Bioinformatics, 31(18), 2015, 2947–2954
doi: 10.1093/bioinformatics/btv304
Advance Access Publication Date: 15 May 2015
Original Paper

Sequence analysis

Repeat- and error-aware comparison of deletions

Roland Wittler[1,*], Tobias Marschall[2,*], Alexander Schönhuth[3,†] and Veli Mäkinen[4,†]

[1]Genome Informatics, Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany, [2]Center for Bioinformatics, Saarland University and Department of Computational Biology and Applied Algorithmics, Max Planck Institute for Informatics, Saarbrücken, Germany, [3]Centrum Wiskunde & Informatica (CWI), Life Sciences Group, Amsterdam, The Netherlands and [4]Helsinki Institute for Information Technology (HIIT), Department of Computer Science, University of Helsinki, Finland

*To whom correspondence should be addressed. The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors
†The authors wish it to be known that, in their opinion, the last two authors should be regarded as Joint Last Authors
Associate Editor: John Hancock

Received on October 6, 2014; revised on May 7, 2015; accepted on May 8, 2015

Abstract

Motivation: The number of reported genetic variants is rapidly growing, empowered by ever faster accumulation of next-generation sequencing data. A major issue is comparability. Standards that address the combined problem of inaccurately predicted breakpoints and repeat-induced ambiguities are missing. This decisively lowers the quality of 'consensus' callsets and hampers the removal of duplicate entries in variant databases, which can have deleterious effects in downstream analyses.
Results: We introduce a sound framework for comparison of deletions that captures both tool-induced inaccuracies and repeat-induced ambiguities. We present a maximum matching algorithm that outputs virtual duplicates among two sets of predictions/annotations. We demonstrate that our approach is clearly superior over *ad hoc* criteria, like overlap, and that it can reduce the redundancy among callsets substantially. We also identify large amounts of duplicate entries in the Database of Genomic Variants, which points out the immediate relevance of our approach.
Availability and implementation: Implementation is open source and available from https://bitbucket.org/readdi/readdi
Contact: roland.wittler@uni-bielefeld.de or t.marschall@mpi-inf.mpg.de
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Next-generation sequencing (NGS) technology has led to generation of 'Big Data' also in biology. The rapid accumulation of NGS data has triggered the development of an equally overwhelming amount of tools for their exploration, including many tools for the prediction of structural variants, see the reviews by Alkan *et al.* (2011) and Medvedev *et al.* (2009). An immediately arising, pressing concern are cross-genome and cross-project comparability of variant call sets (e.g. from The 1000 Genomes Project Consortium, 2010; The Genome of the Netherlands Consortium, 2014). While resolution of related issues is instrumental for successful genetics research, there are only little statistically and algorithmically rigorous approaches addressing this. Here we suggest a formal framework for identifying duplicate deletion calls.

A major source of problems about NGS data formats and tool evaluation is the repetitiveness inherent to the majority of genomes (Treangen and Salzberg, 2012). The resulting read mapping ambiguity can decisively hamper the exact allocation of structural variations (SVs) as well as insertions and deletions (indels) within a genome. On top of that, another major source of prediction inaccuracies are the technical and theoretical, tool-specific limitations. These are often well-known. For instance, sequencing errors, ambiguities during the alignment of reads to a reference, or point mutations close to the breakpoints might cause split-read approaches to be inaccurate by a few bases. Paired-end mapping approaches, on the other hand, infer the deletion size according to the difference of the expected fragment length and the resulting distance of the mapped reads. Since the original fragment length is only known approximately, the deletion size predictions are less accurate than for split-read aligners. In addition, the location of the deletion needs to be confined to the region between the paired ends, which introduces further inaccuracies. See for example Alkan *et al.* (2011) for details on the different kinds of approaches; popular split-read aligners are Pindel (Ye *et al.*, 2009) and Platypus (Rimmer *et al.*, 2014), while Breakdancer (Chen *et al.*, 2009) and Clever (Marschall *et al.*, 2012) are paired-end mapping approaches.

As we will outline in the following, the simultaneous presence of repeats and (even slightly) inaccurate breakpoint predictions makes it difficult to compare two deletion calls, as done when merging call sets created by different tools (as e.g. in the recent consensus caller by Trubetskoy *et al.*, 2015) or when searching variant databases (e.g. Sherry *et al.*, 2001; Zhang *et al.*, 2006). Since indels and SVs play important roles in cancer and many other diseases (Raphael, 2012; Xi *et al.*, 2010), and in general populations (Zhang *et al.*, 2006), alleviating such issues can make a highly beneficial contribution to the field.

1.1 Example

As a simple example for the issues to be discussed, consider the toy reference genome `ACTGCTGCA`, which has `CTG` repeated two times. First, consider two tools one of which predicts `CTG` at positions 2–4 to be deleted, while the second one predicts `TGC` at positions 6–8 to be deleted. The two predictions are *equivalent* insofar as the resulting donor genome sequence is, in both cases, `ACTGCA`. Let us assume that this is the correct donor genome sequence to be predicted. The VCF format (Danecek *et al.*, 2011) addresses this by 'left-aligning' all predictions, that is, by reporting the leftmost deletion which is equivalent to the prediction made. Note that, after left-aligning, the calls of the two tools in our example are identical.

Now consider that the second tool errs, and mistakenly predicts that only nucleotides 7–8 are deleted. In this case, left-aligning the call of the second tool has no effect, because the predicted deletion differs from the repeat unit. As a result, the two predictions, although virtually identical, do not only deviate by 1 base pair in length, but also by 5 base pairs in terms of their left breakpoint. In particular, the two predictions do not overlap. Overall, induced by the combination of repeat structure and a (minor) misprediction, spotting their similarity has become more difficult.

1.2 Our contribution

We suggest a *formal model for the identification of similar deletions*, which takes both *repeat structure* and *prediction inaccuracies* into account. Furthermore, we present an efficient algorithm based on this model that allows for (i) pairwise comparison of all elements of a set of deletions and (ii) comparison of all elements of one set of deletions to all elements of a second set. The first mode of operation facilitates the identification of (clusters of) duplicate entries in databases while the second mode allows comparing deletion calls made by different tools.

