nFuse: Discovery of complex genomic rearrangements in cancer using high-throughput sequencing

Supplementary Text

Andrew McPherson[1], Chunxiao Wu[2], Alexander Wyatt[2], Sohrab Shah[3], Colin Collins[2] and S. Cenk Sahinalp[1]

[1]School of Computing Science, Simon Fraser University
[2]Vancouver Prostate Centre
[3]Department of Molecular Oncology, BC Cancer Research Centre

June 26, 2012

Supplemental

nFuse pipeline overview

The nFuse method builds upon Comrad (McPherson et al., 2011b), our previous work on rearrangement detection in matched RNA-seq and WGSS. We begin this section by briefly describing Comrad, then describe significant differences between Comrad and nFuse. An overview of the nFuse pipeline is shown in Figure 1.

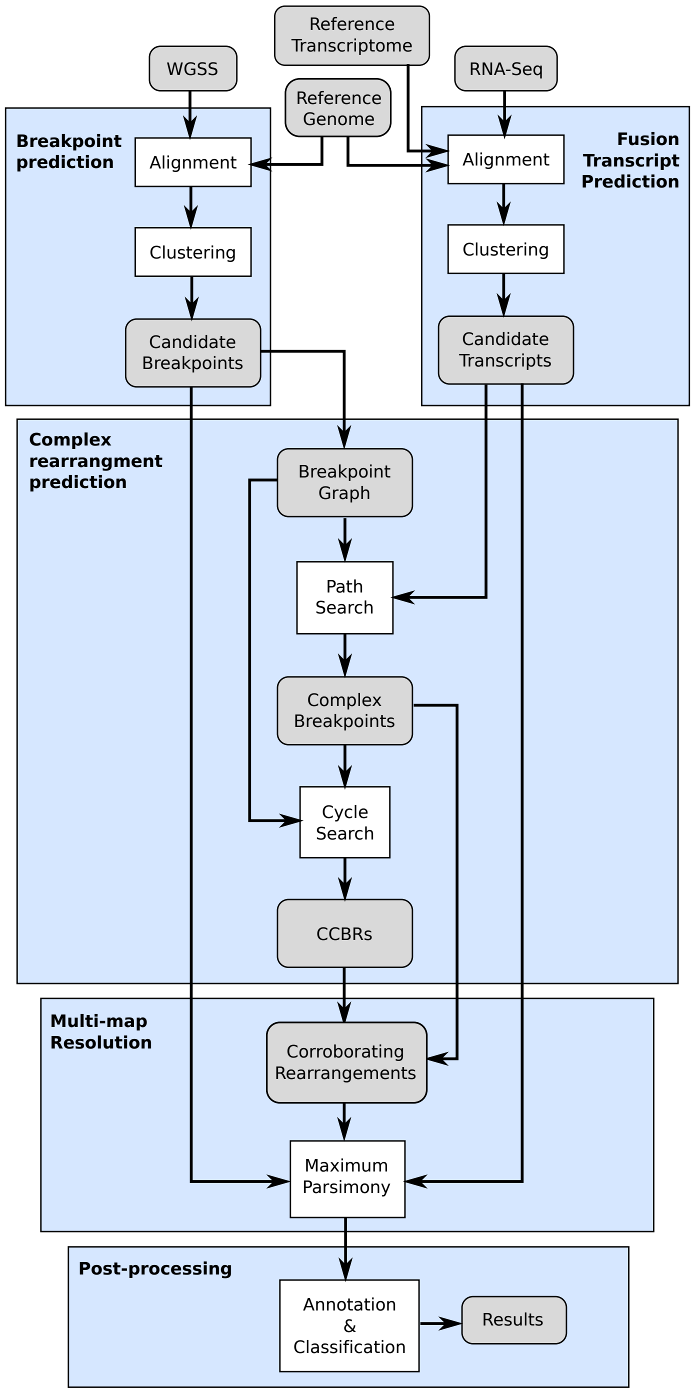


Figure 1: **Overview of the nFuse pipeline.** Generally, the nFuse pipeline involves 5 major steps: breakpoint prediction, fusion transcript prediction, complex rearrangement prediction, multi-map resolution, and post-processing. Shaded nodes represent data and unshaded nodes represent analysis.

Comrad was developed to predict fusion transcripts and their associated rearrangements from matched RNA-seq and WGSS tumour data. Comrad begins by aligning RNA-seq reads to the reference transcriptome and WGSS reads to the reference genome. Paired end RNA-seq reads for which both ends align to the same gene are classified as concordant, and all other reads are classified as discordant. Discordant RNA-seq reads are clustered into sets of reads that suggest the same candidate fusion transcripts. Paired end WGSS reads for which both ends align in the expected direction within 2kb are classified as concordant, and all other reads are classified as discordant. Discordant WGSS reads are clustered into sets of reads that suggest the same candidate breakpoint.

The key contribution provided by Comrad was a method for resolving multiply mapped RNA-seq and WGSS reads using a maximum parsimony based combinatorial formulation. Real and biologically relevant gene fusions are known to exist where one of the fusion partners shares significant sequence homology with other genes in the genome (Lee et al., 2012). These fusions may produce RNA-seq reads with ambiguous genomic origin, suggesting multiple, equally likely fusion transcripts. If a corroborating breakpoint can be predicted from WGSS data, it may be possible to identify the actual fusion transcript. For example, if A-B and A-B' are two fusion transcripts suggested by the same set of multi-mapping RNA-seq reads, an A-B' breakpoint predicted from matched WGSS data would help to identify A-B' as more likely. The reverse is also true: unambiguous RNA-seq reads can assist in predicting the correct breakpoint implied by multi-mapping WGSS reads. In fact, even for ambiguous RNA-seq and WGSS reads, it should be possible to correctly associate the RNA-seq and WGSS evidence even though the exact gene pair remains unknown. Unfortunately, naive application of Comrad to the identification of CGRs will result in over-prediction of these events.

The nFuse pipeline differs from Comrad in 4 major areas: WGSS alignment, discordant read clustering, corroboration between fusion transcripts and breakpoints, and the maximum parsimony formulation. WGSS reads are aligned using a seed and extend strategy, and the best *partial* discordant alignments of the WGSS reads are used to predict breakpoints. Discordant reads are clustered using a mixture model and the EM algorithm. nFuse adds detection of CGRs associated with fusion transcripts, replacing the Comrad method of corroborating fusion transcripts and breakpoints. Finally, nFuse incorporates a new maximum parsimony formulation for resolving multi-map reads that does not over-predict CGRs.

