A fixed-parameter approach for Weighted Cluster Editing

S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß

Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany, E-mail:{*boecker, m2brse, bui, truss*}*@minet.uni-jena.de*

Clustering objects with respect to a given similarity or distance measure is a problem often encountered in computational biology. Several well-known clustering algorithms are based on transforming the input matrix into a weighted graph although the resulting WEIGHTED CLUSTER EDITING problem is computationally hard: here, we transform the input graph into a disjoint union of cliques such that the sum of weights of all modified edges is minimized.
We present fixed-parameter algorithms for this problem which guarantee to find an optimal solution in provable worst-case running time. We introduce a new data reduction operation (merging vertices) that has no counterpart in the unweighted case and strongly cuts down running times in practice. We have applied our algorithms to both artificial and biological data. Despite the complexity of the problem, our method often allows exact computation of optimal solutions in reasonable running time.

1. Introduction

Suppose we are given an undirected graph $G = (V, E)$, and for every tuple $\{u, v\} \in \binom{V}{2}$ we are also given a *weight* that tells us the cost of deleting $\{u, v\}$ from G in case $\{u, v\} \in E$, or inserting $\{u, v\}$ into G in case $\{u, v\} \notin E$. Now the WEIGHTED CLUSTER EDITING problem is defined as follows: Transform G into a *transitive* graph, that is, a disjoint union of cliques, by applying a set of edge modifications with minimum total weight.

The problem of clustering objects according to given similarity or distance values can be found in many areas of computational biology, such as finding families of orthologous genes,[1] or analyzing gene expression data for tissue classification.[2,3] We want to partition these objects into homogeneous and well-separated subsets. In graph theoretical terms, the objects to be clustered are the vertices V of the graph, and a clustering is a disjoint union of cliques. We can transform our input matrix into a weighted graph using a simple threshold, or a stochastic model and log odds.[2,4] For perfect data, the resulting graph is a disjoint union of cliques. But for biological data the input graph is "corrupted", and we have to clean (edit) this graph under the parsimony criterion to reconstruct the best clustering.[2,5] Unlike other clustering techniques, WEIGHTED CLUSTER EDITING does not make any prior assumptions on the number of clusters or their structure.

The WEIGHTED CLUSTER EDITING problem is NP-hard because it generalizes the unweighted case.[6] In recent years, many heuristics were developed for this problem. In particular, the well-known clustering algorithms CAST, HCS, and CLICK build on its graph-theoretic intuition: CAST[4] tries to find the optimal solution with high probability, while both HCS[7] and CLICK[2] greedily use minimal cuts to find an approximate solution. In a recent article[8] we have compared our simple branching strategy (Sec. 4) with two heuristic approaches on biological and random data. To the best of our knowledge, this is the first time a fixed-parameter approach for WEIGHTED CLUSTER EDITING has been proposed.

There exist a variety of results regarding the unweighted CLUSTER EDITING problem.[5] A fixed-parameter algorithm runs in time $O(2.27^k + |V|^3)$ where k is the minimum number of edge modifications.[9] In theory, the best algorithm known for the problem has running time $O(1.92^k + |V|^3)$,[10] but this algorithm uses very complicated branching rules and has never been implemented. See Niedermeier[11] for a recent monograph on fixed-parameter algorithms.

Our contributions. In this paper, we present data reduction rules for WEIGHTED CLUSTER EDITING that are nontrivial extensions of the unweighted case. In particular, we present a new data reduction technique of "merging vertices" that has no counterpart for unweighted graphs. This technique drastically improves the running time of our algorithms in practice. We also show that our data reduction leads to a problem kernel of size $O(k^2)$.