In the latter case of comparing two call sets, one hopes to obtain a one-to-one mapping, that is, each deletion from one set matches exactly one deletion from the other set. In practice, however, several deletions in one set may match one or more deletions in the other, or do not match any of the other deletions at all. Also, deletions within one set may already be redundant, because they are similar or even equivalent to one another. To address this, we introduce a maximal matching based framework to distinguish all relevant cases and compute appropriate matching statistics.

As experiments, we first screen the Database of Genomic Variants (DGV, Zhang *et al.*, 2006) for duplicates, and reveal hundreds of clusters of similar deletions. We also run the four state-of-the-art deletion prediction methods Breakdancer (Chen *et al.*, 2009), Clever (Marschall *et al.*, 2012), Pindel (Ye *et al.*, 2009) and Platypus (Rimmer *et al.*, 2014) on an approved benchmarking dataset which reflects real human genome sequence context (Levy *et al.*, 2007; Marschall *et al.*, 2013). Furthermore, we demonstrate that using our comparison procedure is superior to using *ad hoc* criteria like 50% reciprocal overlap, which has been popular both in evaluating calls and merging independent callsets into a 'consensus' callset.

The developed algorithms are implemented in the easy-to-use open source software package Repeat- and Error-Aware Detection of Duplicate Indels (READDI) to compare sets of deletions given in VCF or BED format, facilitating its use in NGS data analysis pipelines and for curating databases.

1.3 Related work

Our approach on taking repeat structure into account is related to the problem of *tandem repeat finding*, see e.g. Gusfield and Stoye (2004). We use the same machinery of *Longest Common Extension* (LCE) queries (Landau and Vishkin, 1989), now for identifying leftmost and rightmost shifts for each deletion. The fact that deletions and insertions can be shifted without altering the resulting sequence has already been observed by Krawitz *et al.* (2010) and Assmus *et al.* (2013), but the connection to LCEs has been missed there.

Assmus *et al.* (2013) conducted a study of indel equivalence classes based on the same notion of equivalence as we study here for the case of deletions, i.e. taking repeat-induced equivalence into account, but not the notion of similarity we propose for inaccurate predictions. They call an indel *ambiguous* if it belongs to an equivalence class consisting of more than one element. Then they analyse ambiguous indels in the Ensembl database (Hubbard *et al.*, 2002) and reveal that the exact position of such variants can affect the functional annotation; they show examples of ambiguous indels crossing transcript boundaries, affecting start codons and splice sites, and being involved with triple-repeat diseases.

The study of Assmus *et al.* (2013) emphasizes the importance of robust methods to identify potentially identical variations in databases. We complement their study by extending the model of similarity to involve inaccurate predictions. From the algorithmic perspective, we also give a faster routine to detect equivalent deletions with the connection to longest common extension queries. We

also complement the experimental analysis by screening a different variant database, and by studying the hybrid effect of repeats and inaccurate predictions with several existing deletion prediction tools on realistic ground-truth datasets.

Rather than comparing sets of deletions directly, there exists also a quite different approach based on alignments: *Apply* each set of deletions to the reference sequence and compare the created sequences using optimal pairwise alignments. The alignment score gives a ranking for deletion prediction in the case that one of the sets corresponds to the ground truth. This approach was recently extended to the diploid ground-truth setting (Mäkinen and Rahkola, 2013) with an $O(dn)$ algorithm, where d is the unit cost edit distance between predicted sequence and ground-truth, and n is the maximum sequence length. In practice, this approach is not scalable to whole-genome comparisons (Mäkinen and Rahkola, 2013); computation took 6 to 11.5 hours, depending on prediction accuracy, only for the 63 Mbp long human chromosome 20.

Even if the alignment-based approach could be sped up with more algorithm engineering, one should notice that extracting variants from the alignment result in a canonical manner is a non-trivial problem. Placing gaps within alignments has remained one of the most prevalent computational challenges in bioinformatics: See for example Lunter *et al.* (2008) for an illustration of effects such as 'gap wander' or 'gap annihilation', all of which, of course, equally apply when aligning NGS reads. One can try to attach more biological semantics to indels by modeling them as traces of recombination events (Giegerich *et al.*, 1999), but the optimal solution of such recurrence takes cubic time.

2 Methods

We first formalize the phenomenon that deleting two different intervals from a given reference results in the same string. In the second section, we extend this concept to an error-tolerant model of similarity. After that, we introduce an algorithm to find all pairs of similar deletions from two given sets.

2.1 Model: exact case

In the following, let $|S|$ be the length of a string S. We denote deletions by intervals $[i,j]$, where $S \setminus [i,j]$ denote the string that results from removing all characters from (and including) position i to (and including) j from S.

As discussed in Section 1.1, deleting different segments of a given string can yield the same result. We call two such deletions *equivalent*.

Definition 1: Deletions $[i,j]$ and $[i',j']$ are equivalent w.r.t. a reference sequence S, written $[i,j] \Leftrightarrow [i',j']$, if and only if $S \setminus [i,j] = S \setminus [i',j']$. We further define their shift being $s([i,j],[i',j']) := |i'-i|$.

Out of a set of equivalent deletions, we are especially interested in the leftmost and rightmost representative, formally defined as follows.

Definition 2: (rightmost and leftmost shift): Let S be a reference string. Then the leftmost and rightmost shift of a deletion $[i,j]$ are defined as: $L([i,j]) := \text{argmin}\{\, i' \mid [i',j'] \Leftrightarrow [i,j]\}$ and $R([i,j]) := \text{argmax}\{\, i' \mid [i',j'] \Leftrightarrow [i,j]\}$, respectively.

