Next Generation Cluster Editing

Thomas Bellitto[1], **Tobias Marschall**[2,3], **Alexander Schönhuth**[4], **and Gunnar W. Klau**[4]

[1]**Combinatorics and Algorithms Team, LaBRI, Bordeaux, France**[*]
[2]**Center for Bioinformatics, Saarland University, Saarbrücken, Germany**
[3]**Max Planck Institute for Informatics, Saarbrücken, Germany**
[4]**Life Sciences, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands**

ABSTRACT

Genomic structural variations play key roles in genetic diversity and disease. Despite recent advances in structural variation discovery, many variants are yet to be discovered. Midsize insertions and deletions pose particularly involved algorithmic challenges. The recent CLEVER algorithm addressed these challenges with a statistical model on cliques in a graph whose nodes are read alignments and whose edges arise from a statistical test on length and overlap of read alignments. However, the resulting read alignment clusters tend to be too small and are heavily overlapping, which leads to losses in recall performance rates. Here we present a model based on *weighted cluster editing*, which alleviates these issues: clusters are provably non-overlapping and tend to be larger. In order to render the inherent optimization problem tractable on all read alignments of a genome, we present a novel, principled heuristic, which runs in time linear in the length of the genome. The heuristic is based on an exact polynomial-time algorithm for weighted cluster editing in one-dimensional point graphs. We demonstrate that the new model improves recall rates achieved by CLEVER.

Keywords: bioinformatics; next-generation sequencing; graph theory; cluster editing

INTRODUCTION

High-throughput sequencing-based genome analyses have become routine, and databases are rapidly filling up with sequencing and variant data. Single nucleotide polymorphisms (SNPs) as well as more difficult to discover larger structural variations play key roles in genetic diversity and disease (Cirulli and Goldstein, 2010). However, despite the most recent experimental advances also known as third-generation sequencing (TGS) (Eid et al., 2009, PacBio), (Clarke et al., 2009, OxfordNanopore), many genetic variants are yet to be discovered, and the extent of such missing variants and their effects are still largely unclear. This explains why further computational advances in variant discovery and genome assembly are still necessary.

Among such variants, midsize and also some longer insertions and deletions (indels) have remained a particularly complex issue (Marschall and Schönhuth, 2015). The problem is that these variants give rise to signals that interfere with the statistical uncertainties of the read alignment data. Even TGS will most likely not lead to immediate and accurate discovery of midsize indels as the increasing read length comes at the expense of higher indel sequencing errors. This adds another layer of confounding factors on the classical problem of correctly placing gaps in alignments. As of today, every computational advance in midsize and long indel discovery is still highly appreciated.

While the first methods addressing midsize indel discovery suggested solutions that were helpful in general (Chen et al., 2009, BreakdancerMini), (Lee et al., 2009, MoDIL), they proved to be too slow or too inaccurate in practice. Only in the recent past, practically viable solutions were presented (Ye et al., 2009, Pindel, for indels up to 50-60 bp in particular), (Marschall et al., 2012, 2013, CLEVER, MATE-CLEVER, for indels from 40 bp in particular). The combined application of those tools lead to the discovery of significant amounts of such indels also in a population-scale sequencing project (The Genome of the Netherlands Consortium, 2014). Despite these advances, a substantial amount of variants has still not been discovered. The driving idea behind CLEVER (Marschall et al., 2012, 2013) has been *to not discard concordant* read data. CLEVER clusters individual reads and uses a statistical model to assess the signal of the cluster formed by the reads, as a whole, because only then they make a strong variant signal. This is in contrast to all prior methods that throw away concordant data first, that is, assess the signal of single reads before forming clusters.

[*]This work was done during an internship of the first author at CWI.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1301v1 | CC-BY 4.0 Open Access | rec: 13 Aug 2015, publ: 13 Aug 2015

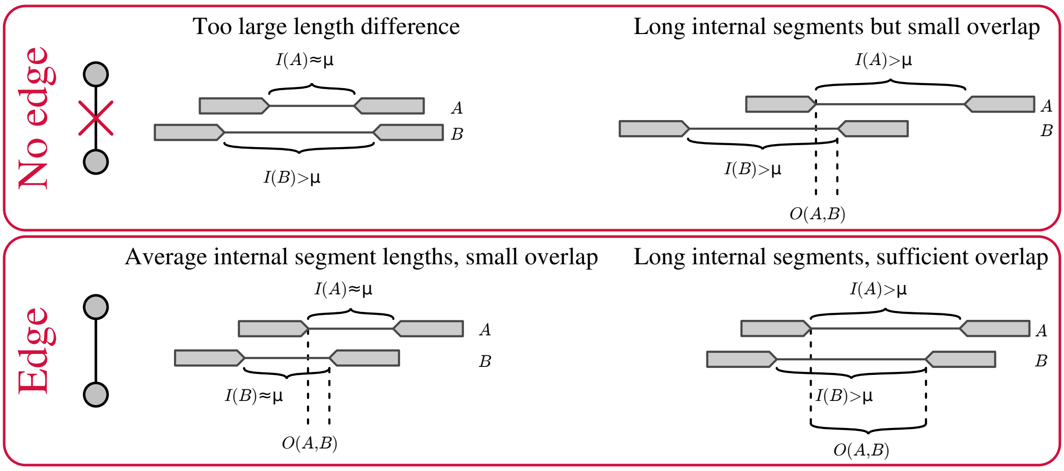
No edge
Too large length difference
I(A)≈μ
A
B
I(B)>μ
Long internal segments but small overlap
I(A)>μ
A
B
I(B)>μ
O(A,B)
Edge
Average internal segment lengths, small overlap
I(A)≈μ
A
B
I(B)≈μ
O(A,B)
Long internal segments, sufficient overlap
I(A)>μ
A
B
I(B)>μ
O(A,B)

Figure 1. Different scenarios of when an edge is added (bottom) or not added (top) to the read alignment graph depending on the values of $I(A)$, $I(B)$, and $O(A,B)$.

