know that there is a partition such that every group C of the partition that contains at least two vertices, contains a high degree vertex u. If u has no neighbors outside of C, since C is a clique we have that $N[u] = C$, and the proof is complete. Assume that u has at least one neighbor outside of C. Assume that every neighbor of u in C has a neighbor outside of C. Then together with the neighbors that u has outside of C, there are more than q edges in G with exactly one endpoint in C, which contradicts the assumption that C is a group. Thus u has a neighbor v in C such that v has no neighbors outside of C. Since C is a clique, $N[v] = C$. □

Proof of Lemma 12

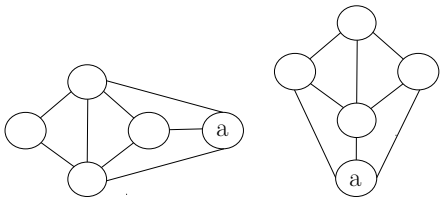
Proof. Let $G = (V, E)$. For Rule 1, clearly since no group contains both u and v, G is a $(1, 3)$-cluster graph if and only if u and v appear in different groups.

For Rule 2, let C be a 1-group of size at least 5, and let u and v be the only pair of non-adjacent vertices of C. If $|C| \geq 6$ then no subset of C is a group since any partition of C into smaller parts results in more than 3 edges with one endpoint in each part. If $|C| = 5$ then every vertex of C, other than u and v, has 4 neighbors in C, hence the only way C can be partitioned into smaller groups is making u or v a group on its own and keeping the rest of C together. In this case, for these two sets to be groups, there cannot be any edges in G with exactly one endpoint in C, since there are already 3 edges between the two sets. Hence if there is a partition of G into groups where C is divided, then there is also a partition where C is kept as a whole. Consequently, since C is maximal, G is a $(1, 3)$-cluster graph if and only if there is a partition into groups where C is one of the groups.

For Rule 3, let C be a clique of size at least 4. Since Rule 2 cannot be applied, C is not part of a 1-group of size at least 5. The only way to partition C into smaller parts is if C is of size 4, a vertex is a group on its own and the rest of C is another group, in which case this is the whole graph, since there are already 3 edges between the two parts. Thus the only way G can be partitioned into groups is if C is one of these groups.

For Rule 4, Let C be a 1-group of size at least 4 such that C has three vertices with neighbors outside of C. We consider 3 cases:

- **Case 1.** (Figure 2(a)) There is a vertex $a \notin C$ adjacent to 3 vertices in C that form a triangle. $C \cup \{a\}$ forms a 4-clique which is a contradiction with the fact that G is a reduced graph.
- **Case 2.** (Figure 2(b)) There is a vertex $a \notin C$ adjacent to 3 vertices in C that induce a path on 3 vertices. Any strict subset of $C \cup \{a\}$ has at least 3 leaving edges. Hence, the



(a) Case 1 (b) Case 2

Figure 2: Two of the cases considered in the proof of correctness of reduction rule 4 of Definition 11.

set C has to be a subset of a group used in the partition.

- **Case 3.** There are at least two vertices outside of C that are adjacent to two different vertices in C. Then for any proper subset $S \subset C$, the number of edges leaving S is at least 2. It follows that if C' is a group and $\emptyset \subset C' \cap C \subset C$ then $C' \setminus C$ is also a group. Hence, if there is a partition of G into groups then there is also a partition such that, for each group C' in the partition, $C \cap C' = \emptyset$ or $C \subseteq C'$.

For the running time claim of the lemma, first notice that the number of connected groups can be enumerated in $\mathcal{O}(m^3 n)$ time: for every connected group C there exist edges $e_1, e_2, e_3 \in E$ and vertex $v \in V$ such that C is the connected component containing v in $(V,\ E \setminus \{e_1, e_2, e_3\})$. For Rule 1, we can restrict ourselves to minimal groups, which are necessarily connected. And for Rule 2, maximal 1-group of size at least 5 also is connected since it misses at most one edge. Hence, by enumerating all connected groups these rules can be applied in polynomial time. For Rules 3 and 4 the same property is trivial. □

Proof of Lemma 13

First we need two definitions. A *triangle* is a set of 3 vertices that induces a complete graph. A *diamond* is a set of 4 vertices that induces a complete graph minus one edge. The proof of Lemma 13 will be based on the following two claims.

Claim 15. *In a reduced graph every minimal non-leaf group is a triangle or a diamond.*

Proof. Let G be a reduced graph and let C be a group in G. (Recall that the degrees are not the actual degrees but penalized degrees, and hence they are the same as in the input graph from which G was reduced.) By Definition 11, $|C| \leq 4$. Let v be a high degree vertex in C. If $C = \{v\}$, there are at least 4 edges leaving C, contradicting that C is a group. If $C = \{v, u\}$ then $N(v) = u$ since any minimal group is connected and there will be too many edges leaving C otherwise. But then C is a leaf group. If $|C| = 3$ and not a triangle, it is a path on 3 vertices since it is connected. If v is one of the endpoints of the path induced by C, there are at least q edges leaving C from v. Thus, no other vertex in C is adjacent to leaving edges and C is a leaf group. If v is the middle vertex of the path induced by C, there are at least $q-1$ edges adjacent to v that leave C. Hence, one of the two other vertices has no edges adjacent to it that leave C. Call such a vertex u. Now $\{u\}$ and $C \setminus \{u\}$ are also groups, and hence C is not a minimal group. □