Property 1: Given two deletions d_1 and d_2 of the same length, the following statements are equivalent:

$$\text{(i) } d_1 \Leftrightarrow d_2 \quad \text{(ii) } L(d_1) = L(d_2) \quad \text{(iii) } R(d_1) = R(d_2)$$

The left- or rightmost shift of a deletion $[i,j]$ can be computed quickly in practice by shifting the deletion one by one as long as $S[i-1] = S[j]$ or $S[i] = S[j+1]$, respectively. They can even be computed in constant time after preprocessing the reference:

Lemma 1: The left- or rightmost shift of a deletion can be computed efficiently by preprocessing the reference sequence to allow for constant time longest common extensions.

Proof: The longest common extension $LCE_S(i,j) = l$ is such that $S[i,i+l-1] = S[j,j+l-1]$ and $S[i+l] \neq S[j+l]$. After linear time preprocessing of S, every $LCE_S(i,j)$ can be computed in constant time (Landau and Vishkin, 1989). The rightmost shift of a deletion $[i,j]$ is $[i+l,j+l]$ where $l = LCE_S(i,j+1)$. The leftmost shift can be computed analogously. □

Due to the repetitive structure of genome sequences, large shifts are possible—even larger than the deletions, that is, there are deletions which are equivalent but not even overlapping. When comparing two sets of deletions, e.g. in benchmarking different deletion prediction tools, or searching for deletions in a database, the phenomenon of equivalent deletions has to be kept in mind. Equivalency in the exact sense is not a matter of sequencing depth, read quality or read length, but of the repeated nature of genome sequences. Additionally, since in practice, both predictions and database entries do usually not come at base pair resolution, some error-tolerance is necessary to account for equivalence.

2.2 Model: similarity

Even a deviation of only a single base in deletion breakpoint predictions can implicate or prevent that deletions are shiftable, which can lead to very large differences in the resulting repeat-corrected breakpoints. To account for such breakpoint inaccuracies, we define the *distance*

$$K([i,j],[i',j']) := |i-i'| + |j-j'|,$$

which, like all following terms, takes both left and right breakpoint equally into account. Based on this, we define *neighborhoods* around deletions as follows.

Definition 3 (k-neighborhood): The k-neighborhood of a deletion $d = [i,j]$ comprises all deletions with a distance of at most k:

$$\mathcal{N}_k(d) := \{d' \mid K(d,d') \leq k\}$$

The value k is called *neighborhood size*.

For comparing two predictions made by two different methods or called on different data, we use separate neighborhood sizes k_1 and k_2 for each of the two data sets. That means, when comparing predictions from a highly accurate method (or called from very accurate data) with less accurate ones, we can set k_1 to a smaller value than k_2.

Note that the output of some deletion callers already explicitly includes tool-specific neighborhoods (a.k.a. confidence intervals), for example, in form of windows for both position and length, or both breakpoints.

We will, in the following, base all our considerations on the k-neighborhood model from above. We do this for the sake of simplicity; our approach to similarity of deletions is generic in the choice of neighborhood definition, such that also other definitions can be used.

We say that two deletions are *similar* if there are equivalent deletions within their neighborhoods.

DEFINITION 4 ((k_1,k_2)-similarity): Deletions d_1 and d_2 are (k_1,k_2)-*similar*, if and only if there are deletions $d_1' \in \mathcal{N}_{k_1}(d_1)$ and $d_2' \in \mathcal{N}_{k_2}(d_2)$ such that d_1' and d_2' are equivalent according to Definition 1.

For one pair of deletions d_1 and d_2, there might exist several choices of $d_1' \in \mathcal{N}_{k_1}(d_1)$ and $d_2' \in \mathcal{N}_{k_2}(d_2)$ that establish (k_1,k_2)-similarity between d_1 and d_2. We summarize all such choices as

$$\mathcal{S}_{k_1,k_2}(d_1,d_2) := \{(d_1',d_2') \in \mathcal{N}_{k_1}(d_1) \times \mathcal{N}_{k_2}(d_2) \,|\, d_1' \Leftrightarrow d_2'\}.$$

To assess by how many base pairs the breakpoints of two deletions need to be corrected to render them equivalent, we define

$$K^{min}_{k_1,k_2}(d_1,d_2) := \min\{K(d_1,d_1') + K(d_2,d_2') \,|\, (d_1',d_2') \in \mathcal{S}_{k_1,k_2}(d_1,d_2)\}.$$

This quantity is instrumental for distinguishing between two factors of similarity: The *neighborhood size* that is required to overcome a possible inaccuracy of deletion predictions on the one hand, and the actual *shift* due to a repetitive structure of the sequence on the other hand. Motivated by maximum parsimony, we minimize the inaccuracy first and the shift second, describing a scenario with as few errors as possible as well as a minimum shift.

DEFINITION 5 (minimum shift): Given two (k_1,k_2)-similar deletions d_1 and d_2, their minimum shift is defined as

$$s^{min}_{k_1,k_2}(d_1,d_2) := \min\{\, s(d_1',d_2') \,|\, (d_1',d_2') \in \mathcal{S}_{k_1,k_2}(d_1,d_2) \wedge K(d_1,d_1') + K(d_2,d_2') = K^{min}_{k_1,k_2}(d_1,d_2)\,\}.$$

2.3 Matching sets of deletions

By determining similar deletions within *one* dataset, we can identify duplicate calls and merge them, e.g. by selecting one representative. To compare *two* datasets D_1 and D_2, we compute the similarity relations between all deletions in D_1 to all others in D_2. This can be modeled as a bipartite graph with node sets D_1 and D_2. Two nodes/deletions $d_1 \in D_1$ and $d_2 \in D_2$ are connected by an edge (d_1, d_2) if they are (k_1, k_2)-similar. Then, a maximum cardinality matching allows to compare or merge the two datasets: Each deletion is related to at most one in the other set. Matched deletions are in common, unmatched are unique. However, we have to keep in mind that both D_1 and D_2 might contain duplicates, i.e. we have to take (k_1, k_1)-similar deletions within D_1 and (k_2, k_2)-similar deletions within D_2 into account. These can be taken care of either prior to building the graph or in a post-processing step.