CLEVER solves the clustering step by determining all maximal cliques in the read alignment graph and filters them with another statistical test. This leads to highly overlapping clusters. Also, due to lack of coverage, repeats and the resulting misplaced alignments, and too low allele frequency, the clusters can still be too small to be called statistically reliably. This leads to reductions in recall, see the following section for more explanations.

In this paper, we suggest *cluster editing* as a different cluster model, and we develop corresponding algorithms to address the issues further sketched in the following section. First, we demonstrate that our algorithm, despite its heuristic nature, yields near-optimal solutions when solving the weighted cluster editing problem on one-dimensional point graphs. Second, we will show that the runtime of our algorithm is linear in the length of the genome, and polynomial in other relevant parameters, such as the coverage. Speed is crucial as, when keeping all concordant read data, one faces a true *Big Data* problem—genomes easily amount to hundreds of gigabytes. Third, we demonstrate that the resulting output is reasonable and indeed has the potential to lead to improvements in recall.

CLIQUE-BASED CLUSTERING AND ITS LIMITS

In this section we introduce the *read alignment graph* and the clique-based clustering approach that is used in CLEVER (Marschall et al., 2012). CLEVER is a paired-end read approach. As such, it is based on the analysis of alignments of fragments whose ends are sequenced, while the internal part (internal segment) is not. Hence the length of the entire fragment is unknown. The idea of a paired-end read approach is to compare the length of the paired-end read alignments with the empirical fragment length distribution. Based on such comparisons, one can determine which read alignments significantly deviate from the distribution. For such read alignments, an indel variant that affects the internal segment is most likely the reason for the deviation.

The vertices of the read alignment graph represent paired-end read alignments. Two read alignments are linked by an edge if they are "close" according to a statistical criterion, which expresses that they are sufficiently likely to support identical (indel) alleles. Intuitively, (i) linked read alignments should sufficiently overlap as they are supposed to reflect locally identical alleles, and, (ii) the sizes of the internal segment of the reads should agree in terms of representing identical indel variants—or, of course, no variant at all.

In the following, let A,B be paired-end read alignments. We denote the length of A by $I(A)$, which is the difference between the leftmost alignment coordinate of the right read end and the rightmost alignment coordinate of the left read end, see Figure 1. We make the assumption that this length is governed by a Gaussian distribution $\mathcal{N}_{\mu,\sigma}$ with mean μ and standard deviation σ, which is well justified for current sequencing protocols. We further define $O(A,B)$ as the size of the overlap of the two alignments, $\Delta(A,B) := |I(A) - I(B)|$ as the length difference of the two alignments, $\bar{I}(A,B) := (I(A) + I(B))/2$ as their average length, and the quantity $U(A,B) := \bar{I}(A,B) - O(A,B)$, which helps checking if the overlap is compatible with a potential indel supported by A and B.

Let $X \sim \mathcal{N}_{(0,1)}$ be a normally distributed random variable. There is an edge (A,B) if and only if both

$$\mathbb{P}\left(|X| \geqslant \frac{1}{\sqrt{2}}\frac{\Delta(A,B)}{\sigma}\right) \geqslant T \qquad \text{and} \qquad \mathbb{P}\left(X \geqslant \sqrt{2}\frac{U(A,B)-\mu}{\sigma}\right) \geqslant T \tag{1}$$

apply, which are standard Z-tests. We refer to them as *size criterion* and *overlap criterion*, respectively. The size criterion expresses that the lengths of the alignments A and B should not differ by an amount that contradicts the fragment length statistics, hence are likely to support identical alleles. The overlap criterion expresses that, if two alignments A and B are both too long and hence give rise to a deletion signal, there should be sufficient overlap to harbor the deletion they both support. Note that lowering T leads to increasing amounts of edges, hence denser read alignment graphs.

The clustering approach of CLEVER is based on enumerating all maximal cliques in the read alignment graph. The algorithm processes alignments from left to right, maintaining a set of *active cliques*, where a clique becomes inactive as soon as their rightmost alignment can no longer overlap with so far unprocessed alignments. The corresponding algorithm runs in time $O(n(\log n + kc^2) + s)$ where k is an upper bound on the size of the cliques, c is an upper bound on the size of the set of active cliques, and s is the size of the output. While this is not guaranteed in theory, we observe for read alignment graphs that c correlates linearly with k.

Clique-based clustering has been shown to work well. However, there are also drawbacks and limitations. First, the model is very restrictive, in the sense that a missing edge between two reads makes it impossible to have them in the same cluster. Also, there are a lot of overlapping cliques in the read alignment graph. A third drawback of the clique-based clustering is its computational complexity. On high coverage data, k and c reach high values such that the $O(n(\log n + kc^2) + s)$ runtime may become problematic.

A WEIGHTED CLUSTER EDITING MODEL

We suggest a new clustering model that avoids overlapping predictions and a heuristic algorithm that computes these predictions in quasi-linear time. We thereby address the drawbacks of the clique-based clustering approach described in the last section.

Given an undirected unweighted graph $G = (V, E)$, the cluster editing problem consists of transforming G into a clustered graph G', *i.e.*, into a union of disjoint cliques by adding or removing edges a minimal number of edges. A solution to this problem can be represented by the set $R \subset E$ of edges to remove and the set $A \subset \binom{V}{2} \setminus E$ of edges to add, or alternatively by a partition of V into clusters. The cost of a solution is the number $|A| + |R|$ of modifications. The cluster editing problem consists in finding a solution with the smallest cost.