Partial alignments of WGSS reads

As sequencing technology improves and read lengths increase, a larger proportion of each DNA fragment is sequenced, and a smaller proportion of the fragment remains unsequenced. Thus it becomes increasingly likely that a breakpoint will fall within a sequenced region of a DNA fragment rather than the unsequenced region in the middle of a fragment. As a result we will be less likely to find a complete and contiguous alignment of both reads produced by DNA fragment harbouring a breakpoint. For instance, in HCC1954, the DNA fragments are approximately 193 bp in length, and many of the reads are 81 bp in length. If we expect complete and contiguous alignments, we will only be able to identify discordant reads for which the breakpoint is in the $193 - 2 \times 81 = 31$bp region in the middle of the read.

To mitigate the aforementioned problem, we search for *partial* alignments of discordant paired end WGSS reads. Let r be the sequence of one end of a paired end read and define the partial alignment of r as an alignment of the first ℓ nucleotides of r where $1 \leq \ell \leq |r|$. A read with a breakpoint at position $\ell + 1$ in the read should ideally produce a partial alignment of length ℓ. The score of a partial alignment is calculated using a fixed bonus for matches, an affine gap penalty, and penalty for mismatches based on the quality of the read at the mismatch. A partial alignment can be calculated using a trivial modification to the dynamic programming algorithm for calculating alignments with affine gap penalties (Gotoh, 1982).

We use bowtie2 in local alignment mode as an approximate but effective method for generating partial alignments of reads (Langmead and Salzberg, 2012). To generate the n top scoring mapping locations for a read, we first use bowtie2 with parameters `--very-sensitive-local -k` $n+1$ to calculate $n+1$ local alignments, and re-score these alignments if they include soft-clipping at the beginning of the read. We use the default scoring method implemented by bowtie2 for end-to-end alignments, re-described here. A gap of length N is given a penalty calculated as,

$$\mathrm{GO} + N \times \mathrm{GE}.$$

We use the bowtie2 defaults, GO $= 5$ and GE $= 3$. Mismatches are given a penalty calculated as,

$$\mathrm{MN} + \mathrm{floor}\left(\frac{\mathrm{MX} - \mathrm{MN}}{\frac{1}{40}\min(Q, 40.0)}\right)$$

where Q is the Phred quality value. We use the bowtie2 defaults, MN $= 2$ and MX $= 6$. Matches are given a bonus MA $= 2$, the default for bowtie2. Let $S = \{s_1, s_2, ..., s_n, s_{n+1}\}$ be the resulting set of scores and let $T = \{s_i : s_i = \max(S)\}$ be the set of scores that attain the maximum value. If $|T| = n+1$ the read is filtered, otherwise the set of alignments T is retained. By default nFuse uses $n = 20$.

Note that we are currently exploring the tradeoff between speed, accuracy and flexibility of available aligners to allow optimal performance of the nFuse breakpoint prediction.

Discordant read clustering

Discordant reads are clustered into sets of reads that support the same fusion transcript/breakpoint using Expectation Maximization applied to a mixture model. Given a breakpoint formed by joining position s in chromosome X to position t in chromosome Y, we can write the likelihood of alignment A given breakpoint $B = (s, t)$ as:

$$P(A|B) \propto \begin{cases} \mathcal{N}(s - x + t - y|\mu, \sigma) & x \leq s \text{ and } y \leq t \\ 0 & x > s \text{ or } y > t \end{cases}$$

Normalization constant W can be calculated exactly by noting that the volume under the surface defined by $P(A|B)$ is equivalent to an extrusion of the normal distribution by μ, thus $W = \mu$. The assumption that $P(A|B) = 0$ for $x > s$ or $y > t$ implies that the partial alignment process is perfect. We soften this assumption because a distribution with support over the full 2d space of alignment positions will be more conducive to an EM type algorithm. Thus define the soft boundary likelihood as follows:

$$P(A|B) \;\propto\; \mathcal{N}(s-x+t-y|\mu,\sigma^2)\cdot e^{-\lambda R(x-s)}\cdot e^{-\lambda R(y-t)}$$

Where $R(x)$ denotes the ramp function defined as follows:

$$R(x) \;=\; \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

As an approximation, we continue to use the normalization constant $W = \mu$ for the soft boundary likelihood.

Suppose now that the N paired end alignments in $\mathcal{A}$ are produced by a mixture of K breakpoints $\mathcal{B}$. At this stage K is assumed given, below we describe how we determine K. Let $z_{nk} = 1$ if and only if alignment A_n was generated by breakpoint $B_k \in \mathcal{B}$, and let $\pi_k = P(z_{nk} = 1)$. Write the log likelihood of $\mathcal{A}$ given $\mathcal{B}$ as follows:

$$\log p(\mathcal{A}|\mathcal{B}) \;=\; \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k P(A_n, z_{nk}|B_k)$$

We use EM to infer the $\mathcal{B}$, $\boldsymbol{\pi}$, and $\boldsymbol{Z}$ that maximize $\log p(\mathcal{A}|\mathcal{B})$. Finally we select K using the Bayesian Information Criterion. We start by setting $K = 1$, running EM, and calculating the BIC (Equation 1). Next we select the paired end alignment m with the minimum model probability (Equation 2). We increment K and augment the previous set of responsibilities, allowing the new cluster to take full responsibility for A_m. We redo EM, starting with the M Step, and calculate BIC for the result. The process continues until the new BIC is larger than the previous, at which point we stop iteration and select the previous solution. Lastly, we assign each paired end alignment n to the cluster k for which $P(z_{nk}|A_n, \hat{\boldsymbol{\pi}}, \mathcal{B})$ is maximum.

$$\text{BIC} = -2\log p(\mathcal{A}|\mathcal{B}) + 2K\log N \tag{1}$$

$$m = \underset{n}{\operatorname{argmin}}\, p(A'|\mathcal{B}) = \underset{n}{\operatorname{argmin}} \sum_{k=1}^{K} \pi_k p(A_n|B_k) \tag{2}$$