We then adopt the $O(3^k)$ branching strategy from Gramm *et al.*[9] for WEIGHTED CLUSTER EDITING: the resulting algorithm runs in time $O(3^k + |V|^3 \log |V|)$ if every edge deletion or insertion has cost at least 1. Furthermore, we provide a strategy with running time $O(2.42^k + |V|^3 \log |V|)$, roughly following the refined branching strategy in Gramm *et al.*[9] Given an arbitrary instance of the problem with fixed k, these algorithms are guaranteed to find an optimal solution with cost at most k or return that no such solution exists. Minimum edit costs are only required to achieve a provable running time.

We have applied both branching strategies to biological and simulated graph instances. We found that without merging vertices, the $O(2.42^k)$ strategy outperforms the $O(3^k)$ strategy, as expected. But if we merge vertices, both strategies become significantly faster and, in particular, the $O(3^k)$ strategy consistently outperforms the $O(2.42^k)$ strategy. We conjecture the discrepancy between theoretical and practical running times to be an indication for the power of our merging technique. Finally, we report preliminary results of applying our method to gene expression data for tissue classification.

2. Preliminaries

For brevity, we use uv as shorthand for an unordered pair $\{u, v\} \in \binom{V}{2}$. We assume a problem instance to be given in the form of a *weight function* $s : \binom{V}{2} \to \mathbb{R}$: For $s(uv) > 0$ an edge uv is present in the graph and has deletion cost $s(uv)$, while for $s(uv) \leq 0$ the edge uv is absent from the graph and has insertion cost $-s(uv)$.

If all connected components of a graph G are cliques, this graph is called *transitive*. If we modify a graph G to obtain a graph G' which is transitive, we call G' a *clustering* of G. It is graph theory folklore that a graph G is transitive if and only if there are no three vertices which induce a 2-path in G. To this end, we call vuw a *conflict triple* in G if uv and uw are edges of G but vw is not.

When given an input graph G, we first identify its connected components. As it never makes sense to connect two components in a clustering, we calculate the best solutions for all components separately and sum up the costs to obtain the total cost for G.

Beside the input graph G, our fixed-parameter algorithms also require a cost limit k. In order to find an optimal solution, we call the algorithm repeatedly, starting with $k = 1$. If we did not find a solution with this value, we increase k by 1, call the algorithm again and so forth. Note that for real-valued edge weights, we have to traverse the complete search tree and find the *best* solution with cost at most k, if any. If we can decide in $O(c^k)$ time if there is a solution with cost at most k, the above procedure to find an optimal solution of cost at most k can also be executed in $O(c^k)$ time.

To obtain provable running times, we assume that all edges have edit costs of at least 1. There cannot be a fixed-parameter algorithm solving WEIGHTED CLUSTER EDITING problem with arbitrarily small edit costs unless P = NP.

Comment. With such an algorithm we can solve the NP-complete unweighted CLUSTER EDITING problem in polynomial time by assigning uniform edit costs of $\frac{1}{k}$ to all edges of the input graph and searching for a solution of cost 1.

3. Data reduction

In the beginning of our algorithm and in each search node, we call the following data reduction routines in order to downsize the input graph as much as possible.

Rule 1: Remove cliques. Remove already existing cliques from the input graph.

Rule 2: Check for unaffordable edge modifications. For each set of two vertices u, v from V, we calculate a lower bound for the costs induced when uv is set to "permanent" or "forbidden", e.g. when the respective edge is modified. Let $N(v) := \{u \mid s(uv) > 0\}$ denote the set of neighbors of a vertex v, and let $A \triangle B$ be the symmetric set difference of sets A and B. We define induced costs $icf(uv)$ and $icp(uv)$ for setting uv to "forbidden" or "permanent", respectively:

$$\begin{aligned} icf(uv) &= \sum_{w \in N(u) \cap N(v)} \min\{s(uw), s(vw)\} \\ icp(uv) &= \sum_{w \in N(u) \triangle N(v)} \min\{|s(uw)|, |s(vw)|\} \end{aligned} \tag{1}$$

This is how we make use of these values:

- For all $u, v \in V$ where $icf(uv) > k$: Insert uv if necessary, and set uv to "permanent" by assigning $s(uv) \leftarrow +\infty$.
- For all $u, v \in V$ where $icp(uv) > k$: Delete uv if necessary, and set uv to "forbidden" by assigning $s(uv) \leftarrow -\infty$.