Definition 1 (weighted graph)**.** *A weighted graph is an ordered pair (V, w), where V is the set of vertices of the graph and $w : \binom{V}{2} \to \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ is the weight function. We consider that there is an edge between vertices a and b iff $w(a,b) \geqslant 0$.*

Definition 2 (positive/negative part of a function)**.** *The positive and negative part of a function f, respectively written f^+ and f^- are defined on the same domain as f by $f^+(x) = \max(f(x), 0)$ and $f^-(x) = \max(-f(x), 0)$.*

The weighted cluster editing problem still consists of transforming the initial graph $G = (V, w)$ into an unweighted clustered graph G', but this time, the edges have a deletion/insertion cost given by their weight. Thus, the cost of inserting an edge in G' between a and b is $w^-(a,b)$ and the cost of removing an edge is $w^+(a,b)$. Note that the unweighted case is a special case of the weighted problem where every edge has weight 1 and every non-edge has weight -1.

Cluster editing can be seen as a compromise between high edge-density community searching and low cut searching, two NP-hard problems. Not surprisingly cluster editing is also NP-hard as shown by Kratochvíl and Krivánek (1988). The problem is even APX-hard, that is, it is NP-hard to approximate the optimal solution with a given approximation factor $k > 1$ (Charikar et al., 2005), and the best deterministic approximation factor reached today with a polynomial time algorithm is 2.5 (van Zuylen and Williamson, 2007). Interestingly, deciding whether the cluster editing problem is NP-complete in the case of interval graphs is still an open question.

We propose a weighted cluster editing model for read alignments using the following weights.

The weight of a pair of read alignments A and B is $w(A,B) = \min\{w_{\text{size}}(A,B), w_{\text{overlap}}(A,B)\}$ with

$$w_{\text{size}}(A,B) = \ln T - \ln\left(\mathbb{P}\left(|X| \geqslant \frac{1}{\sqrt{2}}\frac{\Delta(A,B)}{\sigma}\right)\right)$$

$$w_{\text{overlap}}(A,B) = \begin{cases} -\infty & \text{if } O(A,B) = 0 \\ \ln T - \ln\left(\mathbb{P}\left(X \geqslant \sqrt{2}\frac{U(A,B)-\mu}{\sigma}\right)\right) & \text{if } O(A,B) \geqslant 0 \end{cases}$$

This weights reflect a transformation of the difference of the logarithms of the terms in (1) into reasonable edge weights. Note that, in a rough sketch, adding up our edge weights, as is done in weighted cluster editing, translates into multiplying probabilities that the affected alignments support the same allele. This in turn reflects the joint probability that all of the edges added exist, and that all of the edges removed do not exist.

Cluster editing naturally partitions the read alignment graph into non-overlapping clusters, which mostly correspond to non-overlapping predictions. The weighted edit operations provide a meaningful way to relax the strict definition of cliques. However, the problem is harder to solve and we therefore focus on heuristic algorithms.

ALGORITHMS

Many excellent heuristic and exact cluster algorithms exist. Well known examples are CAST (Ben-Dor et al., 1999), CLICK (Sharan et al., 2003), HCS (Hartuv et al., 1999), TransClust (Wittkop et al., 2010) and yoshiko (Böcker et al., 2011). However, the size of realistic read alignment inputs allows practically only linear time and space algorithms. We therefore designed a new specific algorithm. In this section we present a linear time heuristic for the weighted cluster editing model that is based on solving the problem exactly on one-dimensional point graphs.

Definition 3 (point graph)**.** *Given a set of points V in an n-dimensional metric space (M,d) and a threshold distance $l \in \mathbb{R}^+$, the induced point graph is the unweighted graph $G(V,E)$ with edge set $E = \{(a,b) \in \binom{V}{2} \mid d(a,b) \leqslant l\}$. An unweighted graph G is said to be an n-dimensional point graph iff there exists an n-dimensional metric space (M,d), a threshold distance l and a set of points $V \subset M$ that induces G.*

A one-dimensional point graph allows placing its vertices on a line. This provides an order on the vertices, for example, from left to right. The main result about cluster editing on point graphs is the following:

Theorem 1 (Mannaa (2010))**.** *Given a one dimensional point graph, there exists an optimal clustering where all clusters contain only consecutive vertices.*

This result considerably reduces the size of the set of potential solutions to investigate and makes this set much easier to enumerate. Instead of enumerating the whole set of partitions of V, we only scan the points from left to right and have to determine whether we end the cluster or not after each vertex. A dynamic programming algorithm (Mannaa, 2010) follows: for $j \in [0,|V|]$, we call $\text{opt}(j)$ the cost of optimally clustering vertices 0 to j and for $j \in [0,|V|]$, $i \in [0,j]$, we call $\text{opt}'(j,i)$ the cost of the best clustering for vertices 0 to j, where the last cluster has size $i+1$. We thus have:

$$\text{opt}(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min\limits_{0 \leqslant i \leqslant j} \text{opt}'(j,i) & \text{if } j > 0 \end{cases} \qquad \text{opt}'(j,i) = \begin{cases} \text{opt}(j-1) + \deg^+(v_j) & \text{if } i = 0 \\ \text{opt}'(j-1,i-1) + |i - \deg^+(v_j)| & \text{if } 0 < i \leqslant j \end{cases}$$

Here $\deg^+(v)$ denotes the outdegree of a vertex v, *i.e.* the number of neighbors of v that are to the right of v. We can compute all opt' value in $O(n^2)$ time and space:

Algorithm 1: Computing the values of opt'

```
$\text{opt}'(0,0) \leftarrow 0$
for $j$ from 1 to $n-1$ do
    $\text{opt}'(j,0) \leftarrow \deg^+(v_j) + \min_{0 \leqslant i \leqslant j-1} \text{opt}'(j-1,i)$
    for $i$ from 1 to $j$ do
        $\text{opt}'(j,i) \leftarrow |i - \deg^+(v_j)| + \text{opt}'(j-1,i-1)$
```

We can derive the cluster boundaries by searching where the function $\mathrm{opt}'(j,\cdot)$ reaches its minimum. Once we know opt', it can be processed in time $O(n)$.

