Proc. 5th TAMC, LNCS 4978, pp. 445-456, 2008

Improved Algorithms for Bicluster Editing

Jiong Guo[1,⋆], Falk Hüffner[1,⋆], Christian Komusiewicz[1,⋆⋆], and Yong Zhang[2]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{guo,hueffner,ckomus}@minet.uni-jena.de
[2] Department of Mathematical Sciences, Eastern Mennonite University
Harrisonburg, VA 22802, USA
yong.zhang@emu.edu

Abstract. The NP-hard Bicluster Editing is to add or remove at most k edges to make a bipartite graph $G = (V, E)$ a vertex-disjoint union of complete bipartite subgraphs. It has applications in the analysis of gene expression data. We show that by polynomial-time preprocessing, one can shrink a problem instance to one with $4k$ vertices, thus proving that the problem has a linear kernel, improving a quadratic kernel result. We further give a search tree algorithm that improves the running time bound from the trivial $O(4^k + |E|)$ to $O(3.24^k + |E|)$. Finally, we give a randomized 4-approximation, improving a known approximation with factor 11.

1 Introduction

Data clustering is a classical task, where the goal is to partition a data set into *clusters* such that elements within a cluster are similar, while between clusters there is less similarity. This similarity is often modeled as a graph: Each vertex represents a data point, and two vertices are connected by an edge iff the entities that they represent have some (context-specific) similarity. If the data were perfectly clustered, this would result in a *cluster graph*, that is, a graph where every connected component is a clique. However, for real-world data, there is typically noise in the data. A simple clustering model is then the Cluster Editing problem [4, 19]: find a minimum set of edges to add or delete to make the graph a cluster graph.

Cluster Editing is NP-hard [15]; a number of approaches have been recently suggested to deal with this. After a series of improvements, the best known polynomial-time approximation is by a factor of 2.5 [2, 21]. Another technique is that of *fixed-parameter (FPT) algorithms* [7, 9, 17]. The idea is to accept the superpolynomial running time that seems to be inherent to NP-hard problems, but to restrict the combinatorial explosion to a *parameter* that is expected to

⋆ Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.
⋆⋆ Supported by a PhD fellowship of the Carl-Zeiss-Stiftung.

be small. For Cluster Editing, the number of editing operations k is a suitable parameter, since for data with not too much noise it should be low. Several fixed-parameter algorithms for Cluster Editing have been suggested (see also Hüffner et al. [14] for a survey on FPT techniques in graph-modeled clustering). The search tree algorithm of Gramm et al. [10] with a running time bound of $O(2.27^k + n^3)$ has been experimentally evaluated [6]. A recent manuscript [5] claims a running time of $O(1.83^k + n^3)$ by using a different branching strategy and reports further experimental results.

An important tool of FPT algorithmics is *kernelization* [7, 9, 17]. A kernelization is a polynomial-time preprocessing that reduces an instance to a size that depends only on the parameter k, and not on the input size $|G|$ anymore. Clearly, such a preprocessing is useful for basically any approach to solving the problem, be it exact, approximative, or heuristic. For Cluster Editing, after a series of improvements [8, 10, 18], a kernel of only $4k$ vertices is known [11].

In some settings, the standard clustering model is not satisfactory. An important example is clustering of gene expression data, where under a number of conditions the level of expression of a number of genes is measured. This yields a bipartite similarity graph. Here, clustering only genes or only conditions often does not yield sufficient insight; we would like to find subsets of genes and subsets of conditions that together behave in a consistent way. This is called *biclustering* [16, 20]. A simple formulation of biclustering analogous to Cluster Editing is Bicluster Editing. Here, as a consistency condition for a cluster, we demand that it forms a *biclique*, that is, a complete bipartite subgraph. With bipartite graphs, we mean two-colorable graphs. Further, we do not allow any clusters to overlap.

> Bicluster Editing
> **Instance:** A bipartite graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Can we delete and add at most k edges in G such that it becomes a *bicluster graph*, that is, a graph where every connected component is a biclique?