Claim 16. *In a reduced graph $G = (V, E)$, there do not exist 3 distinct minimal non-leaf groups C_1,C_2,$C_3 \subseteq V$ that satisfy all of the following.*

- *$C_1 \cap C_2 \cap C_3$ contains a high degree vertex v;*
- *for every $1 \leq i \leq 3$, $G[C_i]$ is a triangle or a diamond;*
- *there are at least 2 edges in G with exactly one endpoint in $C_1 \cup C_2 \cup C_3$.*

Proof. We use computer aided case analysis. The program in Listing 1 enumerates all possible graphs $G[C_1 \cup C_2 \cup C_3]$ such that C_1, C_2, and C_3 satisfy the conditions of the claim. Then it checks for each such graph whether C_1 or C_2 or C_3 is a group. If such a group is found the program outputs it. The program does not output any group, and thus the claim is proved. □

Proof of Lemma 13. Assume for contradiction that Lemma 13 is not true. Then there exist distinct minimal non-leaf groups C_1, C_2 and C_3 such that $C_1 \cap C_2 \cap C_3$ contains a high degree vertex v. Since $|C_1 \cup C_2 \cup C_3| > 42$ there are at least two edges in the graph with exactly one endpoint in $C_1 \cup C_2 \cup C_3$. But now Claims 15 and 16 together give us that do not exist such minimal non-leaf groups C_1, C_2 and C_3. Thus we get a contradiction and the lemma follows. □