If the lengths of read alignments that define the read alignment graph were all identical, they would indeed define a one-dimensional point graph, whose point set is for example the left endpoint of the intervals, and whose threshold distance is their length. Sorting the reads by their left endpoint would allow us to use Mannaa's algorithm to solve the cluster editing problem in the unweighted case. We will build on this observation by first extending the algorithm to the weighted case. We define a weighted point graph as follows:

Definition 4 (weighted point graph)**.** *Given a set of points V in an n-dimensional metric space (M,d) and a decreasing function $f:\mathbb{R}^+ \to \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty,+\infty\}$, the induced point graph is the weighted graph $G = (V, w:(a,b) \mapsto f(d(a,b)))$. A weighted graph G is said to be an n-dimensional weighted point graph iff there exists an n-dimensional metric space (M,d), a decreasing function $f:\mathbb{R}^+ \to \overline{\mathbb{R}}$ and a set of points $V \subset M$ that induce G.*

An unweighted point graph is a special case of weighted point graph with $f:\mathbb{R}^+ \to \overline{\mathbb{R}}$ given by $a \mapsto 1$ if $a \leqslant l$ and $a \mapsto -1$ otherwise. Suppose now we create a weighted read alignment graph with only read alignments of the same length. The closer the left endpoints of the two reads, the larger their overlap and the heavier the corresponding edge. So, if all read alignments are equally long, weighted read alignment graphs are one dimensional weighted point graphs, like in the unweighted case. Similarly, Theorem 1 can be extended to weighted point graphs by an analogous proof.

Theorem 2. *Given a one-dimensional weighted point graph, there exists an optimal clustering where all clusters contain only consecutive vertices.*

The dynamic programming algorithm for weighted cluster editing on one-dimensional point graphs is as follows:

$$\mathrm{opt}(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min\limits_{0 \leqslant i \leqslant j} \mathrm{opt}'(j,i) & \text{if } j > 0 \end{cases}$$

$$\mathrm{opt}'(j,i) = \begin{cases} \mathrm{opt}(j-1) + \sum\limits_{k=0}^{j-1} w^+(j,k) & \text{if } i = 0 \\ \mathrm{opt}'(j-1,i-1) + \sum\limits_{k=0}^{j-i-1} w^+(j,k) + \sum\limits_{k=j-i}^{j-1} w^-(j,k) & \text{if } 0 < i \leqslant j \end{cases}$$

The formula for each of the $O(n^2)$ values of opt' now includes a sum that takes $O(n)$ time to compute. Still, those sums are often alike and by computing only one of them, we can deduce the next one in constant time and we still end up with an $O(n^2)$ algorithm:

Algorithm 2: Computing the values of opt' in a weighted graph

```
$\mathrm{opt}'(0,0) \leftarrow 0$
for $j$ from 1 to $n-1$ do
    $X \leftarrow \sum_{0 \leqslant k \leqslant j-1} w^+(j,k)$
    $\mathrm{opt}'(j,0) \leftarrow X + \min_{0 \leqslant i \leqslant j-1} \mathrm{opt}'(j-1,i)$
    for $i$ from 1 to $j$ do
        $X \leftarrow X - w(j,j-i)$
        $\mathrm{opt}'(j,i) \leftarrow X + \mathrm{opt}'(j-1,i-1)$
```

Note that if we do not initialize X with $\sum\limits_{0 \leqslant k \leqslant j-1} w^+(j,k)$, $\mathrm{opt}(j)$ will no longer give the cost of the optimal clustering for vertices 0 to j. However, X is nothing but an additive constant in $\mathrm{opt}'(j,\cdot)$ and what we are really interested in is to determine where the minimum of $\mathrm{opt}'(j,\cdot)$ is reached. If we initialize X with any other value, for example, zero, we will still find the optimal clustering.

When the number of nodes of the input graph reaches hundreds of millions, even $O(n^2)$ algorithms are not suitable anymore. However, we can exploit the following fact: if the optimal clustering for nodes 0 to j is reached with a last cluster of size i, chances are very low that the optimal clustering

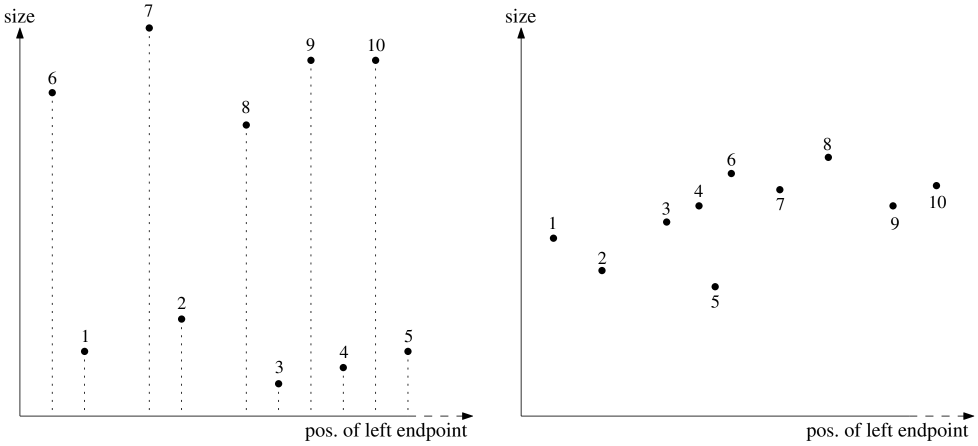

Figure 2. Problematic instances for **(left)** the left-to-right approach; **(right)** the closest-neighbour approach.