Further applications of biclustering arise in collaborative filtering, information retrieval, and data mining. Despite its importance, there are fewer results for Bicluster Editing than for Cluster Editing. Amit [3] proved the NP-hardness and gave a factor-11 approximation based on the relaxation of a linear program. Using a simple branching strategy, the problem can be solved in $O(4^k + m)$ time [18], where m is the number of edges in the graph. Protti et al. [18] showed how to construct a problem kernel with $4k^2 + 6k$ vertices.

Contributions. Following the work recently done for Cluster Editing, our aim is to improve FPT and approximation algorithms also for its sister problem Bicluster Editing. We first improve the size of the problem kernel from $4k^2+6k$ to $4k$ vertices (Sect. 2). The methods used are similar to those of Guo [11]. If the input graph is not already a bipartite graph, we can still get a $6k$-vertex kernel by similar means. Next, we show that the trivial $O(4^k + m)$ time branching

algorithm can be improved to $O(3.24^k + m)$ time by a more refined branching strategy (Sect. 3). Finally, we give a randomized approximation algorithm with an expected approximation factor of 4, using similar techniques as those introduced for Cluster Editing [1].

Preliminaries. We consider only undirected graphs $G = (V, E)$ with $n := |V|$ and $m := |E|$. Since singleton vertices do not play an interesting role in our problems, we assume that $n \in O(m)$. Let P_4 denote an induced path comprising 4 vertices. Furthermore, let $ijkl$ denote a P_4 in which i and l have degree 1 and j and k have degree 2. The neighborhood of a vertex v is denoted by $N(v)$, and the closed neighborhood $N(v) \cup \{v\}$ is denoted by $N[v]$. We furthermore extend this notation to vertex sets, that is, for a vertex set S, $N(S) := (\bigcup_{v \in S} N(v)) \setminus S$. For a vertex v, $N_2(v) := N(N(v)) \setminus \{v\}$ denotes the set of vertices that have distance exactly 2 from v.

Due to lack of space, several proofs are deferred to a full version of this paper.

2 Linear Problem Kernel

In this section, we present a kernelization algorithm for Bicluster Editing that produces a kernel consisting of at most $4k$ vertices, improving the kernel consisting of $O(k^2)$ vertices given by Protti et al. [18]. This kernelization follows the idea of the kernelization algorithm for Cluster Editing in [11] that also produces a kernel consisting of at most $4k$ vertices. However, since here we are dealing with bipartite graphs and bicliques, the concrete handling of the data reduction rules and the argumentation of the kernel size are different from the one for Cluster Editing. The first step is to introduce a useful structure.

Definition 1. *A set S of vertices is called a* critical independent set *if all vertices in S have the same open neighborhood and S is maximal under this property.*

Observe that every critical independent set is an independent set. The connection between critical independent sets and Bicluster Editing is given by the following lemma.

Lemma 1. *For any critical independent set I, there is an optimal solution of* Bicluster Editing *in which any two vertices v_1 and v_2 from I end up in the same biclique.*

We apply the following two data reduction rules; the second one works on critical independent sets.

Rule 1. Remove all connected components that are bicliques from the graph.

Rule 2. Consider a critical independent set R. Let $S := N(R)$ and $T := N(S) \setminus R$. If $|R| > |T|$, then remove arbitrary vertices from R until $|R| = |T|$.

Rule 1 is clearly correct and can be carried out in $O(m)$ time. A situation in which Rule 2 can be applied is illustrated in Fig. 1. Next, we prove the correctness of Rule 2.

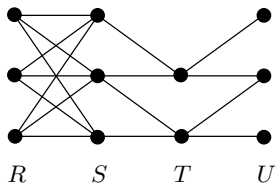


Fig. 1: Example for the application of Rule 2.

Lemma 2. *Rule 2 is correct and works in $O(n^2)$ time.*

Proof. To prove the correctness of Rule 2, we first claim that, as long as $|R| \geq |T|$, there is always an optimal solution constructing a bicluster graph that contains a biclique B with $R \cup S \subseteq B \subseteq R \cup S \cup T$ and, thus, deleting or inserting no edge incident to R. Since the input graph G and the graph resulting by one application of Rule 2 to G differ only in the size of R, the correctness of Rule 2 follows.