To evaluate a callset P against a given 'truth' T, e.g. calls predicted by a different tool or a collection of deletions in a database, the obtained bipartite matching can easily be interpreted as follows. Matched (k_p, k_t)-similar deletions correspond to true positives (TPs), and unmatched deletions in P and T correspond to false positives (FPs) and false negatives (FNs), respectively. To account for duplicates within the two sets, after performing the matching, we also determine all (k_p, k_p)-similar deletions in P and (k_t, k_t)-similar deletions in T. If a deletion in P is unmatched but similar to a TP, it might be considered as being a duplicate of a TP and should thus not be counted as a FP. We report those deletions as *similar positives* (SPs) and do not count them as FPs. Vice versa, we report unmatched deletions in D as *similar negatives* (SNs) and do not count them as FNs. Further, we do not count several unmatched deletions as individual false positives (or negatives) if they are similar. Instead, we report components of (k_p, k_p)-similar deletions in P as *false positive components* (FPCs) and components of (k_t, k_t)-similar deletions in T as *false negative components* (FNCs), see Figure 1. This allows us to define a duplicate-aware recall by $\frac{TP}{TP+FNC}$ and precision by $\frac{TP}{TP+FPC}$.

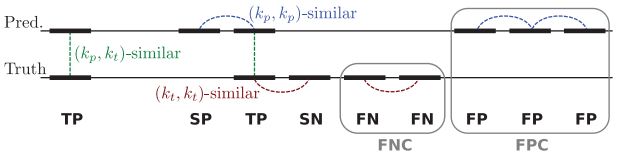

Fig. 1. Visualization of maximum matching framework. Deletions are drawn as boxes on a line representing the reference. The dashed lines indicate pairs of similar deletions

2.4 Algorithm

In the following, we describe an algorithm that, for two given lists of deletions, as specified by reference coordinates, determines all pairs of similar deletions. We assume the lists being sorted by deletion start coordinates. The algorithm outputs all pairs of deletions for which the deletion in the first list is left of or at the same position as the one in the second list. To obtain all pairs of similar deletions of two distinct lists, the algorithms is executed twice: to get the pairs where the deletion from the first list is left and from the second list is right and vice versa. To screen a single list for duplicates, the algorithm is called once by passing this list twice as parameter. Pseudo code is given in Algorithm 1.

The for-loop in line 2 iterates over all deletions in the first list. For each considered deletion d_1, we aim at finding similar deletions d_2 in the second list with a starting position right of or at the same position as d_1. How far a particular deletion can be shifted depends on whether its length matches a repeat unit. Therefore, we enumerate all possible lengths of deletions in the neighborhood of d_1 (line 3); for a given length ℓ, we compute where the rightmost such deletion starts (line 4) and use an LCE query to determine how far it can be shifted to the right (line 5). In line 6, we then iterate over all potentially similar deletions d_2 in the second list, skipping those whose neighborhood does not contain deletions of length ℓ (lines 7 and 8). In case d_2 is so far to the right that a shiftable deletion of length ℓ cannot 'mediate' a similarity relationship, we can stop and do not need to consider any deletions further to the right in the second list (lines 9 and 10). For the remaining candidates d_2, we check in lines 11 and 12 whether there is a length-ℓ deletion in its neighborhood that can establish (k_1, k_2)-similarity (cf. Definition 4): Line 11 computes the leftmost length-ℓ deletion in the neighborhood of d_2. Recall that the length-ℓ deletion starting at $start\_max_1$ is still in the neighborhood of d_1 and that all length-ℓ deletions starting between $start\_max_1$ and $shifted_1$ are equivalent. Therefore, testing for (k_1, k_2)-similarity (mediated by a length-ℓ deletion) of d_1 and d_2 boils down to a simple comparison in line 12.

The three nested loops in Algorithm 1 give rise to a worst-case runtime of $O(|L_1| \cdot |L_2| \cdot k_1)$, where $|L_1|$ and $|L_2|$ are the lengths of the two deletion lists. In practice, however, the runtime behaves linearly in $|L_1| + |L_2|$ (rather than being linear in $|L_1| \cdot |L_2|$) since for each deletion $d_1 \in L_1$, only a local neighborhood is considered for the search of candidates in L_2: the loop in line 6 only considers deletions $d_2 \in L_2$ that are right of d_1 and the search is terminated in line 10 as soon as d_2 is too far right. This 'locality' of the algorithm is achieved by considering each candidate length ℓ separately (line 3) and was a major design goal. Since no recursion stack or complex auxiliary data structures are required, the memory requirement is

mainly given by the input and output—in practice dominated by the size of the reference sequence.

Algorithm 1. Computation of (k_1, k_2)-similar deletions

Input: Two lists of deletions L_1 and L_2, sorted by deletion start positions; reference sequence S.
Output: Set $Pairs$ of all pairs of indices of (k_1, k_2)-similar deletions for which the one in the first list is left of or at the same position as the one in the second list.

```
1  Pairs ← ∅;
2  for each deletion d1 = [start(d1), end(d1)] in L1 do
3      for each possible deletion length ℓ of d1
       (i.e., ℓ = length(d1) − k1, ..., length(d1) + k1) do
4          start_max1 ← ⌊(start(d1)+end(d1)−ℓ+k1+1)/2⌋;
5          shifted1 ← start_max1
                      +LCE_S(start_max1, start_max1+ℓ);
6          for each deletion d2 = [start(d2), end(d2)] in L2
           with start(d1) ≤ start(d2) do
7              if ℓ < length(d2) − k2 or ℓ > length(d2) + k2 then
8                  continue ; // lengths incompatible
9              if shifted1 < start(d2) − k2 then
10                 break ; // no candidate in range
11             start_min2 ← ⌈(start(d2)+end(d2)−ℓ−k2+1)/2⌉;
12             if shifted1 ≥ start_min2 then
                   /* deletion of length ℓ can be
                      shifted from k1-neighborhood of
                      d1 to k2-neighborhood of d2     */
13                 Pairs ← Pairs ∪ {(d1, d2)};
14 return Pairs;
```