for nodes 0 to $j+1$ is reached with a last cluster of size strictly bigger than $i+1$ and we do not need to compute $\text{opt}'(j+1,k)$ for $k \in [i+2, j+1]$. There are of course counter examples, especially with small clusters, but this heuristic argument is generally safe for clusters of hundreds and more vertices.

We therefore obtain a heuristic algorithm by making the following change on line 5 of the exact algorithm: **for** *i from* 1 *to* $\text{argmin}(\text{opt}'(j,\cdot))+1$ **do**

If we initialize X with 0 as suggested above, our heuristic runs in $O(nk)$ where k is the size of the clusters. We are not, however, able to determine the cost of the provided solution in less than $O(n^2)$. Finally, we can note that to determine the clustering, all we need to know at the end are the $\text{argmin}(\text{opt}'(j,\cdot))_{0\leqslant j\leqslant n}$. Once we are done computing $\text{opt}'(j,\cdot)$, we can erase the values of $\text{opt}'(j-1,\cdot)$ as long as we store $\text{argmin}(\text{opt}'(j-1,\cdot))$. This reduces the space complexity of the algorithm from $O(n^2)$ to $O(n)$.

Knowing the left endpoint and length of every alignment gives us the required information to determine the weights of the edges. Therefore we can place our reads in a two-dimensional plane and look for a metric and a weight function that would make it a point graph. However, the dynamic programming solution for one-dimensional point graphs is only possible because of the special structure of solution space in one dimension as described by Theorem 2, which does not easily generalize to higher dimensions.

Nevertheless, in most of the cases the weight of the edges will be determined by the position of the reads and not by the size difference. Thus, even though the data is characterized by two dimensions, it is dominated by the positional dimension, and we can still hope that the one-dimensional study helps.

Looking back at Algorithm 2, we realize that it solves a more general problem than just optimal clustering for one-dimensional point graphs: given any kind of graph and an order on the vertices, it will find the optimal clustering where clusters only contain consecutive vertices. Regarding the heuristic algorithm, if two vertices with a very heavy edge are far from each other in the order, the algorithm may not even try to put them in the same clusters and may thus pay a high cost. This will however only happen if the order is bad. With good orders, the heuristic algorithm will return a good solution and will still be decisively faster than the exact one.

This leads us to studying a new problem: the optimal order for cluster editing. Given an order on the vertices, we define its cost as the cost of the optimal clustering where clusters are made of consecutive vertices. The purpose is now to find a fast heuristic for finding an optimal order and to combine it with the clustering heuristic.

The following example illustrates that the left-to-right order does not work well on the read alignment data. Imagine we have 10 reads stemming from two different alleles (reads 1 to 5 from the first one and read 6 to 10 from the other) and that the position of the left endpoint of the reads on the reference genome happen to be close even if the reads stem from different alleles. If reads stemming from different alleles have different sizes, a good clustering method should still be able to distinguish them. Figure 2 (left) illustrates this example; reads are organised on a plane according to their positions and lengths. Such a configuration of the reads is very likely to happen in practice.

An optimal clustering would consist of the two clusters $\{1,2,3,4,5\}$ and $\{6,7,8,9,10\}$. However, the left-to-right order is [6,1,7,2,8,3,9,4,10,5], which likely leads to a suboptimal clustering.

A good way to solve this problem is as follows: start with the left vertex, and at each step, insert

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1301v1 | CC-BY 4.0 Open Access | rec: 13 Aug 2015, publ: 13 Aug 2015

in the order the unassigned vertex which is the closest from the previous one, *i.e.*, the one with the heaviest edge. In the previous example, the resulting order would be $[6,7,8,9,10,5,4,3,2,1]$. There will be n steps each consisting of investigating the $O(k)$ vertices whose associated reads overlap with the read of the previous vertex. Note that the edges we do not investigate have weight $-\infty$. If all such vertices are already assigned, choose arbitrarily which vertex will follow in the order. This algorithm is quite simple and gives significantly better results than the left-to-right order.

There are, however, still configurations where this approach produces suboptimal orders. Figure 2 (right) shows a typical problem. If we apply the closest-neighbor approach with the example of Figure 2 (right), the resulting order will be $[1,2,3,4,6,7,8,9,\cdots]$ and vertex 5 will be skipped. The algorithm will go back to vertex 5 only later and it will therefore probably end up in a singleton.

We therefore generalize the closest-neighbor approach and identify the unassigned vertex that maximizes the sum of the weight of the edges with the $h > 1$ previously assigned vertices. In practice, $h = 3$ or $h = 5$ give already good results. This leads to a runtime of $O(hkn) = O(kn)$, which is the same as the clustering algorithm we will run on the provided order.

In our final algorithm, we will run simultaneously the heuristic for best order and for best clustering with a given order: as soon as we know the vertex j, we compute $\mathrm{opt}'(j,\cdot)$ and we will use $h = \mathrm{argmin}(\mathrm{opt}'(j,\cdot))$ to choose the vertex $j+1$. Our algorithm runs in time $O(nk^2)$ which is acceptable since k can be assumed constant.