Corroboration between fusion transcripts and breakpoints

Corroboration between fusion transcripts and breakpoints differs significantly between nFuse and Comrad. For Comrad, fusion transcripts and rearrangements breakpoints are said to corroborate given satisfaction of two conditions that loosely determined whether a fusion transcript conceivable arose from a single breakpoint. Corroboration was determined by comparing all fusion transcripts with all breakpoints, and assessing satisfaction of the two conditions. By contrast, nFuse constructs a breakpoint graph from breakpoint predictions, including those supported by ambiguous WGSS reads. nFuse then searches for the shortest alternating path through the breakpoint graph that would corroborate each fusion transcript. As stated above, we add vertices representing the predicted fusion boundaries, and search for the shortest alternating path between those vertices. If a path exists below a given threshold score, that path is said to corroborate the fusion transcript. nFuse also searches for evidence of CCBRs associated with fusion transcripts. For each breakpoint in an alternating path corroborating a fusion transcript, nFuse searches for a CCBR that includes that breakpoint. As stated above, we first remove the breakpoint edge from the graph, then search for the shortest alternating path between the two vertices of that breakpoint. We also identify fusion transcripts with mapped genomic distance less than 200 kbp, and with an orientation suggestive of a deletion, and label these as read-throughs.

Maximum parsimony formulation for resolving multi-map reads

Let B be the set of all breakpoint predictions implied by WGSS reads G, and let F be the set of all fusion transcript predictions implied by RNA-seq reads R. Let $\mathcal{M}_G \subseteq G \times B$ be a one-to-many mapping from WGSS reads to breakpoints, and let $\mathcal{M}_R \subseteq R \times F$ be a one-to-many mapping from RNA-seq reads to fusion transcripts, each produced by the alignment and clustering process. We would like to identify $\mathcal{U}_G \subseteq \mathcal{M}_G$ and $\mathcal{U}_R \subseteq \mathcal{M}_R$, such that $\mathcal{U}_G$ and $\mathcal{U}_R$ are one-to-one (unique mappings), and such that $\mathcal{U}_G$ and $\mathcal{U}_R$ maximize parsimony. Under the assumption that fusion transcripts and breakpoints are rare, we define the $\mathcal{U}_G$ and $\mathcal{U}_R$ that maximizes parsimony as the $\mathcal{U}_G$ and $\mathcal{U}_R$ that minimizes the number of fusion transcripts and breakpoints. However, we would also like to maximize the number of CGRs we discover, without over-predicting CGRs. We do so by searching within the space of maximum parsimony solutions for a solution that also maximizes the weighted sum of discovered CGRs (herein referred to as the *CGR corroboration score*). Our formulation can thus be seen as a type of multi-objective optimization, for which the minimization of fusion transcripts and breakpoints is primary, and the maximization of the CGR corroboration score is secondary.

We define the space of maximum parsimony solutions as follows. Let $\delta_f(\mathcal{U}_R)$ indicate that $\mathcal{U}_R$ maps at least one RNA-seq read to fusion transcript f. Similarly, Let $\delta_b(\mathcal{U}_G)$ indicate that $\mathcal{U}_G$ maps at least one WGSS read to breakpoint b. The space of maximum parsimony solutions $\mathcal{P}$ is defined as given in Equation 3.

$$\begin{aligned}
\mathcal{P}_R &= \underset{\mathcal{U}_R}{\operatorname{argmin}} \sum_f \delta_f(\mathcal{U}_R) \\
\mathcal{P}_G &= \underset{\mathcal{U}_G}{\operatorname{argmin}} \sum_b \delta_b(\mathcal{U}_G) \\
\mathcal{P} &= \mathcal{P}_R \times \mathcal{P}_G \qquad (3)
\end{aligned}$$

Next we define the CGR corroboration score for paths as follows. Let Q be the set of pairs of fusion transcripts and corroborating paths. Let $\delta_p(\mathcal{U}_G)$ indicate that $\delta_b(\mathcal{U}_G) = 1$ for all breakpoints b in path p. Read-throughs are represented by empty paths and by definition $\delta_p(\mathcal{U}_G) = 1$ for read-throughs. The CGR corroboration score s_p for paths is given by Equation 4.

$$\begin{aligned} s_p(\mathcal{U}_R, \mathcal{U}_G) &= \sum_{(f,p)\in Q} w_p \cdot \delta_f(\mathcal{U}_R) \cdot \delta_p(\mathcal{U}_G) \\ &= \sum_{p} w_p \cdot \delta_p(\mathcal{U}_G) \sum_{(f,p)\in Q} \delta_f(\mathcal{U}_R) \end{aligned} \tag{4}$$

We down-weight more complex paths by defining w_p as given by Equation 5.

$$w_p = \frac{1}{1 + |p|} \tag{5}$$

Finally we define the CGR corroboration score for cycles as follows. Let c be a cycle and let $\delta_c(\mathcal{U})$ indicate that $\delta_b(\mathcal{U}) = 1$ for all breakpoints b in cycle c. The CGR corroboration score s_c for cycles is given by Equation 6.

$$s_c(\mathcal{U}_G) = \sum_{c} w_c \; \delta_c(\mathcal{U}_G) \tag{6}$$

Similar to paths, we down-weight more complex cycles by defining w_c as given by Equation 7.

$$w_c = \frac{1}{1 + |c|} \tag{7}$$

Our final optimization problem is to identify a solution to Equation 8.

$$\operatorname*{argmax}_{(\mathcal{U}_R, \mathcal{U}_G)\in\mathcal{P}} \left[s_p(\mathcal{U}_R, \mathcal{U}_G) + s_c(\mathcal{U}_G) \right] \tag{8}$$

A complete search of $\mathcal{P}$ would be prohibitively expensive for even small instances of the problem. Thus, we identify an optimal $(\mathcal{U}_R, \mathcal{U}_G)$ using a heuristic search algorithm. Our heuristic is based on the greedy set cover algorithm, and as such guarantees a worst case approximation ratio of $\mathcal{O}(\log n)$ for the problem of minimizing the number of fusion transcripts and breakpoints. No guarantee is provided for maximizing the CGR corroboration score.