Listing 1: Computer-aided case analysis

```
public class ClusterFinal {

	// add triangle to M
	static void addTriangle(int[][] M, int a, int b, int c) {
		M[a][b]=M[b][a]=1;
		M[a][c]=M[c][a]=1;
		M[b][c]=M[c][b]=1;
	}

	// add K4\e to M, ad is the non-edge.
	static void addAlmostK4(int[][] M, int a, int b, int c, int d) {
		M[a][b]=M[b][a]=1;
		M[a][c]=M[c][a]=1;
		M[b][c]=M[c][b]=1;
		M[b][d]=M[d][b]=1;
		M[c][d]=M[d][c]=1;
	}


	 // Outputs a group, in a K4' the output is [a, b, c, d], ad is the intended non-edge.
	static String makeString(int t,int a,int b,int c,int d) {
		String s1="", s2="";
		if(t==0) s1 = "[k3,";
		if(t==1) s1 = "[k4',";
		if(t==2) s1 = "[k4',";

		if(t==0) s2 = a + "," + b + "," + c + "]";
		if(t==1) s2 = a + "," + b + "," + c + "," + d +"]";
		if(t==2) s2 = b + "," + a + "," + c + "," + d +"]";

		return s1+s2;
	}

	// Checks that two groups are not equal, assuming (a,b,c) are always different and nonzero.
	static boolean isSame(int t1, int a1, int b1, int c1, int t2, int a2, int b2, int c2) {
		  if(t1 != t2) return false;
		  if(t1 == 0) {
			  if(a1 != a2 && a1 != b2) return false;
			  if(b1 != a2 && b1 != b2) return false;
		  } else {
			  if(a1 != a2 && a1 != b2 && a1 != c2) return false;
			  if(b1 != a2 && b1 != b2 && b1 != c2) return false;
			  if(c1 != a2 && c1 != b2 && c1 != c2) return false;
		  }
		  return true;
	}

	public static void main(String[] args) {
```

```
//enumerate group type. 0 = k3, 1=k4\e with d(v)=2, 2=k4\e with d(v)=3
for(int t1=0; t1 < 3; ++t1) {
for(int t2=0; t2 < 3; ++t2) {
for(int t3=0; t3 < 3; ++t3) {

// Vertices of group 1. v is always vertex 0.
int v11=1;
int v12=2;
int v13=3;   // if t1==0 this vertex is ignored.

// Vertices of group 2. v is vertex 0,
for(int v21 = 1; v21 <= 4; ++v21) {
for(int v22 = 1; v22 <= 5; ++v22) {
        if(v22==v21) continue;
for(int v23 = 1; v23 <= 6; ++v23) {  // if t2==0 this vertex is ignored.
        if(v23==v22 || v23==v21) continue;

// Vertices of group 3. v is vertex 0
for(int v31 = 1; v31 <= 7; ++v31) {
for(int v32 = 1; v32 <= 8; ++v32) {
        if(v32==v31) continue;
for(int v33 = 1; v33 <= 9; ++v33) {  // if t3==0 this vertex is ignored.
        if(v33==v32 || v33==v31) continue;

//----------------- Verify that no two groups are the same.
if(isSame(t1,v11,v12,v13,t2,v21,v22,v23)
   || isSame(t1,v11,v12,v13,t3,v31,v32,v33)
   || isSame(t2,v21,v22,v23,t3,v31,v32,v33)) continue;

//----------------- BUILD G
int[][] M = new int[10][10];

// group 1:
if(t1 == 0) {
      addTriangle(M,0,v11,v12);
} else if(t1 == 1) {
      // Nonedge: 0,v13.
      addAlmostK4(M,0,v11,v12,v13);
} else {
      // Nonedge: v11,v13.
      addAlmostK4(M,v11,0,v12,v13);
}

// group 2:
if(t2 == 0) {
      addTriangle(M,0,v21,v22);
} else if(t2 == 1) {
      // Nonedge: 0,v23.
      addAlmostK4(M,0,v21,v22,v23);
} else {
      // Nonedge: v21,v23.
      addAlmostK4(M,v21,0,v22,v23);
}

// group 3:
if(t3 == 0) {
      addTriangle(M,0,v31,v32);
} else if(t3 == 1) {
      // Nonedge: 0,v33.
      addAlmostK4(M,0,v31,v32,v33);
} else {
      // Nonedge: v31,v33.
      addAlmostK4(M,v31,0,v32,v33);
}

// ---------------------- Check Whether the K4\e's are induced.
if(t1 == 1 && M[0][v13]==1) continue;
if(t1 == 2 && M[v11][v13]==1) continue;

if(t2 == 1 && M[0][v23]==1) continue;
if(t2 == 2 && M[v21][v23]==1) continue;

if(t3 == 1 && M[0][v33]==1) continue;
if(t3 == 2 && M[v31][v33]==1) continue;

// ---------------The graph must have at least 2 edges leaving, guess where the edges are and penalize endponts.
for(int pen1 = 0; pen1 < 10; ++pen1) {
for(int pen2 = 0; pen2 < 10; ++pen2) {

// pen1 and pen2 must be actual vertices
if(pen1 != 0 && pen1 != v11 && pen1 != v12 && (t1 == 0 || pen1 != v13)
                  && pen1 != v21 && pen1 != v22 && (t2 == 0 || pen1 != v23)
                  && pen1 != v31 && pen1 != v32 && (t3 == 0 || pen1 != v33)) continue;

if(pen2 != 0 && pen2 != v11 && pen2 != v12 && (t1 == 0 || pen2 != v13)
                  && pen2 != v21 && pen2 != v22 && (t2 == 0 || pen2 != v23)
                  && pen2 != v31 && pen2 != v32 && (t3 == 0 || pen2 != v33)) continue;
```

```
        int[] pen = new int[10];
        pen[pen1]++;
        pen[pen2]++;

        // ---------------------- Find degree of v.
        int d=pen[0];
        for(int i = 1; i < 10; ++i) {d += M[0][i];}

        // --- v must have high degree, penalize until it does.
        while(d < 4) {d++; pen[0]++;}

        // ------------------------ Find ## edges leaving group 1, 2, 3.
        int cut1=0,cut2=0,cut3=0;
        for(int i = 0; i < 10; ++i) {
            for(int j = 0; j < 10; ++j) {
                if(j==i) continue;
                if(M[i][j]==1) {
                    if(t1==0){
                        cut1 += (i==0 || i==v11 || i==v12) && (j != 0 && j != v11 && j != v12) ? 1 : 0;
                    } else {
                        cut1 += (i==0 || i==v11 || i==v12 || i == v13)
                                                    && (j != 0 && j != v11 && j != v12 && j != v13) ? 1 : 0;
                    }

                    if(t2==0){
                        cut2 += (i==0 || i==v21 || i==v22) && (j != 0 && j != v21 && j != v22) ? 1 : 0;
                    } else {
                        cut2 += (i==0 || i==v21 || i==v22 || i == v23)
                                                    && (j != 0 && j != v21 && j != v22 && j != v23) ? 1 : 0;
                    }

                    if(t3==0){
                        cut3 += (i==0 || i==v31 || i==v32) && (j != 0 && j != v31 && j != v32) ? 1 : 0;
                    } else {
                        cut3 += (i==0 || i==v31 || i==v32 || i == v33)
                                                    && (j != 0 && j != v31 && j != v32 && j != v33) ? 1 : 0;
                    }
                }
            }
        }

        // If any cut + penalty is too big, not a counterexample.
        if(cut1 + pen[0] + pen[v11] + pen[v12] + (t1 > 0 ? pen[v13] : 0) > 3) continue;
        if(cut2 + pen[0] + pen[v21] + pen[v22] + (t2 > 0 ? pen[v23] : 0) > 3) continue;
        if(cut3 + pen[0] + pen[v31] + pen[v32] + (t3 > 0 ? pen[v33] : 0) > 3) continue;


        // ----- OUTPUT GRAPH!
        System.out.println(makeString(t1,0,v11,v12,v13) + ","
                           + makeString(t2,0,v21,v22,v23) + ","
                           + makeString(t3,0,v31,v32,v33));

        }}  // Penalties
        }}} // Group 3
        }}} // Group 2
        }}} // Outer loop

        } // end of main method
}
```