See Algorithm 3 for this. We are given a set of vertices that we number from 0 to $n-1$ from left to right, and a weight function. Computing once every value of w for overlapping reads can be done in $O(nk)$. We then can store those values in a hash table to determine $w(a,b)$ in constant time. If the pair (a,b) has no entry in the hash table, we know that the reads do not overlap and that $w(a,b) = -\infty$. We then create the following objects:

- an integer array `order` of size n. Ultimately, $\mathrm{order}(i)$ will be the i^{th} vertex of the order we will use for cluster editing.
- a boolean array `assigned` of size n initialized with false. It will indicate whether vertex i has already been assigned in order.
- an (integer list) array `neighbours` of size n, such that $\mathrm{neighbours}(a)$ contains a vertex b iff their associated reads overlap. In other words, $\mathrm{neighbours}(a)$ is the list of vertices b such that $w(a,b) \neq -\infty$. We can also create this array and the weight function w simultaneously.
- a function `select`: given an integer list l and a predicate $p : \mathbb{N} \to \{\mathrm{true}, \mathrm{false}\}$, it returns the list of all elements of l that satisfy the predicate p.
- A function `anyvertex` which returns a vertex that has not been assigned yet.
- A matrix `opt'` of size $n \times n$ and an array `opt` of size n whose purpose will be the same as previously. We will also use an array `size` of size n to keep $\mathrm{argmin}(opt'(j,\cdot))$ in memory. Thus, $\mathrm{size}(j)$+1 will indicate the size of the last cluster in the optimal clustering of the first j vertices.

As noted before, it is possible to erase $\mathrm{opt}'(j-1,\cdot)$ after the for loop from line 14 to 16. To avoid creating an $n \times n$ array, one can also create a $2 \times n$ array and replace all the $\mathrm{opt}'(i,j)$ in the algorithm with $\mathrm{opt}'(i \bmod 2, j)$. The final algorithm runs in $O(nk)$ in time and $O(n)$ in space.

EXPERIMENTAL RESULTS

Random Graphs. We generate random weighted graphs of size n by placing n points randomly on $[0,1]$. We then choose a threshold distance l, the distance between two points a and b is of course $|a-b|$ and we create the graph by composing the distance with the function $f_l : \mathbb{R}^+ \to \overline{\mathbb{R}}$ given by $a \mapsto \frac{l^2-a^2}{la}$. This function was chosen because of three intuitive properties we expected from the weight: $f_l(0) = +\infty$, $f_l(l) = 0$ and $\lim\limits_{x\to+\infty} f_l(x) = -\infty$. The values of the threshold distance l will determine the edge density of the graph and therefore the size c of the clusters. We created 1 000 random graphs of 10 000 each for each of the three parameter values $l \in \{0.1, 0.01, 0.001\}$. As expected, the average size of clusters decreases as l decreases; we observe values of 1633.7, 159.2, and 15.5, respectively. The quality of the obtained solutions was excellent: for the three values of l, the heuristic solutions were worse than the exact solution by only 0.0%, 0.04%, and 1.0% on average, in terms of the objective function value.

Venter's Genome. We evaluate the proposed algorithm on the same benchmark data set used by Marschall et al. (2012) to facilitate comparability and describe how it was created in the following. All variants reported by Levy et al. (2007) for Venter's genome—including SNPs, indels, larger insertions and deletions, and inversions—were incorporated into the reference genome to reconstruct

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1301v1 | CC-BY 4.0 Open Access | rec: 13 Aug 2015, publ: 13 Aug 2015

Algorithm 3: Final cluster editing algorithm

```
$\text{opt}'(0,0) \leftarrow 0$; $\text{opt}(0) \leftarrow 0$; $\text{size}(0) \leftarrow 0$; $\text{order}(0) \leftarrow 0$; $\text{assigned}(0) \leftarrow \text{true}$
for $j$ from 1 to $n-1$ do
    $X \leftarrow 0$; $\text{opt}'(j,0) \leftarrow X + \min_{0 \leqslant i \leqslant j-1} \text{opt}'(j-1,i)$
    $\text{candidates} = \text{select}(\text{neighbours}(order(j-1)), p : i \mapsto not(\text{assigned}(i)))$
    if candidates = [ ] then
        $\text{order}(j) \leftarrow \text{anyvertex}()$
    else
        $\text{order}(j) \leftarrow \text{argmax}(\text{candidates}, f : i \mapsto \sum_{k=0}^{\text{size}(j-1)} w(i, \text{order}(j-1-k))$
    $\text{assigned}(\text{order}(j)) \leftarrow \text{true}$
    for $i$ from 1 to $\text{size}(j-1)$ do
        $X \leftarrow X - w(j, j-i)$
        $\text{opt}'(j,i) \leftarrow X + \text{opt}'(j-1, i-1)$
    $\text{opt}(j) \leftarrow \min([0, \text{size}(j-1)], f : i \mapsto \text{opt}'(j,i))$
    $\text{size}(j) \leftarrow \text{argmin}([0, \text{size}(j-1)], f : i \mapsto \text{opt}'(j,i))$
clusters ← [ ]; $j \leftarrow n-1$
while $j > 0$ do
    $\text{clusters} \leftarrow \text{clusters} \cup \left\{ \bigcup_{k=j-\text{size}(j)}^{j} \text{order}(k) \right\}$
    $j \leftarrow j - \text{size}(j) - 1$
return clusters
```