To show the claim, let $U := N(T) \setminus S$. First, observe that R will not be "split" (following from Lemma 1), that is, there is always an optimal solution leaving a biclique B with $R \subseteq B$. Second, we prove that no vertex outside of $R \cup S \cup T$ can be in B, that is, $B \subseteq R \cup S \cup T$. To see this, if a vertex $u \notin R \cup S \cup T$ is in B, then obviously u is from U. However, to add u to B needs at least $|R|$ edge insertions. Thus, as long as $|R| \geq |T|$, adding u to B is never better than putting u in a biclique different from B, which requires at most $|T|$ edge deletions. Finally, we show that no vertex from S can be outside of B, that is, $R \cup S \subseteq B$. This is easy to see, since every vertex $u \in S$ has only neighbors in R and T. By $|R| \geq |T|$ and $B \subseteq R \cup S \cup T$, including u in B requires at most $|T|$ edge modifications, namely, deleting all edges between u and $N(u) \cap T$ and adding edges between u and $T \setminus N(u)$. In comparison, excluding u from B needs at least $|R|$ edge deletions, since $R \subseteq N(u)$.

Concerning the running time, one can compute all critical independent sets in $O(m)$ time [12]. Then, we determine the sets S and T for all independent sets R, which can be done in $O(n^2)$ time. To check the applicability of Rule 2, one iterates over all critical independent sets and uses the already computed information about S and T to decide if the precondition of Rule 2 is fulfilled by R. Note that after one application of Rule 2, one has only to consider the critical independent sets whose vertices are in S and T and to change the sizes of their S's and T's. Therefore, each application of Rule 2 can be carried out in $O(n)$ time. Rule 2 can be applied at most n times, which gives the total running time $O(n^2)$. □

With these two rules we can now prove a kernel consisting of at most $4k$ vertices.

Theorem 1. Bicluster Editing *on bipartite graphs admits a $4k$-vertex problem kernel.*

Proof. Let G denote a bipartite graph on which Rules 1 and 2 have been exhaustively applied. Furthermore, let F be a bicluster editing set with $|F| \leq k$ and let G' be the resulting bicluster graph after applying the edge modifications in F to G. We partition the vertices in G' into two sets, X containing the endpoints of the edges in F, and Y the rest. Clearly, $|X| \leq 2k$. It remains to upper-bound $|Y|$. Suppose G' consists of l biclusters, $B_1, \ldots, B_l$. It is easy to see that for every $i \in \{1, \ldots, l\}$, the unaffected vertices from one partition in B_i must have the same neighborhood in G. Hence, for every $i \in \{1, \ldots, l\}$, the vertices in $B_i \cap Y$ form at most two critical independent sets in G. Let R be one critical independent set in B_i, $S := N_G(R)$, and $T := N_G(S) \setminus R$. Due to Rule 2, $|R| \leq |T|$. Due to Rule 1, all vertices of T are in X. Some of them are in B_i after gaining some edges between them and the vertices in S and the others are in other bicliques after losing all edges between them and S. Therefore, summing up over all critical independent sets in all bicliques, we can conclude that $|Y| \leq |X|$, giving the claimed number of vertices in the kernel. □

Protti et al. [18] have considered Bicluster Editing on general graphs as well and presented a problem kernel with $O(k^2)$ vertices. With a slight modification of Rule 2, we can improve this result to a $6k$-vertex problem kernel. The main difference to the kernelization for bipartite graphs lies in the edges between the vertices of S: If the vertices in S have much more edges between them than the size of R, it could be better to keep them and to remove some edges between R and S. To take this into account, we make a partition of the vertices in S as described below.

Modified Rule 2. Consider a critical independent set R, let $S := N(R)$ and $T := N(S) \setminus R$. Further partition S into two sets, S_1 the set of vertices without neighbors in S and $S_2 := S \setminus S_1$. If $|R| > |S_2| + |T|$, then reduce R until $|R| = |S_2| + |T|$.