3 Results and discussion

3.1 Datasets

Venter predictions and venter truth. We use a benchmark dataset based on Craig Venter's genome (Levy *et al.*, 2007) that was described in Marschall *et al.* (2013). It consists of three human genomes that represent a mother-father-child trio and are realistic in particular in terms of true sequence context and amounts of variants across the whole variant spectrum (including also mixed events and inversions, for example). Here, we use the father genome. By *Venter Truth*, we refer to the set of known true differences of this genome to the reference genome. From this genome, we sample 2×100 bp reads with HiSeq error profiles to $30\times$ coverage using SimSeq (Earl *et al.*, 2011) and run Breakdancer (Chen *et al.*, 2009, version 1.4.4), Clever (Marschall *et al.*, 2012, version 2.0rc3), Pindel (Ye *et al.*, 2009, version 0.2.4 t) and Platypus (Rimmer *et al.*, 2014, version 0.7.9.1) to produce several call sets referred to as *Venter Predictions*. While Breakdancer and Clever are internal-segment-size based tools that yield rather inaccurate deletion calls, Pindel and Platypus, as split-read aligners, yield calls that are often highly accurate. In our experiments, in particular the results for the Platypus calls show a very similar behaviour to those for the Pindel set. These call sets serve as examples to demonstrate the behavior of our similarity criterion on realistic datasets produced by state-of-the-art tools. Here, we do not focus on comparing the tools' performance rates.

Database of Genomic Variants. The DGV (Zhang *et al.*, 2006) contains curated structural variations in human, detected in healthy control samples. It comprises different types of structural variations, predictions from different studies found by different techniques. We downloaded DGV variants mapped to GRCh 37 (release date 2014-10-16) and extracted deletion predictions (varianttype 'CNV' and variantsubtype 'deletion' or 'loss').

Throughout our experiments, we focus on deletions between 20 and 10 000 bp. Note that our tool is generic in deletion size. However, shorter deletions are still within reach of standard read mappers, and therefore less prone to prediction inaccuracies. Removal of duplicates among very long deletions is easy due to their size and because they are very few in number.

3.2 Removing duplicates in single datasets

We determined all (k, k)-similar deletions in Venter Truth and in the DGV for $k = 25$, and for each (k, k)-similar pair of deletions d_1, d_2 (excluding cases where $d_1 = d_2$), we computed their minimum shift $s^{\min}_{k,k}(d_1, d_2)$. In addition to about 60 s for reading the reference sequence, this took about 8 s (Venter) and 95 s (DGV) on a standard laptop without computing the minimum shift, and about 45 s (Venter) and 600 s (DGV) including the computation of the minimum shift. The peak memory usage was reached after reading the input files, where the reference sequence of about 3 GB clearly dominated all other data structures since the given 10–20 000 deletions are represented by four integers each, i.e. only some kilobytes were used in total. A general observation is that a small minimum shift often implies that the k-neighborhoods of d_1, d_2 overlap. For increasing minimum shift, however, the amount of overlapping neighborhoods necessarily decreases—for pairs with a minimum shift larger than k, by definition, a shift is required, which translates into the fact that the k-neighborhoods are disjoint. Note that a shift might not be required even if $s^{\min}_{k,k}(d_1, d_2)$ is larger than zero, because the definition of $s^{\min}_{k,k}$ prioritizes minimizing k over minimizing the shift.

Venter truth. This dataset is based on a *de novo* sequence assembly using Sanger sequencing (Levy *et al.*, 2007). Therefore it is of very high quality and presumably contains nearly no duplicates due to technical errors. Duplicates reported by our algorithm are thus likely to be false positives, that is, cases where two deletions are indeed biologically different, but are falsely assumed to be the same by our method. In total, we found 25 such pairs of similar deletions involving 50 out of 10 001 deletions in the dataset (0.5%), 32 of which (0.3%) required shifting. The fraction of 0.5% can be interpreted as an estimate of the *false positive rate* of our method. When manually inspecting the reported pairs, however, one can hypothesize that some of them are not wrongly called by our method but indeed represent artifacts in the dataset, meaning that the false positive rate could be even lower. Some such cases are listed in Section A of the Supplement. When choosing k larger than 25, the resulting rates naturally increase. For $k = 60$ and $k = 120$, we obtain 190 and 464 deletions involved in similar pairs, respectively. Depending on the application, the corresponding false positive rates of 1.9 and 4.6% may still be acceptable.

Database of genomic variants. Unlike the Venter dataset, the DGV contains massive amounts of virtual duplicates (15.8%): See Figure 2 for a histogram of the number of similar deletions versus the minimum shift observed. For about one third of the non-unique deletions, $K^{min} = 0$; for larger values of K^{min}, about 70–500 deletions have been found each. (See Supplement B for a plot of the distribution.) Since the DGV is a collection of deletions resulting from many, often large-scale studies, and entries are merged per study

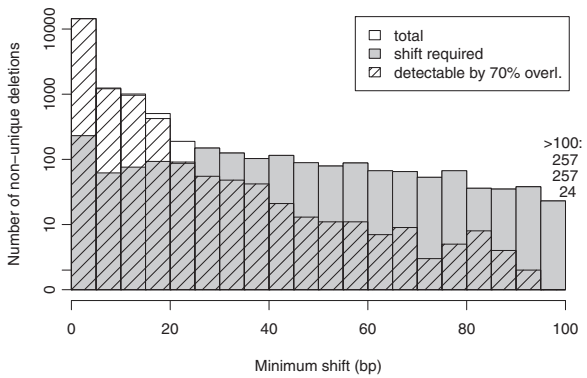

Fig. 2. Non-unique deletions in the DGV dataset. Minimum deletion size was 48 bases. Identical predictions and deletions larger than 10 K bases have been filtered out (118 349 deletions of 159 136 were left). Pairs of similar deletions with $k=25$ have been computed and these pairs involved 18 753 (15.8%) deletions, the minimum shift to any similar deletion has been recorded. Gray bars indicate those deletions that are similar with a shift required, i.e. the 25-neighborhoods of the two deletions are disjoint. The striped bars indicate those deletions which are also non-unique according to 70% overlap. (An overlap of gray and striped areas does not imply an overlap of the underlying sets of deletions.) Note the logarithmic scale of the *y*-axis