If there is a pair uv such that both conditions hold simultaneously, the problem instance is not solvable.

Remark. The above conditions do not take into account edit cost of the edge uv itself. For implementation, we test whether $\max\{0, s(uv)\} + icf(uv) > k$ or $\max\{0, -s(uv)\} + icp(uv) > k$ holds. We disregard this subtlety for the sake of readability.

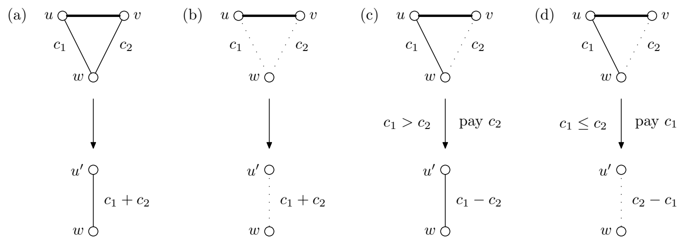

Figure 1. Merging two vertices u, v into a new vertex u': Let $c_1 = |s(uw)|$, $c_2 = |s(vw)|$ be the edit costs. Dotted edges are nonexistent.

Rule 3: Merge vertices incident to permanent edges. As soon as we set an edge uv to "permanent", we infer that u and v must be in the same clique in every solution. In this case we *merge* u and v creating a new vertex u'.

If w is a neighbor of both u and v, we create a new edge $u'w$ whose deletion costs as much as the deletion of both uw and vw. If w is neither a neighbor of u nor of v, we calculate the insertion cost of the nonexistent edge $u'w$ analogously. In case w is a neighbor of u or v but not both, uvw or vuw is a conflict triple, and we must decide whether we delete the edge connecting w with u or v, or we insert the nonexistent edge. By summing the weights (one of which is negative) to calculate $s(u'w)$ we carry out the cheaper operation decreasing k accordingly, and maintain the possibility to edit $u'w$ later.

This is how we merge u and v into a new vertex u': For each vertex $w \in V \setminus \{u, v\}$ set $s(u'w) \leftarrow s(uw) + s(vw)$. Let $k \leftarrow k - icp(uv)$, and delete u and v from the graph. See Fig. 1. Note that these reduction rules conserve the optimal solution.

To start our data reduction, we have to compute $icf(uv)$ and $icp(uv)$ for all $u, v \in V$ what takes $O(|V|^3)$ time. Setting an edge to "forbidden" or "permanent" can reduce the parameter k because we might have to delete or insert an edge. If we merge a permanent edge, this can further reduce the parameter. This, in turn, may trigger other edges to become forbidden or permanent. In addition, setting an edge to "forbidden" or "permanent" will change the induced costs of other edges. We now show how to execute our data reduction for an arbitrary input graph in time $O(|V|^3 \log |V|)$.

Let $n := |V|$. The induced costs $icf(uv)$ and $icp(uv)$ for each vertex pair $u, v \in V$ can be computed in $O(n)$ time. Therefore it initially takes $O(n^3)$ time to compute the induced costs of all $u, v \in V$. Note that during the data reduction, at most $\binom{n}{2}$ edges will be set to "forbidden", and at most $n - 1$ merge operations are executed before the graph collapses into a single vertex. For each $u \in V$ we use a binary heap to store all $icf(uv)$ and another binary heap to store all $icp(uv)$ for $v \in V$. This allows us to find $\max_v\{icf(uv)\}$ and $\max_v\{icp(uv)\}$ for each $u \in V$ in constant time.