The correctness proof of the modified Rule 2 is almost the same as the one for Lemma 2, namely, showing that in case $|R| \geq |S_2| + |T|$ there is always an optimal solution creating a biclique B, such that $R \cup S \subseteq B \subseteq R \cup S \cup T$. The only difference concerns the vertices in S_2. Since they have neighbors in S_2, including them in B requires not only deleting and adding edges between them and T but also deleting edges between them and their neighbors in S_2. However, if $|R| \geq |S_2| + |T|$, then it is never better to exclude them from B than to include them in B.

Theorem 2. Bicluster Editing *on general graphs admits a* $6k$*-vertex problem kernel.*

3 Fixed-Parameter Algorithm

In this section, we present a search tree algorithm for Bicluster Editing in bipartite graphs that is based on the forbidden subgraph characterization of

Bicluster Editing and has a running time of $O(3.24^k + m)$, improving upon the trivial search tree algorithm with a running time of $O(4^k + m)$ [18].

Let $ijkl$ be a P_4 in G. The trivial search tree algorithm for Bicluster Editing branches on $ijkl$ in 4 cases: one case corresponds to adding edge $\{i, l\}$; the other three cases correspond to removing one of the three edges of $ijkl$. The improvement of the running time of our algorithm is achieved by applying a refined branching strategy on larger subgraphs that contain a P_4. In this branching strategy, we distinguish two main cases. For the first case, we show that an improved branching can be achieved. For the second case, we show that it can be solved in polynomial time. In the following, we describe this branching strategy.

Clearly, branching is only performed as long as G is not a bicluster graph. Therefore, we assume that G contains a P_4. Furthermore, we deal with each connected component separately. Therefore, without loss of generality assume that G is connected. We distinguish two main cases.

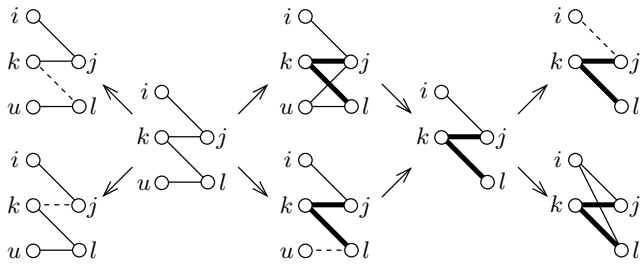
Case 1: There is a connected subgraph of size 5 *of* G *that contains a* P_4 *and has 4 edges.* Let G' be such a subgraph, and let $ijkl$ denote a P_4 contained in G'. Since G' is connected and contains 5 vertices and 4 edges, it must contain a vertex u that is connected to exactly one vertex in $ijkl$. Hence, the resulting graph is either a P_5—in case u is adjacent to i or l—or a so-called *fork*—in case u is adjacent to j or k. We describe the branching strategy for P_5's in detail. Branching on forks works analogously. Both branchings are depicted in Fig. 2.

Let $ijklu$ be the P_5 that we branch on. In the first branch, we delete the edge $\{k, l\}$, the parameter is decreased by 1. In the second branch, we delete the edge $\{j, k\}$ and the parameter is decreased by 1. Since we have already considered deleting $\{k, l\}$ or $\{j, k\}$, we can mark these two edges as *permanent*, that is, we may not delete these edges in the remaining branches. To destroy $jklu$, we must either delete $\{l, u\}$ or add $\{j, u\}$. But after performing either of these two edge modifications the graph still contains $ijkl$, and $\{j, k\}$ and $\{k, l\}$ are marked as permanent. Hence, for each of these two branches, we create two subbranches, one in which $\{i, l\}$ is added, and one in which $\{i, j\}$ is deleted. In total, we have 6 branches, two branches in which the parameter is decreased by 1, and 4 branches in which the parameter is decreased by 2. To estimate the size of the search tree, we use the concept of *branching vectors* [17]. The branching vector of this branching is $(1, 1, 2, 2, 2, 2)$. The branching on a fork works analogously.