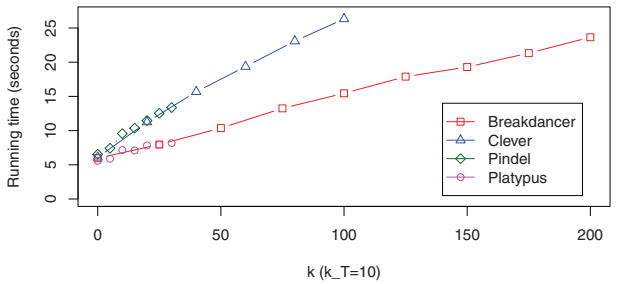

Fig. 3. Running times on a standard laptop (2 GHz, 8 GB memory) for determining similarity and performing the matching for the Venter dataset, not including ∼50 s for reading the reference sequence once

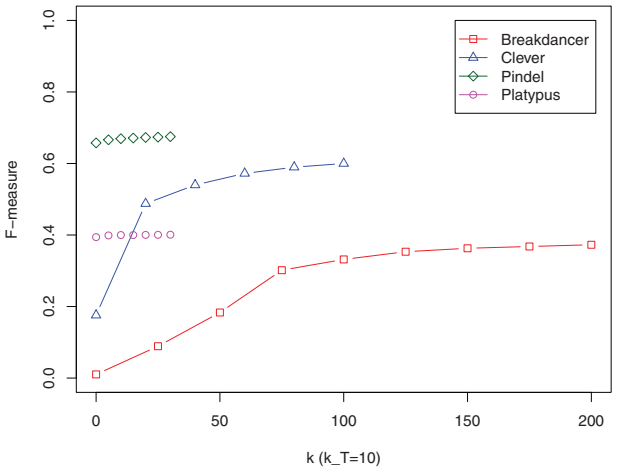

Fig. 4. F-measure for callsets of different tools compared with truth (Venter dataset). Predictions have been compared with varying neighborhood sizes to the truth with a fixed neighborhood size of $k_T=10$. Precision and recall have been determined by the matching approach described in Section 2.2 and are shown in Section D in the Supplement

only and not across studies, this high rate is expected and reflects concordance of the contained studies. But it is noteworthy that there is a large amount of deletion duplicates with large minimum shifts, which could not be detected without shifting the calls, as indicated by the gray part of the bars in Figure 2. As shown by the striped bars, about 7.3% of those deletions that we detected being non-unique cannot be detected by the criterion used in DGV, i.e. 70% overlap. The experiment described in the previous paragraph suggests that this discrepancy is unlikely caused by false positives from our approach. To collect further evidence of a low false positive rate, we analyzed pairs of matching deletions in DGV for which one deletion originated from Venter's genome and one deletion originated from the 1000 Genomes Project and observed that the 1000 Genomes deletions that are part of such a pair are strongly enriched for events called in Caucasian individuals, which is consistent with Craig Venter's ancestry (see Section C in the Supplement for details). We are thus confident that our false positive rate is low. Based on the evidence presented in this section, we conclude that ad-hoc criteria alone are insufficient when it comes to creating consensus call sets from multiple, individual call sets.

3.3 Pairwise comparison of call sets

We evaluated our matching approach by identifying duplicates among the call sets from the four tools mentioned above (Venter Predictions) and also by identifying duplicates among the individual call sets and the true annotations.

Tools versus 'Truth'. As outlined above, for each pair of sets of deletions, we consider a graph where nodes are calls and edges are drawn if two deletions are (k, k_T)-similar, where k refers to the tool in use and k_T refers to the 'Truth'. Thereby, we fix k_T to 10—since the 'Truth' has been determined experimentally, it may be prone to inaccuracies itself, which justifies a choice of k_T larger than zero. However, we vary k, so as to explore the corresponding effects for the different tools. As shown in Figure 3, the running times for determining similarity and performing the matching increase approximately linearly from about 6 s for $k=0$ for each dataset to, e.g. ∼26 s for the Clever set of size 7473 with $k=100$, not including ∼50 s for initial imports of the reference sequence. Results are shown in Figure 4 and Supplement D. We observe that, first, both precision and recall, and thus F-measure, approach a stable value for increasing k and, second, the speed and the limit of this trend differ between the tools. The two insert size based approaches, Clever and Breakdancer, reach their limits slower (in k) than Pindel and Platypus, which are split-read aligners, hence are more accurate. The limits, in turn, give evidence of the quality of the call sets.

For further analyses of the different datasets, we determined the minimum value of k (in steps of 10) for which the F-measure exceeds 95 % of its limit. This method is also included in our software package READDI. For the Clever data, we obtained $k=60$ and for Breakdancer $k=120$. For Pindel and Platypus, the above procedure yielded $k=0$. To nevertheless allow for some minimum flexibility, we chose $k=10$ for these data sets in all experiments reported below.

Choice of criteria: (k_1, k_2)-similarity versus overlap. We ran our matching algorithm both using similarity as edge criterion and defining similarity by 50% reciprocal overlap, which reflects a naive, but popular criterion for comparing results and merging duplicate calls (Lam *et al.*, 2012; Malhotra *et al.*, 2013; Teer *et al.*, 2013; The 1000 Genomes Project Consortium, 2010).