We repeatedly do the following: Using $\max_w\{icf(uw)\}$ and $\max_w\{icp(uw)\}$ for each $u \in V$, we find the overall maximum icf and icp value in time $O(n)$. We test if there exist $u, v \in V$ with $icf(uv) > k$ or $icp(uv) > k$. If no such u, v exist, we *stop*. Otherwise, we set the corresponding edge to "forbidden" or "permanent", we update parameter $k \leftarrow k - |s(uv)|$ and also the heaps for icf and icp values as described below. The running time of this part of the algorithm is $O(n^3 \log n)$.

Setting an edge uv to “forbidden” affects the values $icf(ux)$, $icf(vx)$, $icp(ux)$, and $icp(vx)$ for all vertices $x \in V$. We concentrate on updating $icf(ux)$, the other updates can be executed similarly. Let s_0 be our weight function before the update and s_1 after the update, then these functions agree except for $s_0(uv) \neq s_1(uv) = -\infty$. Analogously, let $icf_0(ux)$ and $icf_1(ux)$ denote “induced costs forbidden” of the tuple u, x before and after the update, respectively. If $s_0(uv) \leq 0$ then no edge is deleted and we see from (1) that $icf_1(ux) = icf_0(ux)$ must hold. A similar argument resolves the case $s_0(xv) \leq 0$. If $s_0(uv) > 0$ and $s_0(xv) > 0$ then u, v as well as x, v were adjacent in the initial graph and $icf_1(ux) = icf_0(ux) - \min\{s_0(uv), s_0(xv)\}$ must hold. Clearly, computing $icf_1(ux)$ takes constant time. Updating all affected icf values and all binary heaps takes $O(n \log n)$ time: In case we have to decrease a key we can remove the corresponding entry from the heap in time $O(\log n)$ and reinsert a new entry also in time $O(\log n)$. Because every edge can be set to “forbidden” at most once, and since there are $O(n^2)$ many edges, all updates induced from setting edges to “forbidden” take total time $O(n^3 \log n)$.

When we set an edge uv to “permanent”, the data reduction merges u, v into a new vertex u' and deletes u, v from the graph. We iterate over all vertices $w \in V$ and first compute $s(u'w) \leftarrow s(uw) + s(vw)$ as well as $icf(u'w)$ and $icp(u'w)$ using (1). Analogous to the previous paragraph, this affects the values $icf(wx)$ and $icp(wx)$ for all vertices $x \in V$. For every vertex w, computing $icf(u'w)$, $icp(u'w)$ and updating all heaps takes time $O(n \log n)$, and so does updating the induced costs of all icf and icp values affected by $s(u'w)$. Hence, merging an edge can be executed in total time $O(n^2 \log n)$. There can be at most $n - 1$ merge operations, so the running time of all merge operations is also bounded by $O(n^3 \log n)$. Finally, we can detect and remove all connected components which are cliques in time $O(n^2)$.

The following lemma shows that our data reduction produces a problem kernel as the size of the resulting graph is polynomial in k. We omit the proof for the sake of brevity.

Lemma 3.1. *If every edge deletion or insertion has cost at least* 1*, then our data reduction results in a problem kernel with at most* $2k^2 + k$ *vertices and* $2k^3 + k^2$ *edges.*

4. Initial branching strategy

Given a weighted graph $G = (V, E)$, we now describe a simple recursive algorithm that is guaranteed to find an optimal solution for WEIGHTED CLUSTER EDITING. Recall that an undirected graph is transitive if and only if it does not contain a conflict triple. Our algorithm takes advantage of this observation: Search for a conflict triple, and let u be the vertex of degree two and v, w be the leaves. For algorithmic reasons, we can set existent (nonexistent) edges to “permanent” (“forbidden”) by assigning infinite edit costs to them. Recursively branch into three cases:

(1) Insert vw, set uv, uw, and vw to “permanent”.
(2) Delete uv, set uw to “permanent” and uv and vw to “forbidden”.
(3) Delete uw, set uw to “forbidden”.