the full genome as faithfully as possible. Each heterozygous variant was placed on a randomly picked haplotype. From the two reconstructed haplotypes, synthetic reads were generated using the simseq program (Earl et al., 2011). Simulated paired-end reads had a length of 2×100 bp and a Gaussian internal segment size distribution with mean $\mu = 112$ and standard deviation $\sigma = 15$, which is in line with current sequencing protocols. Here, we restrict our analysis to Chromosome 1, which is the largest human chromosome and represents about 9% of the DNA in human cells. We compare our results to the call sets for CLEVER (Marschall et al., 2012), Breakdancer (Chen et al., 2009), and Pindel (Ye et al., 2009) that were produced by Marschall et al. (2012). Refer to the original CLEVER paper (Marschall et al., 2012) for performance statistics for GASV (Sindi et al., 2009), Variation Hunter (Hormozdiari et al., 2009), Hydra (Quinlan et al., 2010), MoDIL (Lee et al., 2009), and SV-seq2 (Zhang et al., 2012). For comparing predicted insertions/deletions to the ground truth, we measure center distance and length difference between the two calls and consider them a match (i.e. a true positive) if both statistics are below a predefined threshold of 100 bp. Refer to (Marschall et al., 2012) for a discussion of why this criterion is more sensible than using reciprocal overlap and on why such matches are statistically significant.

Our algorithm was implemented in OCaml. The edge threshold T was set to 0.4. Chromosome 1 of the Venter data set described above gave rise to a graph with $n = 41\,437\,854$ vertices and $nk = 588\,570\,096$ edges with a weight not equal to $-\infty$. The sets of read pairs constituting the clusters where tested for jointly deviating from the background insert size distribution and false discovery rate was controlled at 0.1 using the Benjamini-Hochberg procedure, as both also done by CLEVER (Marschall et al., 2012). Processing the data took approximately 75 minutes.

Table 1 shows precision and recall for calling insertions and deletions, stratified by length. Here, *precision* is the percentage of predictions in the respective length class that matched a true event and *recall* is the percentage of true events in the respective length class that was recovered by a prediction.

One important reason for proposing cluster editing as a new model for SV finding is that CLEVER can miss SVs that do not lead to sufficiently large cliques of read alignments. This problem is caused by the binary edge model based on a hard cutoff. Here, we use a soft (i.e. weighted) edge model and allow edges to be inserted in the cluster editing process as needed, leading to larger groups of reads. Therefore, we expect a gain in power. In fact, the results displayed in Table 1 show that the recall of the approach proposed here (labeled *ClustEd*) improves on CLEVER in four categories. It even improves recall by more than 5 percent in three categories, whereas it does not deviate by more than

Table 1. Results on Chromosome 1 of Venter's genome. The algorithm described in this paper is represented as *ClustEd*. Results for insertions of lengths 250 and above are omitted since none of the tools made any calls.

	Insertions		Deletions	
Tool	Precision	Recall	Precision	Recall
Length range: 20–49 (644 true insertions, 618 true deletions)				
ClustEd	86.7	50.0	65.2	63.8
CLEVER	93.4	46.7	95.0	65.0
BreakDancer	–	7.0	89.7	7.4
PINDEL	95.3	67.9	68.8	76.1
Length range: 50–99 (153 true insertions, 125 true deletions)				
ClustEd	53.8	79.1	55.2	77.6
CLEVER	67.0	73.2	69.2	76.8
BreakDancer	93.7	57.5	96.8	51.2
PINDEL	88.9	27.5	74.4	37.6
Length range: 100–249 (90 true insertions, 63 true deletions)				
ClustEd	80.0	27.8	68.8	57.1
CLEVER	50.0	27.8	77.5	54.0
BreakDancer	49.0	25.6	63.7	49.2
PINDEL	–	3.3	65.0	15.9

	Deletions	
Tool	Precision	Recall
Length range: 250–999 (106 true insertions, 113 true deletions)		
ClustEd	92.9	69.9
CLEVER	91.9	70.8
BreakDancer	71.8	68.1
PINDEL	94.8	62.8
Length range: 1 000–50 000 (27 true insertions, 21 true deletions)		
ClustEd	77.8	66.7
CLEVER	87.5	66.7
BreakDancer	76.5	61.9
PINDEL	60.0	57.1

five percent in the remaining categories. Note that CLEVER already sets high standards in terms of recall and outperforms the other tools in all categories except for the shortest events of 20 to 49 bp. We reckon that the worse precision is highly likely due to applying CLEVER's postprocessing scheme to the output of ClustEd, which is specifically tailored towards processing maximal cliques. We expect to reach equally high precision levels by developing a similarly highly engineered statistical postprocessing schemes also for ClustEd.

We like to emphasize that, at this point, ClustEd is intended to serve as a proof-of-concept rather than a production-ready tool. Therefore, we restricted ourselves to demonstrating that (i) cluster editing is a good formalization of the variant calling problem and (ii) that the proposed algorithm provides fast and good solutions to the arising cluster editing instances.

CONCLUSION

In this paper we have presented a new algorithm by which to cluster *all*, and not just discordant, read alignments resulting from a whole-genome NGS experiment. The model is based on the problem of *weighted cluster editing*, which is NP-hard. Here, we present a heuristic algorithm that is based on an exact polynomial-time algorithm for weighted clustering in one-dimensional point graphs, whose solutions do not deviate by more than 1% from the optimal solution. It has runtime $O(nk^2)$ where n is the length of the genome and k is a bound on the size of the clusters. Our algorithm allows thus to process whole-genome experiments in short time—just a little more than an hour per chromosome. It has the potential to yield further improvements in recall, as earlier approaches suffer from too small and overlapping clusters (Marschall et al., 2012), excessive runtimes (Lee et al., 2009) or quite simply just removal of all concordant reads, which is relevant data, through a preprocessing step.