Figure 5 shows Venn diagrams for Clever and Pindel predictions. The amount of duplicates identified varies drastically between the

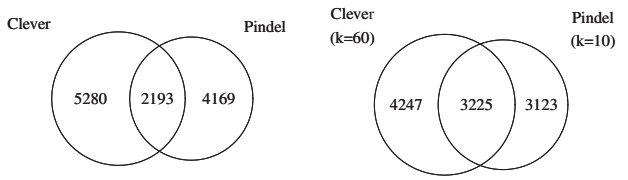

Fig. 5. Venn diagrams comparing deletion predictions on the Venter dataset by Clever and Pindel. The numbers of common and unique deletions are determined by counting the components resulting from running the matching algorithm (see Section 2.3). Diagram (a) is based on matchings of predictions where the edges are defined by a reciprocal overlap of 50% and Diagram (b) is based on similarity, where we used Word order: $k = 10$ for Pindel and $k = 60$ for Clever. Note that, due to the abovementioned usage of components, the total number of calls differs slightly between the diagrams

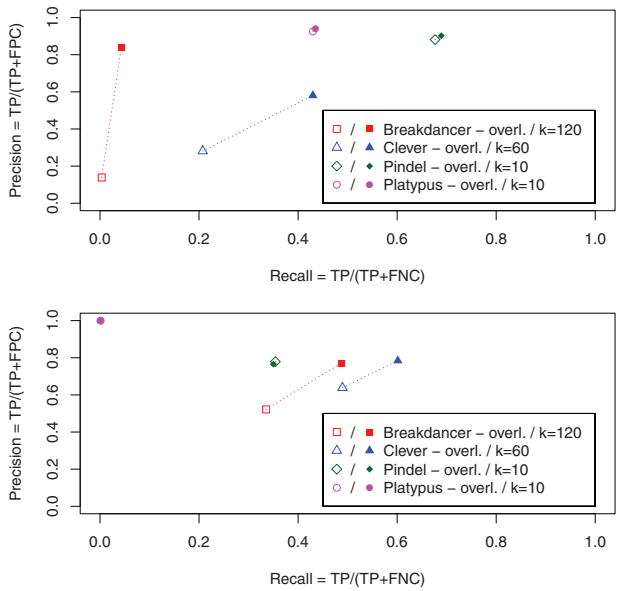

Fig. 6. Effect of using our similarity criterion instead of 50% reciprocal overlap for comparing precision and recall of different tools with respect to Venter Truth (separated by deletion size). Lines connect corresponding statistics for the two criteria. Recall has been determined by comparing all predictions to only small or large true deletions, resp. Precision has been determined by comparing only small or large predictions, resp., to all true deletions. The used definition of neighborhood for the tools is given in the legend. For 'truth', $k = 10$ has been chosen

two different criteria, where the similarity criterion detects substantially more duplicates than the overlap criterion, 3225 common calls compared with 2193 (47.1 % more). When comparing Pindel to Breakdancer and Clever to Breakdancer, we observe an increase in the number of common calls of 49.1 % and 60.0 %, respectively (see Venn diagrams in Section E of the Supplement). As we demonstrated in Section 3.2, the false discovery rate of our similarity criterion is rather negligible, which points out that overlap leaves substantial amounts of common calls unidentified.

Figure 6 shows precision/recall plots for evaluating the tools relative to the two criteria. The results demonstrate that the choice of criterion can distort qualitative statements about the accuracy of call sets substantially. For example, Breakdancer and Clever perform worse than Pindel under the overlap criterion, whereas they perform better than Pindel in the length range 41–10 K under the similarity model. These comparisons demonstrate that a sound definition of two predictions 'being the same' is crucial for the purposes of evaluating SV discovery tools.

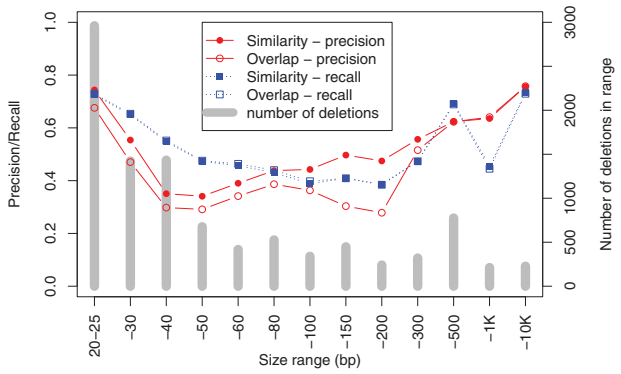

Fig. 7. Comparison of the consensus sets based on (k, k)-similarity and based on 50% reciprocal overlap to the 'truth'. Precision (TP/(TP + FP)) and recall (TP/(TP + FN)) are stated in terms of deleted bases

3.4 Model accuracy

To gather additional evidence that our model is superior to the overlap criterion, we performed the following experiment.

We computed all pairs of similar deletions within and between the call sets of the three tools Breakdancer, Clever and Pindel. For each connected component in the resulting graph of deletions, we replaced all similar deletions by one representative, favoring predictions by Pindel over those by Clever over those by Breakdancer and favoring left deletions over right. In the end, 11 095 of 16 813 deletions were left in the consensus set. We repeated the above steps defining edges by 50% reciprocal overlap instead of by our model of similarity and obtained a consensus of 12 947 deletions.

To assess the two consensus sets by a method independent of the two criteria, we compare them to the ground truth on *single-nucleotide* level. That is, for each nucleotide that is part of a deletion in each consensus set, we determine whether it is also part of a deletion in Venter Truth. On that basis, we compute recall and precision for the two consensus sets, stratified by deletion length ranges. This stratification is important as not to skew statistics towards longer deletions. Results are visualized in Figure 7.

Although the consensus set based on similarity was by 14.3 % smaller than the one based on overlap (11 095 versus 12 947 deletions), *recall* was virtually the same across all length ranges. The *precision* of our model was higher across all size ranges and the difference is especially large for medium size deletions (100–200 bases), where similarity outperforms overlap by about 20 percentage points. We hypothesize that flexibility plays an important role especially in this range, because split read approaches lose predictive power while deletions of a few hundred bases are usually detected with some inaccuracy by paired-end mapping approaches. Further, shifts can be large in this length range in comparison to the deletion size—covered by (k, k)-similarity but not by the overlap criterion.