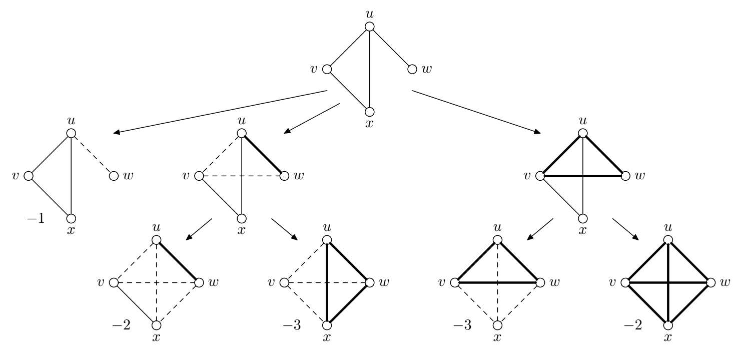

Figure 2. Case (W2a): Vertices v, w do not share a common neighbor; v has a neighbor x connected with u.

In each branch, we lower k by the insertion or deletion cost required for the executed operation. If a connected component decomposes into two components, we calculate the optimum solutions for these components separately. If k falls below zero, we discard the respective branch of the algorithm. Again, we omit the proof.

Theorem 4.1. *If every edge of the weighted graph $G = (V, E)$ has weight of at least one, the* WEIGHTED CLUSTER EDITING *problem can be solved in $O(3^k + |V|^3 \log |V|)$ time.*

5. Refined branching strategy

In the following, we will refine the simple branching strategy resulting in a search tree of size $O(2.42^k)$, considering induced subgraphs of size 4. Unfortunately, the $O(2.27^k)$ branching strategy of Gramm *et al.*[9] cannot be used in the weighted case because it is based on an observation (Lemma 5) that does not hold for weighted graphs. We now modify this branching strategy accordingly.

Note that the automated search tree generator of Gramm *et al.*[10] also found an $O(2.42^k)$ search tree for induced subgraphs of size 4, but the branching strategy is not explicitly described there. If we consider induced subgraphs of size 5, this results in an $O(2.27^k)$ search tree.[10] The latter branching strategy requires case distinction with 20 initial cases and branching vectors of size at most 16. In comparison, our branching strategy distinguishes only four initial cases and branching vectors of length five.

Let vuw be a conflict triple as above. We distinguish the following cases:

(W1) Vertices v, w have no neighbors except for u, that is, $N(v) = \{u\}$ and $N(w) = \{u\}$.

(W2) Vertices v, w do not share a common neighbor, but there exists a vertex x such that, say, $vx \in E$. We distinguish two sub-cases: (W2a) $ux \in E$ (see Fig. 2), and (W2b) $ux \notin E$ (see Fig. 3).

(W3) Vertices v, w share a common neighbor $x \neq u$, so $vx \in E$ and $wx \in E$. We distinguish two sub-cases: (W3a) $ux \in E$ (see Fig. 2 in Gramm *et al.*[9]), and (W3b) $ux \notin E$ (see Fig. 3 in Gramm *et al.*[9]).

In case (W1) holds, we *ignore* the conflict triple vuw for the moment, and continue

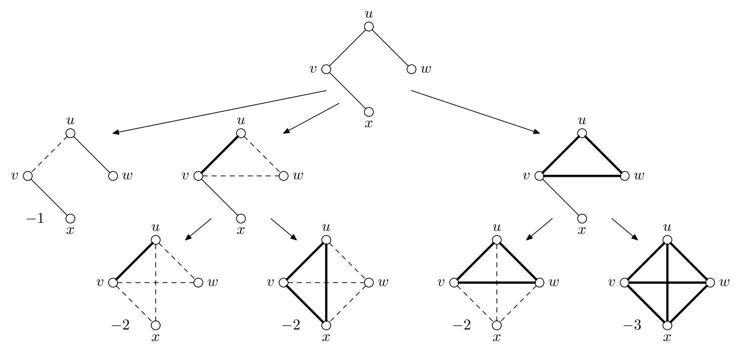

Figure 3. Case (W2b): Vertices v, w do not share a common neighbor; v has a neighbor x not connected with u.