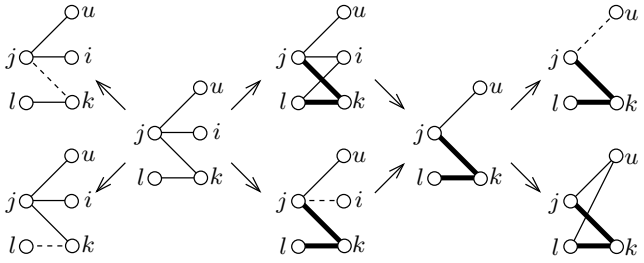
Case 2: Otherwise. Since Case 1 did not apply, every connected subgraph of size 5 that contains a P_4 has at least 5 edges. We show that in this case, no branching is needed because G can be turned into a biclique by adding one edge.

Lemma 3. *Let* $G = (V_1, V_2, E)$ *be a fork-free and* P_5*-free connected bipartite graph, and let* $ijkl$ *be a* P_4 *in* G*. Then, adding edge* $\{i, l\}$ *transforms* G *into a biclique.*

Proof. W.l.o.g. assume that $\{i, k\} \subseteq V_1$ and $\{j, l\} \subseteq V_2$. We prove the lemma by showing that with the exception of $\{i, l\}$ all edges are present in G.



(a) Case 1.1: Branching on a P_5.



(b) Case 1.2: Branching on a fork.

Fig. 2: Branching on subgraphs of size 5 that contain a P_4 and exactly 4 edges. *Dashed lines* are deleted edges; *bold lines* are permanent edges.

First, we show that k is adjacent to all vertices in V_1, and that j is adjacent to all vertices in V_2. Clearly, if i has a neighbor u, then u is a neighbor of k. Otherwise, $G[\{i, j, k, l, u\}]$ is a P_5, and G is thus not P_5-free. Also, every vertex $u \in N(k) \setminus \{l\}$ is a neighbor of i. Otherwise, $G[\{i, j, k, l, u\}]$ is a fork, and G thus not fork-free. Therefore $N(i) = N(k) \setminus \{l\}$. Analogously, we can show that $N(l) = N(j) \setminus \{i\}$. Furthermore, every vertex $v \in N_2(i)$ is adjacent to j. Otherwise, suppose that there is a vertex $v \in N_2(i)$ that is not adjacent to j and let u be the common neighbor of i and v. Then, the subgraph $G[\{i, k, l, u, v\}]$ is a fork, because u is also adjacent to k (since $u \in N(i) \subset N(k)$), and u is not adjacent to l (since $u \notin N(j) \supset N(l)$). Hence, G is not fork-free in this case. Analogously, we can show that k is adjacent to all vertices in $N_2(l)$. With this it becomes obvious that k is adjacent to all vertices in V_1, and j is adjacent to all vertices in V_2.

Now we show that every pair of vertices $u \in V_1 \setminus \{i\}$ and $v \in V_2 \setminus \{l\}$ must be pairwise adjacent. Suppose, there are two vertices $u \in V_1 \setminus \{i\}$ and $v \in V_2 \setminus \{l\}$ that are not pairwise adjacent. Since i is adjacent to all vertices in $V_1 \setminus \{l\}$, it is adjacent to v. Analogously, one can show that j and l are adjacent to u. Therefore, $G[\{i, j, l, v, u\}]$ is a P_5 if v and u are not adjacent, contradicting the fact that G is P_5-free.

Since all vertices in $V_1 \setminus \{i\}$ are adjacent to all vertices $V_2 \setminus \{l\}$, it is clear that adding edge $\{i, l\}$ transforms G into a biclique. □

In the following theorem, we bound the running time of the described search tree algorithm, when it is combined with kernelization.