The algorithm we propose alternates between minimizing the number of fusion transcripts and minimizing the number of breakpoints. The number of fusion transcripts is minimized, and the CGR corroboration score maximized, given an estimate for the value of δ_p. Next the number of breakpoints is minimized, and the CGR corroboration score maximized, given an estimate for the value of $\kappa_p = \sum_{(f,p)\in Q} \delta_f$. The value κ_p represents an estimate of the number of fusion transcripts corroborated by path p. At each iteration we re-estimate values for δ_p and κ_p.

For path p, let $\chi_p = \left(\delta_{p_1}, \delta_{p_2}, ..., \delta_{p_i}\right)$ be a sequence of δ_p values from previous solutions to the problem of minimizing breakpoints. We initialize χ_p to a sequence of m successes, placing a heavy prior on existence of each path in the form of m pseudo-counts. Furthermore, let $\psi_p = \left(\kappa_{p_1}, \kappa_{p_2}, ..., \kappa_{p_i}\right)$ be a sequence of κ_p values from previous solutions to the problem of minimizing fusion transcripts. We initialize ψ_p as follows. For each path p, identify $\text{ub}(\kappa_p)$, an upper bound on κ_p, by running Algorithm 3 (with null bonuses) for only the fusion transcripts in the set $\{f : (f, p) \in Q\}$. Initialize ψ_p as a sequence of m values $\text{ub}(\kappa_p)$, placing a heavy prior on the existence of the fusion transcripts corroborating p.

The algorithm proceeds as follows. Let Z be the set of complex rearrangements (paths and cycles).

1. Estimate $\hat{\kappa}_p = \frac{\sum_i \kappa_{p_i}}{|\psi_p|}$. Let V be a set of bonuses for rearrangements, where v_z is a bonus given for the solution that selects rearrangement z. Set the bonus v_c for cycle c as w_c and the bonus v_p for path p as $\hat{\kappa}_p w_p$. Use Algorithm 1 to select a minimal set of breakpoints while attaining the maximum total in bonuses.
   Let $C_G = \text{MinimizeBreakpoints}(G, B, Z, V, \epsilon)$. Create $\mathcal{U}_G$ from C_G by assigning each read g to the breakpoint in C_G that covers g and contains the greatest number of reads, breaking ties randomly.

2. For each path p, calculate δ_p based on $\mathcal{U}_G$ from step 1, and append it to χ_p.

3. Estimate $\hat{\delta}_p = \frac{\sum_i \delta_{p_i}}{|\chi_p|}$. Let Y be a set of bonuses for fusion transcripts, where y_f is a bonus for a solution that selects fusion transcript f. Set the bonus y_f for fusion transcript f corroborated by path p as $\hat{\delta}_p w_p$. If f is a read-through then $y_f = 1$ since $\delta_p = 1$ and $w_p = 1$ by definition for read-throughs. If f is neither a read-through or corroborated by a path, $y_f = 0$. Use Algorithm 3 to select a minimal set of fusion transcripts while attaining the maximum total in bonuses.
   Let $C_R = \text{MinimizeFusions}(R, F, Y, \epsilon)$. Create $\mathcal{U}_R$ from C_R by assigning each read r to the fusion transcript in C_R that covers r and contains the greatest number of reads, breaking ties randomly.

4. For each path p, calculate κ_p based on $\mathcal{U}_R$ from step 3, and append it to ψ_p.

5. Repeat n times and return $\mathcal{U}_G$ and $\mathcal{U}_R$ from the most recent iteration.

For the purposes of this study, we have used $m = 10$ and $n = 10$.

Algorithm 1 takes as input B, Z and V. Each element of B is a labelled set of reads representing a breakpoint. Each element of Z is a set of labels representing a set of breakpoints putatively forming a CGR. Y contains a bonus for each CGR in Z. Initially, each breakpoint is given a weight 1. The bonus v_z for CGR z is distributed evenly between each breakpoint $b \in z$ by subtracting a small bonus, $\epsilon \frac{v_z}{|z|}$ from the weight of each b. At each iteration the algorithm selects the most cost efficient breakpoint b' and adds it to the set of chosen breakpoints C. All breakpoints not in C but completely covered by $\bigcup C$ are considered *invalid.* All CGRs containing invalid breakpoints are invalid. The bonuses of invalid CGRs are removed from the CGR's remaining valid breakpoints. Finally, for each CGR z containing b', the bonus from z is redistributed to the remaining breakpoints in z that are not in C. The algorithm terminates when C covers all reads. Calculation of weights is given by Algorithm 2.

Algorithm 1 MinimizeBreakpoints(G,B,Z,V,ϵ)

```
INPUT: WGSS reads $G$
INPUT: Breakpoints $B$ as labelled sets of reads
INPUT: CGRs $Z$ as sets of breakpoints
INPUT: CGRs bonus $V$
INPUT: Small constant $\epsilon$
OUTPUT: Breakpoints $C \subseteq B$
  $C \leftarrow \emptyset$
  for all $b \in B$ do
    $w_b \leftarrow$ CalculateBreakpointWeight($b$,$B$,$Z$,$V$,$C$,$\epsilon$)
  end for
  while $\bigcup C \neq G$ do
    select $b \notin B$ that minimizes $\frac{w_b}{|b \setminus \bigcup C|}$
    $C \leftarrow C \cup \{b\}$
    for all $b' \in B : b' \neq b, b' \subseteq \bigcup C$ do
      for all $z \in Z : b' \in z$ do
        $Z \leftarrow Z \setminus \{z\}$
        for all $b'' \in z$ do
          $w_{b''} \leftarrow$ CalculateBreakpointWeight($b''$,$B$,$Z$,$V$,$C$,$\epsilon$)
        end for
      end for
      $B \leftarrow B \setminus \{b'\}$
    end for
    for all $z \in Z : b \in z$ do
      for all $b' \in z$ do
        $w_{b'} \leftarrow$ CalculateBreakpointWeight($b'$,$B$,$Z$,$V$,$C$,$\epsilon$)
      end for
    end for
  end while
  return $C$
```

```
Algorithm 2 CalculateBreakpointWeight($b$,$B$,$Z$,$V$,$C$,$\epsilon$)
INPUT: Breakpoint $b$
INPUT: Valid breakpoints $B$ as labelled sets of reads
INPUT: Valid CGRs $Z$ as sets of breakpoints
INPUT: CGR bonuses $V$
INPUT: Chosen breakpoints $C$
INPUT: Small constant $\epsilon$
OUTPUT: weight $w$
  $w \leftarrow 1$
  for all $z \in Z$ do
    if $z \subseteq B$ then
      for all $b \in z$ do
        $u = b \setminus \mathcal{C}$
        $w = w - \frac{\epsilon \cdot v_b}{|u|}$
      end for
    end if
  end for
  return $w$
```