4 Conclusions

The above analyses show that a mistaken ansatz to duplicate identification can have negative effects. We have pointed out that repeats lead to ambiguities that have the potential to introduce errors. We have also pointed out that the quality of breakpoint predictions can vary substantially among variant discovery tools, which can equally hamper the identification of duplicates among deletion calls. In the light of increasing amounts of projects that aim at large-scale discovery of genetic variants in populations or in diseased cells, sound standardization and consistency among calls reported is an urgent and vital issue.

To overcome this, we have presented a framework that accounts for repeat-aware and error-tolerant duplicate identification, hence soundly captures both error sources. We have devised a maximum matching algorithm that can make use of this framework, and which one can employ to spot duplicates among deletions. As we demonstrated, the algorithm operates at negligible false discovery rates. Thereby, it outperforms ad-hoc criteria, such as 50% reciprocal overlap substantially.

As future work, we will invest in adjusting our framework, which is generic in its core ideas, to other variant types. Including insertions, although being the reversal process of deletions, is not straightforward. The complexity arises from the difference that a deletion is uniquely described by its coordinates, but in contrast, an insertion prediction also includes the inserted sequence. In particular, allowing for small errors in the predictions is not trivial. Another interesting, and potentially promising line of research is to integrate allele frequency related statistics, likely as a postprocessing step. Moreover, it may be interesting to study selecting k not only conditioned on the method in use, but also on coverage and/or on locus, and combinations thereof, in a sound machine learning framework.

Funding

R.W. is partially funded by the International DFG Research Training Group 'Computational Methods for the Analysis of the Diversity and Dynamics of Genomes', GRK 1906/1. A.S. is funded by the Dutch Scientific Organization (NWO) through Vidi grant 639.072.309. V.M. is partially funded by the Finnish Centre of Excellence in Cancer Genetics Research (grant 250345) and the Finnish Cultural Foundation.

Conflict of Interest: none declared.

References

Alkan,C. *et al.* (2011) Genome structural variation discovery and genotyping. *Nat. Rev. Genet.*, **12**, 363–376.

Assmus,J. *et al.* (2013) Equivalent indels–ambiguous functional classes and redundancy in databases. *PLoS ONE*, **8**, e62803.

Chen,K. *et al.* (2009) Breakdancer: an algorithm for high-resolution mapping of genomic structural variation. *Nat. Methods*, **6**, 677–681.

Danecek,P. *et al.* (2011) The variant call format and VCFtools. *Bioinformatics*, **27**, 2156–2158.

Earl,D. *et al.* (2011) Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome Res.*, **21**, 2224–2241.

Giegerich,R. *et al.* (1999) An algebraic dynamic programming approach to the analysis of recombinant DNA sequences. In: *Workshop on Algorithmic Ascpects of Advanced Programming Languages (WAAAPL)*, pp. 77–88.

Gusfield,D. and Stoye,J. (2004) Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.*, **69**, 525–546.

Hubbard,T. *et al.* (2002) The ensembl genome database project. *Nucleic Acids Res.*, **30**, 38–41.

Krawitz,P. *et al.* (2010) Microindel detection in short-read sequence data. *Bioinformatics*, **26**, 722–729.

Lam,H.Y. *et al.* (2012) Detecting and annotating genetic variations using the hugeseq pipeline. *Nat. Biotechnol.*, **30**, 226–229.

Landau,G.M. and Vishkin,U. (1989) Fast parallel and serial approximate string matching. *J. Algorithms*, **10**, 157–169.

Levy,S. *et al.* (2007) The diploid genome sequence of an individual human. *PLoS Biol.*, **5**, e254.

Lunter,G. *et al.* (2008) Uncertainty in homology inferences: assessing and improving genomic sequence alignment. *Genome Res.*, **18**, 298–309.

Mäkinen,V. and Rahkola,J. (2013) Haploid to diploid alignment for variation calling assessment. *BMC Bioinformatics*, **14**, S13.

Malhotra,A. *et al.* (2013) Breakpoint profiling of 64 cancer genomes reveals numerous complex rearrangements spawned by homology-independent mechanisms. *Genome Res.*, **23**, 762–776.

Marschall,T. *et al.* (2012) CLEVER: clique-enumerating variant finder. *Bioinformatics*, **28**, 2875–2882.

Marschall,T. *et al.* (2013) MATE-CLEVER: Mendelian-inheritance-aware discovery and genotyping of midsize and long indels. *Bioinformatics*, **29**, 3143–3150.

Medvedev,P. *et al.* (2009) Computational methods for discovering structural variation with next-generation sequencing. *Nat. Methods*, **6**, S13–S20.

Raphael,B.J. (2012) Structural variation and medical genomics. *PLoS Comput. Biol.*, **8**, e1002821.

Rimmer,A. *et al.* (2014) Integrating mapping-, assembly-and haplotype-based approaches for calling variants in clinical sequencing applications. *Nat. Genet.*, **46**, 912–918.

Sherry,S. *et al.* (2001) dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res.*, **29**, 308–11.

Teer,J.K. *et al.* (2013) Massively-parallel sequencing of genes on a single chromosome: a comparison of solution hybrid selection and flow sorting. *BMC Genomics*, **14**, 253.

The 1000 Genomes Project Consortium (2010) A map of human genome variation from population-scale sequencing. *Nature*, **467**, 1061–1073.

The Genome of the Netherlands Consortium (2014) Whole-genome sequence variation, population structure and demographic history of the Dutch population. *Nat. Genet.*, **46**, 818–825.

Treangen,T. and Salzberg,S. (2012) Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nat. Rev. Genet.*, **13**, 557–567.

Trubetskoy,V. *et al.* (2015) Consensus genotyper for exome sequencing (CGES): improving the quality of exome variant genotypes. *Bioinformatics*, **31**, 187–193.

Xi,R. *et al.* (2010) Detecting structural variations in the human genome using next generation sequencing. *Brief Funct. Genomics*, **9**, 405–415.

Ye,K. *et al.* (2009) Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, **25**, 2865–2871.

Zhang,J. *et al.* (2006) Development of bioinformatics resources for display and analysis of copy number and other structural variants in the human genome. *Cytogenet. Genome Res.*, **115**, 205–214.