While we have presented an algorithm that makes such clusterings possible, we have not yet presented principles that address a statistically sound evaluation of the output. Instead, we have evaluated the output clusters as if they were maximal cliques, which evidently leaves substantial room for improvements.

In future work, we will focus on the development of statistical principles that render this improved evaluation of the new clusters possible. The runtime $O(nk^2)$ now also enables to process deep sequencing datasets, such as is common in the analysis of viruses and in metagenomics. There coverage reaches values of even larger than 10^4x, such that clustering read alignments of somewhat longer genomes was often impossible so far.

Acknowledgements

The authors thank COST action BM1006 SeqAhead for supporting this work and Murray Patterson for his advice on an early version of this manuscript.

REFERENCES

Ben-Dor, A., Shamir, R., and Yakhini, Z. (1999). Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297.

Böcker, S., Briesemeister, S., and Klau, G. (2011). Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica (New York)*, 60(2):316–334.

Charikar, M., Guruswami, V., and Wirth, A. (2005). Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383.

Chen, K., Wallis, J. W., McLellan, M. D., Larson, D. E., Kalicki, J. M., Pohl, C. S., McGrath, S. D., Wendl, M. C., Zhang, Q., and Locke, D. P. (2009). BreakDancer: an algorithm for high-resolution mapping of genomic structural variation.

Cirulli, E. T. and Goldstein, D. B. (2010). Uncovering the roles of rare variants in common disease through whole-genome sequencing. *Nature Reviews Genetics*, 11(6):415–425.

Clarke, J., Wu, H.-C., Jayasinghe, L., Patel, A., Reid, S., and Bayley, H. (2009). Continuous base identification for single-molecule nanopore DNA sequencing. *Nature nanotechnology*, 4(4):265–270.

Earl, D., Bradnam, K., St John, J., Darling, A., Lin, D., Fass, J., and et al. (2011). Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome research*, 21(12):2224–2241. PMID: 21926179.

Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., Peluso, P., Rank, D., Baybayan, P., Bettman, B., et al. (2009). Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133–138.

Hartuv, E., Schmitt, A. O., Lange, J., Meier-Ewert, S., Lehrach, H., and Shamir, R. (1999). An algorithm for clustering cDNAs for gene expression analysis. In *RECOMB*, pages 188–197.

Hormozdiari, F., Alkan, C., Eichler, E. E., and Sahinalp, S. C. (2009). Combinatorial algorithms for structural variation detection in high throughput sequenced genomes. In Batzoglou, S., editor, *RECOMB*, volume 5541 of *Lecture Notes in Computer Science*, pages 218–219. Springer.

Kratochvíl, J. and Krivánek, M. (1988). On the computational complexity of codes in graphs. In Chytil, M., Janiga, L., and Koubek, V., editors, *MFCS*, volume 324 of *Lecture Notes in Computer Science*, pages 396–404. Springer.

Lee, S., Hormozdiari, F., Alkan, C., and Brudno, M. (2009). MoDIL: detecting small indels from clone-end sequencing with mixtures of distributions.

Levy, S., Sutton, G., Ng, P. C., Feuk, L., Halpern, A. L., Walenz, B. P., and et al. (2007). The diploid genome sequence of an individual human. *PLoS Biol*, 5(10):e254.

Mannaa, B. (2010). Cluster editing problem for points on the real line: A polynomial time algorithm. *Inf. Process. Lett.*, 110(21):961–965.

Marschall, T., Costa, I. G., Canzar, S., Bauer, M., Klau, G. W., Schliep, A., and Schönhuth, A. (2012). CLEVER: clique-enumerating variant finder. *Bioinformatics*, 28(22):2875–2882.

Marschall, T., Hajirasouliha, I., and Schönhuth, A. (2013). MATE-CLEVER: Mendelian-inheritance-aware discovery and genotyping of midsize and long indels. *Bioinformatics*, 29(24).

Marschall, T. and Schönhuth, A. (2015). *NGS Analysis*, chapter Discovery and Genotyping of ”Twilight Zone” Deletions. Wiley.

Quinlan, A. R., Clark, R. A., Sokolova, S., Leibowitz, M. L., Zhang, Y., Hurles, M. E., Mell, J. C., and Hall, I. M. (2010). Genome-wide mapping and assembly of structural variant breakpoints in the mouse genome. *Genome research*, 20(5):623–635.

Sharan, R., Maron-Katz, A., and Shamir, R. (2003). CLICK and EXPANDER: a system for clustering and visualizing gene expression data. *Bioinformatics*, 19(14):1787–1799.

Sindi, S. S., Helman, E., Bashir, A., and Raphael, B. J. (2009). A geometric approach for classification and comparison of structural variants. *Bioinformatics*, 25(12).

The Genome of the Netherlands Consortium (2014). Whole-genome sequence variation, population structure and demographic history of the dutch population. *Nature Genetics*, 46:818–825.

van Zuylen, A. and Williamson, D. P. (2007). Deterministic algorithms for rank aggregation and other ranking and clustering problems. In Kaklamanis, C. and Skutella, M., editors, *WAOA*, volume 4927 of *Lecture Notes in Computer Science*, pages 260–273. Springer.

Wittkop, T., Emig, D., Lange, S., Rahmann, S., Albrecht, M., Morris, J. H., Böcker, S., Stoye, J., and Baumbach, J. (2010). Partitioning biological data with transitivity clustering. *Nature methods*, 7(6):419–420.

Ye, K., Schulz, M. H., Long, Q., Apweiler, R., and Ning, Z. (2009). Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads.

Zhang, J., Wang, J., and Wu, Y. (2012). An improved approach for accurate and efficient calling of structural variations with low-coverage sequence data. *BMC Bioinformatics*, 13(S-6):S6.