Algorithm 3 takes as input F and Y. Each element of F is a labelled set of reads representing a fusion transcript. Y contains a bonus for each fusion transcript in F. Each fusion transcript f is given a weight $1-\epsilon \cdot y_f$. At each iteration the algorithm selects the most cost efficient breakpoint f' and adds it to the set of chosen breakpoints C. The algorithm terminates when C covers all reads.

```
Algorithm 3 MinimizeFusions($R$,$F$,$Y$,$\epsilon$)
INPUT: RNA-seq reads $R$
INPUT: Fusion transcripts $F$ as labelled sets of reads
INPUT: Fusion transcripts bonuses $Y$
INPUT: Small constant $\epsilon$
OUTPUT: Fusion transcripts $C \subseteq F$
  $C \leftarrow \emptyset$
  for all $f \in F$ do
    $w_f \leftarrow 1 - \epsilon \cdot y_f$
  end for
  while $\bigcup C \neq R$ do
    select $f \notin F$ that minimizes $\frac{w_f}{|f \setminus \bigcup C|}$
    $C \leftarrow C \cup \{f\}$
  end while
  return $C$
```

Post-processing

The result of the mult-map resolution stage of Comrad will be a set of fusion transcripts and corroborating breakpoints. Assembled sequences for breakpoint predictions are re-aligned to the reference genome using blat. Breakpoint sequences that align with 90% identity within 2kb sized genomic region are filtered, as are the associated CGRs. The fusion transcripts are further processed as for deFuse (McPherson et al., 2011a). In brief, targeted dynamic programming is used to identify split reads and assemble nucleotide level fusion transcripts. A probability is calculated for each assembled fusion transcript using a classifier trained on known positive and negative fusion transcript predictions.

Calculating breakpoint probability

We predict breakpoints from discordant paired end alignments. Our approach aims for high sensitivity by including reads with multiple genomic mappings, and reads that map only partially to the genome. To ensure adequate specificity, we calculate a probability for each breakpoint based on the alignment evidence and use that probability in downstream analysis including CGR discovery.

Let $\mathbf{R}$ be the set of paired end WGSS reads. We generate a set of mapping locations $\mathbf{M}$ for $\mathbf{R}$ using the following well established strategy (Tuzun et al., 2005; Volik et al., 2003). For each paired end read $(r_j^1, r_j^2) \in \mathbf{R}$:

1. identify a single concordant mapping location if it exists.
2. if no concordant mapping location exists:
    (a) identify the n top scoring mapping locations for r_j^1
    (b) identify the n top scoring mapping locations for r_j^2

We identify the n top scoring mapping locations for r_j^1 (and r_j^2) as follows. Let s_j be the maximum alignment score attained by *partial* alignment of read j to the genome. Let k be the number of mappings of read j that attain s_j. If $k > n$ assume the read is unmappable and filter it, otherwise retain the k mapping locations.

Let $\mathbf{m}_j \in \mathbf{M}$ be the mapping locations identified for read $(r_j^1, r_j^2) \in \mathbf{R}$. Define the following indicator variables:

$$\begin{aligned} c_j &\equiv \text{read } j \text{ is } \underline{\text{c}}\text{oncordant} \\ d_j &\equiv \text{the true alignment was } \underline{\text{d}}\text{iscovered and is in the set } \mathbf{m}_j \end{aligned}$$

We make the assumption that reads mapped concordantly by the aligner are in fact concordant (with probability 1). We filter the concordantly mapped reads to create the set of discordant reads $\mathbf{R}^d$ and set of discordant mappings $\mathbf{M}^d$. As a result, $P(c_j = 1, d_j = 1) = 0$ for the set of filtered reads. We estimate probabilities for the remaining two possibilities for the true alignment of each read:

$$\begin{aligned} P(c_j = 1|\cdot) &\equiv \text{concordant but missed by the aligner} \\ P(d_j = 1|c_j = 0, \cdot) &\equiv \text{discordant but missed by the aligner} \end{aligned}$$

We estimate $P(c_j = 1|\cdot)$ using the maximum $\underline{c}$oncordant alignment $\underline{s}$core cs_j. To calculate cs_j, we align both ends of read j to all mapping locations in the set $\mathbf{m}_j$, and set cs_j to the maximum alignment score identified by this process. We then calculate $P(c_j = 1|cs_j)$, and use it to approximate $P(c_j = 1|\cdot)$. We approximate $P(d_j = 1|c_j = 0, \cdot)$ as $P(d_j = 1|c_j = 0, as_j)$ where as_j is the $\underline{a}$lignment $\underline{s}$core for read j.

Next, we cluster the discordant alignments $\mathbf{M}^d$ based on the likelihood that a set of alignments were generated by the same breakpoint. Let the resulting clusters of alignments represent putative breakpoints. Let g_{ij} indicate that putative breakpoint i $\underline{g}$enerated read j. Assume $g_{ij} = 0$ if read j is not in the cluster that supports breakpoint i. We estimate $P(g_{ij} = 1|\cdot)$ as $P(g_{ij} = 1|nm_j, d_j = 1)$, where nm_j is the $\underline{n}$umber of alternate $\underline{m}$apping locations of read j. Under the assumption that all mapping locations discovered by the aligner are equally likely, we calculate $P(g_{ij} = 1|nm_j, d_j = 1) = \frac{1}{nm_j}$.