with the next triple. In all other cases, we branch as indicated by Figs. 2, 3 in this article and Figs. 2, 3 in Gramm *et al.*[9] We describe the branching in detail for case (W2a), see Fig. 2: Here, edges uv, uw, ux, and vx are present in the induced graph. We branch into five sub-cases:

- Delete uw and set uw to "forbidden".
- Set uw to "permanent", delete uv, ux, and set uv, ux, vw, wx to "forbidden".
- Insert wx, set uw, ux, wx to "permanent", delete uv, vx, and set uv, vw, vx to "forbidden".
- Insert vw, set uv, uw, vw to "permanent", delete ux, vx, and set ux, vx, wx to "forbidden".
- Insert vw, wx and set all six edges to "permanent".

The branching strategies for case (W2b) can be easily derived from Fig. 3. Regarding branching strategies for cases (W3a) and (W3b) we refer to cases (C2) and (C3) of Gramm *et al.*[9]

One can easily check that if only conflict triples of type (W1) are present in a connected graph, this graph is a star graph, that is, a tree where all vertices but one are leaves. It is straightforward how to quickly find an optimal solution for this case. We omit the details. The analysis of the refined branching strategy leads to Theorem 5.1.

Theorem 5.1. *If every edge of the weighted graph $G = (V, E)$ has a weight of at least one, then the total running time using our refined branching strategy is $O(2.42^k + |V|^3 \log |V|)$.*

6. Experiments and results

To explore the performance of our algorithms for WEIGHTED CLUSTER EDITING and compare the abovementioned branching strategies, we test both algorithms on the same artificial and protein similarity datasets we used in Rahmann *et al.*[8] For a given number of vertices, an artificial instance is generated by first creating a random number of clusters of random size, then setting edge weights and false edges using a Gaussian distribution. The probability to see an undesired or missing edge in the resulting graph is about $15\,\%$. The

Table 1. Results for artificial data. Each row of the table corresponds to ten instances.

$\lvert V\rvert$	$\lvert E\rvert$	# edit	avg. cost	avg. edge	time 3^k	time 2.42^k	no merging
10	11–30	8.30	95.77	24.05	10 ms	11 ms	11 ms
20	65–165	28.10	301.9	24.41	54 ms	56 ms	69 ms
30	138–296	66.70	671.2	23.79	1.0 s	1.9 s	8.3 s
40	251–533	115.5	1238.3	24.30	29 s	52 s	31 min
50	402–821	183.2	1860.0	23.94	7.6 min	28 min	> 5 h

Note: '# edit' is the average number of edit operations, 'avg. cost' is the average total cost, 'avg. edge' is the average cost per edge, and 'no merging' is the running time of the 2.42^k branching strategy without merging permanent edges.

random instances can be seen as a worst case scenario, as we expect biological instances to be closer to a clustering. Our biological instances stem from protein similarity data, generated using more than 192 000 protein sequences from the COG dataset.[12] Edge weights are computed from $\log$ E-values of bidirectional BLAST hits using a threshold of 10^{-10}: The modification cost of each vertex pair is the difference between the logarithms of the threshold and the respective proteins' E-values. In the resulting graph 3964 connected components are not transitive, and 3788 of these have up to 100 vertices. See Rahmann *et al.*[8] for more details. Recently, Wittkop *et al.*[13] showed that the Cluster Editing model leads to valid clusterings when applied to protein similarity data and manages to outperform other methods.