Theorem 3. Bicluster Editing *can be solved in* $O(3.24^k + m)$ *time.*

4 Randomized 4-Approximation Algorithm

We present a polynomial time randomized factor-4 approximation algorithm for Bicluster Editing that is based on a technique introduced by Ailon et al. [1]. This improves the previously best factor-11 approximation algorithm by Amit [3]. The basic strategy of the algorithm is to randomly pick a pivot vertex v, and then to randomly destroy all P_4's that contain v. In doing so, we create an isolated biclique that contains v, since a connected component in which one vertex does not appear in a P_4 is a biclique. This procedure is applied until the graph is a bicluster graph. In the following, we describe how the P_4's containing the pivot vertex v are destroyed.

Given a pivot vertex i, we create a vertex set C that initially contains $N[i]$. In the end this set C contains the vertices that are in the same biclique as i in the final bicluster graph. First, we add all vertices that are in the same critical independent set as i.

Then we randomly decide for each vertex w that is adjacent to at least one vertex of $N(i)$ whether w should be added to C. Since w is adjacent to neighbors of $N(i)$ but is not in the same critical independent set as i, there must be a P_4 that contains i and w. By randomly deciding whether i and w end up in the same biclique, we randomly decide which edge modification is made to destroy the P_4. After this is done for all such vertices, we output C and cut C from G.

This is done until G has no vertex. The pseudo-code of the algorithm is shown in Fig. 3. In order to apply the method of Ailon et al. [1], the algorithm must guarantee that after an edit operation is made on an edge, this edge is never again modified during the course of the algorithm, and that for a given P_4 each edit operation has the same probability. In our case this probability is $\frac{1}{4}$, which leads to an approximation factor of 4.

To prove this upper bound on the approximation factor, we first need the notion of a *fractional packing*.

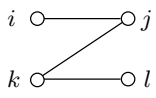
Definition 2. *Let* $G = (V_1, V_2, E)$ *be a bipartite graph,* P *the set of* P_4*'s of* G*, and* $w : P \to \mathbb{R}^+$ *a weight function. The function* w *is called a* fractional packing *of* P *if* $\forall i \in V_1, j \in V_2 : \sum_{\{p \in P | \{i,j\} \in p\}} w(p) \leq 1$.

In the following lemma, we show that given a fractional packing w, the sum of the weights $w(p)$ of all $p \in P$ is a lower bound on the cost of optimal solutions.

Lemma 4. *Let* $G = (V_1, V_2, E)$ *be a bipartite graph,* C_{Opt} *the cost of an optimal solution of* Bicluster Editing *of* G*, and* P *the set of* P_4*'s. If a weight function* $w : P \to \mathbb{R}^+$ *is a fractional packing of* P*, then* $\sum_{p \in P} w(p) \leq C_{\text{Opt}}$.

```
ApproxBicluster(G = (V1, V2, E))
1   G' := (∅, ∅, ∅)
2   while V1 ∪ V2 ≠ ∅:
3         randomly select a pivot vertex i ∈ V1 ∪ V2
4         C := {i} ∪ N(i)
5         for all j ∈ {v ≠ i | N(v) ∩ N(i) ≠ ∅} :
6               if N(j) = N(i) : add j to C
7               else : add j to C with probability 1/2
8         transform G[C] into an isolated biclique
9         G' := G' ∪ G[C]        ▷ add G[C] as new component to G'
10        G := G[V \ C]          ▷ remove C from G
11  output set of edge modifications from G to G'
```

Fig. 3: A randomized factor-4 approximation algorithm for Bicluster Editing.



pivot	$Pr[E_{ij}]$	$Pr[E_{il}]$	$Pr[E_{kj}]$	$Pr[E_{kl}]$
i	0	0	$\frac{1}{2}$	$\frac{1}{2}$
j	0	$\frac{1}{2}$	0	$\frac{1}{2}$
k	$\frac{1}{2}$	$\frac{1}{2}$	0	0
l	$\frac{1}{2}$	0	$\frac{1}{2}$	0
$i \vee j \vee k \vee l$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

Fig. 4: The probabilities of edge modifications in $ijkl$ in case one of $\{i, j, k, l\}$ is chosen as pivot. The event that there is an edge modification between two vertices i and j is denoted by E_{ij}.