Finally, let b_i indicate that breakpoint i is true, let $\mathbf{G}_i$ be the set of all g_{ij} for breakpoint i, and let n_i be the number of reads that were generated by breakpoint i, that is $n_i = \sum_{g_{ij} \in \mathbf{G}_i} g_{ij}$. We estimate $P(b_i|n_i)$ and use it to estimate $P(b_i|\cdot)$ as given by equation 9.

$$\begin{aligned} P(b_i|\cdot) = \sum_{\mathbf{G}_i} P(b_i|n_i) \prod_j & P(g_{ij} = 1|nm_j, d_j = 1) \\ & \times P(d_j = 1|as_j, c_j = 0) \\ & \times P(c_j = 0|cs_j) \end{aligned} \tag{9}$$

We first describe methods for estimating the above probability distributions, then describe an algorithm for calculating $P(b_i|\cdot)$ from these distributions.

Calculating $P(c_j = 0|cs_j)$

Alignment scores are calculated using dynamic programming with affine gap penalties (Gotoh, 1982). We use the scoring function implemented for bowtie2 (Langmead and Salzberg, 2012), redescribed here. A gap of length N is given a penalty calculated as,

$$\mathrm{GO} + N \times \mathrm{GE}.$$

We use the bowtie2 defaults, GO = 5 and GE = 3. Mismatches are given a penalty calculated as,

$$\mathrm{MN} + \mathrm{floor}\left(\frac{\mathrm{MX} - \mathrm{MN}}{\frac{1}{40}\min(Q, 40.0)}\right)$$

where Q is the Phred quality value. We use the bowtie2 defaults, MN = 2 and MX = 6. Matches are not given a bonus.

We calculate the maximum concordant alignment score cs_j for discordant read j as follows. Let r_j^1 and r_j^2 be end 1 and 2 of read j, and let $\mathbf{m}_j^1$ and $\mathbf{m}_j^2$ be the discordant mappings of each end. Mappings $\mathbf{m}_j^1$ and $\mathbf{m}_j^2$ were generated by a partial alignment of r_j^1 and r_j^2. First truncate r_j^1 and r_j^2 to the maximum aligned lengths in $\mathbf{m}_j^1$ and $\mathbf{m}_j^2$ respectively, to form tr_j^1 and tr_j^2 respectively. For each mapping $m_{jk}^1 \in \mathbf{m}_j^1$ align tr_j^2 to the 1000nt region adjacent to m_{jk}^1 in the genome (downstream if the strand of m_{jk}^1 is "+", upstream if the strand of m^1_{jk} is "-"). Repeat for $m^2_{jk} \in \mathbf{m}^2_j$ and tr^1_j. Calculate cs_j as the maximum of all scores identified using the above procedure.

Letting $P(c_j = 0) = \pi_c$, we can calculate $P(c_j = 0|cs_j)$ using bayes rule.

$$P(c_j = 0|cs_j) = \frac{P(cs_j|c_j = 0)\pi_c}{P(cs_j|c_j = 0)\pi_c + P(cs_j|c_j = 1)(1 - \pi_c)}$$

We estimate $P(cs_j|c_j = 0)$ from alignments of reads to random locations in the genome. We first uniformly sample 1/1000 reads from the WGSS data. For each sampled read, we produce copies of the read truncated to lengths ranging from 20nt to the length of the read. We then align the truncated reads to genomic locations selected uniformly and at random. For each truncation length, we use the samples to calculate a density using gaussian kernel density estimation with bandwidth 1. Let f^ℓ be the resulting density for truncation length ℓ. Since cs_j is the result of multiple trials, we cannot naively use f^ℓ to calculate $P(cs_j|c_j = 0)$. We instead calculate extreme value distributions based on f^ℓ, and use these to calculate $P(cs_j|c_j = 0)$. Let t be the number of trials used to calculate maximum concordant alignment score x, and let $f^\ell_t(x)$ represent to probability of attaining maximum concordant alignment score cs_j after t trials. $f^\ell_t(cs_j)$ can be calculated from $f^\ell_1 = f^\ell$, and the cumulative density F^ℓ_1 using the following recursion.

$$f^\ell_t = f^\ell_{t-1} \cdot F^\ell_1 + F^\ell_{t-1} \cdot f^\ell_1 - f^\ell_{t-1} \cdot f^\ell_1$$

We then estimate $P(cs_j|c_j = 0) = f^\ell_t(cs_j)$.

We estimate $P(cs_j|c_j = 1)$ from concordant alignments. We first uniformly sample 1/1000 reads from the WGSS data. For each sampled read, we produce copies of the read truncated to lengths ranging from 20nt to the length of the read. We then align the truncated reads to genomic locations given by the concordant alignments of the original non-truncated reads. Given that read j is concordant, the alignment score cs_j represents the score produced by aligning read j to its single location of origin. Thus for $P(cs_j|c_j = 1)$ we do not use an extreme value distribution. Instead, for each truncation length we fit a negative binomial distribution to the samples and use that to estimate $P(cs_j|c_j = 1)$.

Finally, we estimate π_c using the EM algorithm. Let J be the number of potentially discordant reads. The expected value of the log likelihood function with respect to the conditional distribution of all $c_j \in \mathbf{C}$ given $\pi^{(t)}_c$ and $cs_j \in \mathbf{S}$ is as follows.

$$\begin{aligned} E_{\mathbf{C}|\mathbf{S},\pi^{(t)}_c}\left[\log L(\pi_c|\mathbf{C},\mathbf{S})\right] &= \sum_j (1 - P(c_j = 0|cs_j, \pi^{(t)}_c)) \cdot \log\left[(1 - \pi_c) \cdot P(cs_j|c_j = 1)\right] \\ &\quad + P(c_j = 0|cs_j, \pi^{(t)}_c) \cdot \log\left[\pi_c \cdot P(cs_j|c_j = 0)\right] \end{aligned}$$

The maximum likelihood estimates of π_c yield the following update equation.

$$\pi^{(t+1)}_c = \frac{\sum_k P(c_j = 0|cs_j, \pi^{(t)}_c)}{J}$$

Thus we calculate $P(c_j = 0|cs_j)$ and π_c by iteratively calculating $P(c_j = 0|cs_j, \pi^{(t)}_c)$ for all j (Expectation step) and using those values to calculate $\pi^{(t+1)}_c$ (Maximization step) repeating until convergence.