We implemented the weighted data reduction and both branching strategies in C++. The results are reported in Tables 1 and 2. Running times were measured on an AMD Opteron-275 2.2 GHz with 3 GB of memory running SunOS 5.1. For protein similarity data, three instances with 84, 91, and 98 vertices did not stop after 13 days of computation with either branching strategy and are omitted from Table 2. Using the $O(2.42^k)$ strategy another five instances did not stop after 13 days of computation. For the $O(3^k)$ strategy only 30 out of 3788 instances (0.8 %) had a running time of more than ten minutes. These components are typically not near transitivity and admit many cheap edge modifications. So, despite the hardness of the WEIGHTED CLUSTER EDITING problem and the worst-case running times of our branching strategies, optimal solutions were usually computed in a matter of minutes. Note that a naïve algorithm using exhaustive enumeration cannot handle instances with more than, say, 12 vertices.

We want to evaluate whether it pays off in practice to follow a more complicated and thus time-consuming strategy to obtain a better worst-case running time. Recent experiments by Dehne *et al.*[14] show that on unweighted graphs the $O(2.27^k)$ branching is faster than the $O(3^k)$ branching.[9] As can be seen in Tables 1 and 2, in our experiments the basic strategy outperforms the refined branching strategy for all graph sizes, in contrast to the fact that its worst-case running time is inferior.

One potential cause for this unexpected result is the fact that both strategies immediately merge permanent edges. To estimate the impact of merging edges, we evaluate both branching strategies using the same implementation, but we disable merging permanent edges. Here, the refined strategy is faster than the simple branching strategy, in agreement with theoretical running times (data not shown). In addition, both algorithms are significantly slower than our branching strategies that merge edges, see Table 1.

Finally, we report some preliminary results regarding tissue classification using gene expression data: here, one uses gene expression data to discriminate between similar cancer types. We analyzed four datasets from Monti *et al.*[3] where the correct classification is known: "Leukemia" with 38 samples, "Novartis multi-tissue" with 103 samples, "CNS tumor" with 42 samples, and "Normal tissue" with 90 samples. See Monti *et al.*[3] for details and references on these datasets. We transformed each gene expression matrix into a distance matrix between samples using normalized scalar products. For our intitial study, we did not transform these values into log-odds as suggested by Sharan *et al.*[2] but used "raw" distances. We believe that applying the probabilistic framework of Sharan *et al.*[2] will further improve the quality of our results. Thresholds for building the weighted graph were set by manual inspection, but repeated runs showed that our method was relatively insensitive to varying these thresholds. Running times of our approach range from a few seconds, to 4.6 days for the "Novartis" dataset. In three out of four cases, our results outperform that of all methods investigated in Monti *et al.*:[3] For example, using the "Novartis" dataset our clustering has an adjusted Rand index of 0.934 while the best method reported in Monti *et al.*,[3] namely Hierarchical Clustering, results in an adjusted Rand index of 0.921.[a] We will further investigate the application of our method to the problem of tissue classification in the future.

7. Conclusion

Albeit the undisputed elegance of the fixed-parameter approaches for unweighted CLUSTER EDITING, their practical use in computational biology is limited because biological data is almost always weighted in nature. Here, we have presented fixed-parameter algorithms for the WEIGHTED CLUSTER EDITING problem that guarantee to find the optimal solution with provable worst-case running times. In application, running times of our algorithms are much better than these worst-case bounds suggest: For biological data where the input graph is sufficiently close to a clustering, our algorithms finds the optimal solution

[a]The adjusted Rand index[15] is a measure of agreement between partitions with potentially different numbers of clusters. An index of 1 corresponds to perfect agreement, the expected value for two random partitions is 0.

Table 2. Results for protein similarity data.