Using Lemma 4, we can show an upper bound on the approximation factor of ApproxBicluster, as follows. First, we show that the sum of the probabilities of making edge modifications in all P_4's caused by choosing one of their vertices as pivot equals the expected cost of the solution that is output by ApproxBicluster. Then, we show that dividing these probabilities by 4 also yields a fractional packing and thus that the expected cost of the output solution is at most 4 times the cost of an optimal solution.

Theorem 4. ApproxBicluster *is a randomized factor-4 approximation algorithm for* Bicluster Editing*, running in* $O(n^2)$ *time.*

Proof. Obviously, the output of ApproxBicluster is a solution of Bicluster Editing. We thus prove the theorem by bounding the approximation factor of the expected cost of the output of ApproxBicluster. Let C be the cost of a solution that is output by ApproxBicluster, and let C_{Opt} be the cost of an optimal solution. We prove the theorem by showing that the expected cost $E[C] \leq 4 \cdot C_{\text{Opt}}$.

Clearly, the algorithm inserts or removes edges only between vertices that appear in a P_4. An edit operation is performed in a P_4 $ijkl$ only if i, j, k, and l are still in the graph and one of them is chosen as pivot. Let A_{ijkl} denote the event that one vertex in $\{i, j, k, l\}$ is chosen as pivot when all of them are still in the graph. Furthermore, let π_{ijkl} denote the probability that event A_{ijkl} occurs during the execution of ApproxBicluster. In case that event A_{ijkl} occurs, we perform exactly one edge edit operation between the vertices of $\{i, j, k, l\}$. Therefore, the expected cost $E[C]$ of ApproxBicluster is $\sum_{p \in P} \pi_p$.

We complete the proof by first showing that the weight function w that is obtained by assigning the weight $\frac{\pi_{ijkl}}{4}$ to the P_4 $ijkl$ is a fractional packing of the P_4's of G, and then showing that this leads to the claimed expected approximation factor.

Let p be a P_4, and let $i \in V_1$, $j \in V_2$ be two vertices in p. Let E_{ij} denote the event that there is an edit operation between i and j.

As Fig. 4 shows, the probability $Pr[E_{ij} \mid A_p] = \frac{1}{4}$. Therefore,

$$Pr[E_{ij} \wedge A_p] = Pr[E_{ij} \mid A_p] \cdot Pr[A_p] = \frac{1}{4}\pi_p.$$

Furthermore, note that after event E_{ij} occurred, at least one of i and j is removed from the graph, and thus no further editing between i and j takes place. Therefore, for distinct P_4's p and p', the events $E_{ij} \wedge A_p$ and $E_{ij} \wedge A_{p'}$ are disjoint and thus

$$\sum_{\{p \in P \mid \{i,j\} \in p\}} Pr[E_{ij} \wedge A_p] = \sum_{\{p \in P \mid \{i,j\} \in p\}} \frac{1}{4}\pi_p \leq 1.$$

With this, it becomes obvious that assigning the weight $\frac{1}{4}\pi_p$ to every $p \in P$ results in a fractional packing of the P_4's of G. As shown by Lemma 4, this means that $\sum_{p \in P} \frac{1}{4}\pi_p \leq C_{\text{Opt}}$. Therefore,

$$E[C] = \sum_{p \in P} \pi_p \leq 4 \cdot C_{\text{Opt}},$$

which proves the upper bound on the approximation factor.

Now we prove the running time of the algorithm. In a preprocessing step, we compute the critical independent sets of the graph, which can be performed in $O(n+m)$ time [12]. This is done so that the test in line 6 of the algorithm can be performed in constant time. Determining which vertices end up in C takes $O(m)$ time overall, since the test for membership in the same critical independent set can now be performed in constant time and each edge is visited at most once: either the edge is cut or it is part of the isolated biclique that is removed from G and added to G'. Finally, the number of added edges is in $O(n^2)$, which results in the claimed running time bound. □

Note that the running time of Theorem 4 can be improved to $O(m)$ when the output is merely a list of the bicliques and the vertices they contain. Otherwise, a linear running time cannot be achieved, because the output size cannot be bounded by $O(m)$.