Calculating $P(d_j = 1|as_j, c_j = 0)$

Alignment scores as_j are calculated using dynamic programming with affine gap penalties as described in the previous section. A discordant read may have a marginal alignment score for the following reasons:

i the read is poor quality
ii the region has polymorphisms compared to the reference genome
iii the read is non-genomic
iv the read is mapped to the wrong location

We would like to distinguish the first two possibilities from the last two possibilities. We do so by calculating $P(d_j = 1|as_j, c_j = 0)$, the probability that discordant read j has a valid discordant mapping given its alignment score (referred to herein as simply $P(d_j = 1|as_j)$. We can estimate $P(as_j|d_j = 1)$ from concordant alignments similar to how we estimated $P(cs_j|c_j = 1)$ in the previous section. Unfortunately it is difficult to estimate $P(as_j|d_j = 0)$. Thus we formulate the problem of calculating $P(d_j = 1|as_j)$ as a learning problem with positive and unlabelled data and use a modified version of a previously described method (Elkan and Noto, 2008). Let indicator s_j represent whether read j has been *sampled*. Then we can write,

$$P(d_j = 1|as_j) \quad = \quad \frac{P(s_j = 1|as_j)}{P(s_j = 1|d_j = 1)}$$

We first sample m scores from the concordant alignments (see previous section), where m is the number of discordant alignments. We then build a k-nearest-neighbour classifier ($k = 0.05m$) from the *unlabelled* scores U from the discordant alignments, and the *labelled positive* scores L from the concordant alignments. The KNN classifier will yield a function $g(as_j) = P(s_j = 1|as_j)$. We then estimate $c = P(s_j = 1|d_j = 1)$ from the labelled positive scores as described previously (estimator 1 from Elkan and Noto (2008)).

$$\hat{c} \quad = \quad \frac{1}{|L|} \sum_{as \in L} g(as)$$

Calculating $P(g_{ij} = 1|nm_j, d_j = 1)$

We assume all discordant alignment mapping locations are equally likely. Thus,

$$P(g_{ij} = 1|nm_j, d_j = 1) \quad = \quad \frac{1}{nm_j}.$$

Calculating $P(b_i|n_i)$

Calculation of $P(b_i|n_i)$ is formulated as a positive unlabelled learning problem and uses a similar technique as described for calculating $P(d_j = 1|as_j, c_j = 0)$. We build a set of *labelled positives* as follows. Select 100,000 genomic positions uniformly at random, and identify the number of reads that span (spanning read count) those positions from concordant alignments. Remove positions not covered by any reads and re-sample to produce a set of

m spanning read counts, where m is the number of breakpoint predictions. Build a k-nearest neighbour classifier ($k = 0.05m$) from the *labelled positive* spanning read counts from concordant alignments and the *unlabelled* spanning read counts from the breakpoint predictions. Estimate $P(b_i|n_i)$ as described above for $P(d_j = 1|as_j, c_j = 0)$.

Calculating $P(b_i|\cdot)$

Finally, we require an efficient way of calculating a sum over all possible settings of the values of $\mathbf{G}_i$, and we do this using dynamic programming similar to previously described methods (Hormozdiari et al., 2009). Rearranging the expression for $P(b_i|\cdot)$ we obtain the following.

$$P(b_i|\cdot) \quad = \quad \sum_{n_i=0}^{|\mathbf{G}_i|} P(b_i|n_i) \sum_{\mathbf{G}_i:\sum_j g_{ij}=n_i} \prod_j P(g_{ij} = 1|nm_j, as_j, cs_j)$$

Let $f(p, q)$ be defined as follows.

$$f(p, q) \quad = \quad \sum_{\mathbf{G}_i:\sum_j g_{ij}=p} \prod_{j=1}^{q} P(g_{ij}|nm_j, as_j, cs_j)$$

Then we can calculate $f(\{0, 1, ..., |\mathbf{G}_i|\}, |\mathbf{G}_i|)$ in $O(n^2)$ time using the following recurrence.

$$\begin{aligned} f(0,0) &= 1 \\ f(-1,q) &= 0 \\ f(q+1,q) &= 0 \\ f(p,q) &= f(p-1,q-1) \cdot P(g_{ij} = 1|nm_j, as_j, cs_j) + \\ &\quad f(p,q-1) \cdot P(g_{ij} = 0|nm_j, as_j, cs_j) \end{aligned}$$

Given $f(\{0, 1, ..., |\mathbf{G}_i|\}, |\mathbf{G}_i|)$, we can calculate $P(b_i|\cdot)$ as follows.

$$P(b_i|\cdot) \quad = \quad \sum_{n_i=0}^{|\mathbf{G}_i|} P(b_i|n_i) \cdot f(n_i, |\mathbf{G}_i|)$$

Shortest alternating path algorithm

The algorithm we propose for finding the shortest alternating path follows the algorithm proposed by Brown (1974). We first create a new directed graph H from the undirected breakpoint graph G as follows. For each vertex v in the G, add two vertices, v_{in} and v_{out} in H. If (v_1, v_2) is a breakpoint edge in the G, add the edge (v_{1in}, v_{2out}) to H. If (v_1, v_2) is an adjacency edge in the G, add the edge (v_{1out}, v_{2in}) to H. We then find a shortest alternating path in G by finding a shortest path in H.

Finding the shortest path in H is a non-trivial problem. Adjacency edges connect vertex v in G to vertices that are: a) on the same chromosome as v, and b) and have opposite direction to v. Thus G and also H can be considered dense. The algorithm we propose is similar to dijkstras algorithm, however it has the benefit of being faster than dijkstras for a constrained search in a dense graph. We constrain our shortest path search in 2 ways. First, we search for paths with length less than a maximum allowable distance. Second, we place a limit on the number of vertices relaxed by the algorithm. Like dijkstras we maintain a set of vertices S for which we know the shortest path from the starting vertex s to any vertex in S. Also like dijkstras, we maintain a priority queue of the neighbours of S that are closest to vertices in S. However, we refrain from maintaining a priority queue containing *all* vertices adjacent to S. Instead, we maintain a priority queue such that for each vertex v in the priority queue, v is the vertex in $\bar{S}$ that is the closest neighbor in $\bar{S}$ of some vertex in S. We also maintain, for each vertex v in the priority queue, a set, $p(v)$ of the vertices in S for which v is the closest neighbor in $\bar{S}$.