$\lvert V\rvert$	$\lvert E\rvert$	# inst.	# edit	avg. cost	avg. edge	time 3^k	time 2.42^k	# hard
3–10	2–44	2075	2.32	6.63	19.75	2.0 ms	2.0 ms	0
11–20	14–189	725	11.78	36.49	23.23	12 ms	14 ms	0
21–30	47–434	307	30.92	95.56	23.31	103 ms	184 ms	0
31–40	62–773	178	62.02	200.1	21.52	680 ms	2.2 s	0
41–50	143–1224	181	101.6	333.9	21.26	91 s	7.1 min	3
51–60	204–1603	117	137.4	424.2	21.38	24 min	$>$ 3.2 h	8
61–70	191–2214	93	176.4	616.5	26.08	47 min	$>$ 3.6 h	1
71–80	614–3078	57	227.8	820.2	26.39	6.8 h	$>$ 16 h	10
81–90	266–3388	29	430.9	1372.7	25.87	3.7 h	$>$ 27 h	5
91–100	494–4504	23	315.0	1332.6	37.04	49 s	7.1 min	0

Note: '# inst.' is the number of instances in this group. '# hard' is the number of "hard" instances in this group where the $O(3^k)$ strategy had running time of more than ten minutes.

in reasonable time even when hundreds of edge modifications are necessary. We successfully applied our algorithm to protein similarity data, and reported first results for tissue classification using gene expression data.

Different from what worst-case running times suggest, our $O(3^k)$ branching strategy almost always outperforms the $O(2.42^k)$ strategy. The reason for this unexpected outcome is certainly the power of the merge operation which outweighs advantages of complicated branching rules. We think that there may exist simple branching strategies that make even better use of edge merging, resulting in faster running times in practice and, eventually, also in improved worst-case bounds.

Acknowledgments

We thank Jan Baumbach, Sven Rahmann, and Tobias Wittkop for providing both artificial and protein similarity data.

Bibliography

1. A. Krause, J. Stoye and M. Vingron, *BMC Bioinformatics* **6**, p. 15 (2005).
2. R. Sharan, A. Maron-Katz and R. Shamir, *Bioinformatics* **19**, 1787(Sep 2003).
3. S. Monti, P. Tamayo, J. Mesirov and T. Golub, *Mach. Learn.* **52**, 91 (2003).
4. A. Ben-Dor, R. Shamir and Z. Yakhini, *J. Comput. Biol.* **6**, 281 (1999).
5. R. Shamir, R. Sharan and D. Tsur, *Discrete Appl. Math.* **144**, 173 (2004).
6. M. Křivánek and J. Morávek, *Acta Inform.* **23**, 311(June 1986).
7. E. Hartuv, A. O. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach and R. Shamir, *Genomics* **66**, 249 (2000).
8. S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truß and S. Böcker, Exact and heuristic algorithms for Weighted Cluster Editing, in *Proc. of Computational Systems Bioinformics (CSB 2007)*, 2007.
9. J. Gramm, J. Guo, F. Hüffner and R. Niedermeier, *Theor. Comput. Syst.* **38**, 373 (2005).
10. J. Gramm, J. Guo, F. Hüffner and R. Niedermeier, *Algorithmica* **39**, 321 (2004).
11. R. Niedermeier, *Invitation to Fixed-Parameter Algorithms* (Oxford University Press, 2006).
12. R. L. Tatusov, N. D. Fedorova, J. D. Jackson, A. R. Jacobs, B. Kiryutin, E. V. Koonin, D. M. Krylov, R. Mazumder, S. L. Mekhedov, A. N. Nikolskaya, B. S. Rao, S. Smirnov, A. V. Sverdlov, S. Vasudevan, Y. I. Wolf, J. J. Yin and D. A. Natale, *BMC Bioinformatics* **4**, p. 41 (2003).
13. T. Wittkop, J. Baumbach, F. P. Lobo and S. Rahmann, Large-scale clustering of protein sequences with FORCE – a layout based heuristic for Weighted Cluster Editing., To appear in BMC Bioinformatics, (2007).
14. F. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw and Y. Zhang, The Cluster Editing problem: Implementations and experiments, in *Proc. of International Workshop on Parameterized and Exact Computation (IWPEC 2006)*, , LNCS Vol. 4169 (Springer, 2006).
15. L. Hubert and P. Arabie, *J. Classif.* **2**, 193 (1985).