5 Outlook

We have improved kernelization, parameterized algorithm, and approximation algorithm for Bicluster Editing. It is probably possible to further improve the bound of the FPT algorithm, albeit only at the cost of a more complicated case distinction. Further improvement of the approximation factor and derandomization of the result of Theorem 4 should be possible using similar techniques as for Cluster Editing [1, 21, 22].

Together, the improvements make non-heuristic algorithm implementations much more feasible. In particular useful seems the kernelization, which for example is guaranteed to reduce an instance with 1000 vertices and $k = 50$ to only 200 vertices, without losing optimality (note that here the quadratic kernelization [18] does not give any useful bound). It is conceivable that in many cases the kernelized instance can be solved by the branching algorithm from Sect. 3 within reasonable time. Further, it would be interesting to see whether the observed approximation quality of the approximation algorithm from Sect. 4 improves by the preprocessing.

Variants of Bicluster Editing are also of interest, for example considering weights, allowing the deletion of vertices instead of adding and deleting edges, or the generation of a prespecified number of bicliques (the corresponding variations for Cluster Editing have received some attention, see e. g. [11, 13]).

References

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *Proc. 37th STOC*, pages 684–693. ACM Press, 2005. 3, 8, 11

[2] N. Ailon, M. Charikar, and A. Newman. Proofs of conjectures in "Aggregating inconsistent information: Ranking and clustering". Technical Report TR-719-05, Department of Computer Science, Princeton University, 2005. 1

[3] N. Amit. The bicluster graph editing problem. Master's thesis, Tel Aviv University, School of Mathematical Sciences, 2004. 2, 8

[4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1–3):89–113, 2004. 1

[5] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. PEACE: Parameterized and exact algorithms for cluster editing. Manuscript, Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Sept. 2007. 2

[6] F. K. H. A. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw, and Y. Zhang. The cluster editing problem: Implementations and experiments. In *Proc. 2nd IWPEC*, volume 4169 of *LNCS*, pages 13–24. Springer, 2006. 2

[7] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. 1, 2

[8] M. R. Fellows, M. A. Langston, F. A. Rosamond, and P. Shaw. Efficient parameterized preprocessing for cluster editing. In *Proc. 16th FCT*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007. 2

[9] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. 1, 2

[10] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005. 2

[11] J. Guo. A more effective linear kernelization for Cluster Editing. In *Proc. 1st ESCAPE*, volume 4614 of *LNCS*, pages 36–47. Springer, 2007. 2, 3, 11

[12] W. Hsu and T. Ma. Substitution decomposition on chordal graphs and applications. In *Proc. 2nd International Symposium on Algorithms*, volume 557 of *LNCS*, pages 52–60. Springer, 1991. 4, 10

[13] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. In *Proc. 8th LATIN*, volume 4598 of *LNCS*, pages 711–722. Springer, 2008. 11

[14] F. Hüffner, R. Niedermeier, and S. Wernicke. Fixed-parameter algorithms for graph-modeled data clustering. In *Clustering Challenges in Biological Networks*. World Scientific, 2008. To appear. 2

[15] M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986. 1

[16] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004. doi: 10.1109/TCBB.2004.2. 2

[17] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. 1, 2, 6

[18] F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized bicluster editing. In *Proc. 2nd IWPEC*, volume 4169 of *LNCS*, pages 1–12. Springer, 2006. To appear under the title "Applying modular decomposition to parameterized cluster editing problems" in *Theory of Computing Systems*. 2, 3, 5, 6, 11

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

[20] A. Tanay, R. Sharan, and R. Shamir. Biclustering algorithms: A survey. In S. Aluru, editor, *Handbook of Computational Molecular Biology*, pages 26–1 – 26–17. Chapman Hall/CRC Press, 2006. 2

[21] A. van Zuylen and D. P. Williamson. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In *Proc. 5th WAOA*, LNCS. Springer, 2007. To appear. 1, 11

[22] A. van Zuylen, R. Hegde, K. Jain, and D. P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. In *Proc. 18th SODA*, pages 405–414. SIAM, 2007. 11