Clearly, the vertex v_{next} at the top of the priority queue will be the same for our algorithm as for dijkstras. We relax v_{next} by adding it to S and recording the shortest path to v_{next}. We then maintain the priority queue by doing the following. For each vertex u in the set $\{v_{next}\} \cup p(v_{next})$, find the vertex v in $\bar{S}$ that is closest to u, and add v to the priority queue if it exists. For this task we require a list of the neighbours of u sorted by their distance from u. If the outgoing edges from u in H are breakpoint edges in G this is trivial, since there is only one outgoing edge. If the outgoing edges from u in H are adjacency edges in G, then we can simply use a sorted list of breakpoint positions, where the same sorted list need not be duplicated multiple times for different vertices.

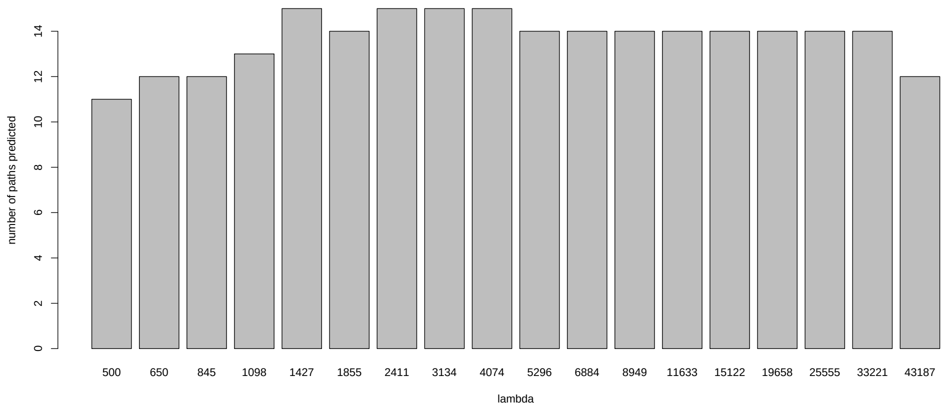
Suppose that we wish to constrain our search to a maximum of k relaxation steps. At each relaxation step we select the next closest vertex, v_{next}, an $O(\log k)$ operation that will be repeated k times. Next we iterate through the set $p(v_{next})$, finding the next closest vertices. In the worst case, at the k^{th} step, all $k-1$ vertices in S will be in the set $p(v_{next})$. Furthermore, for each of these $k-1$ vertices the next closest $k-2$ vertices will be in S. Thus for each of the $k-1$ vertices in S, the algorithm will iterate over the the other $k-2$ vertices before getting to the vertex in $\bar{S}$ that is closest. The complexity of the algorithm is dominated by this step and is worst case k^2. Thus the algorithm will be beneficial if $k \ll n$, where n is the number of vertices, since dijkstras would use best case $O(kn)$ operations.

Path search parameter β_p

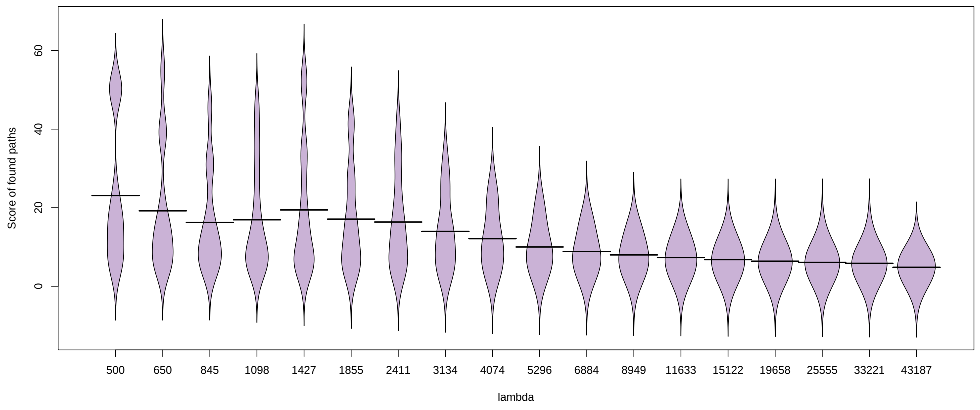
The parameter β_p can be seen as a mixing parameter for the two types of edges in the shortest alternating path optimization problem. Lower values of lambda will place more importance on the genomic distances between breakpoints in a path (adjacency edges), whereas higher values will place more importance on breakpoint probabilities.

We sought to estimate the effect of β_p on path searches using the HCC1954 data. A range of β_p values was selected, and the shortest alternating path algorithm was used to predict paths for each value of β_p. We made the strong assumption that paths were *valid* if they were composed entirely of breakpoints validated in 3 previous studies (Bignell et al., 2007; Stephens et al., 2009; Galante et al., 2011).

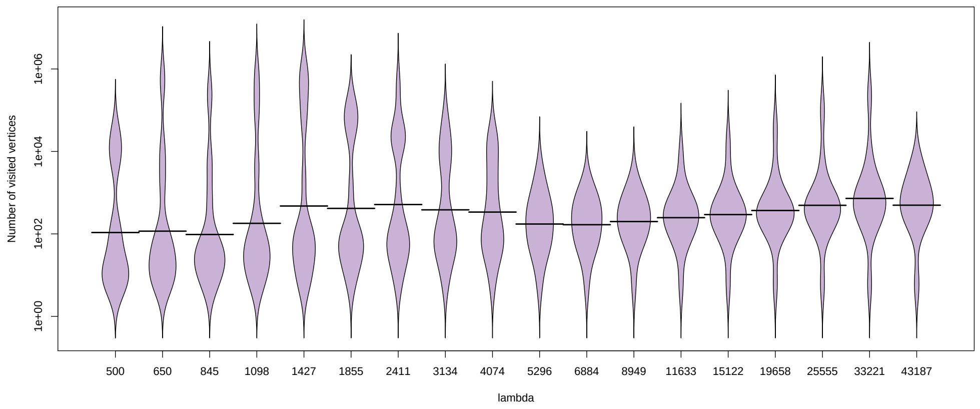
We then used the resulting *valid* paths to estimate an optimal value of β_p, and measure the effect of β_p on the shortest path algorithm. The number of valid paths ranged from 11 to 14 across the range of β_p values (Figure 2a), with the greatest number predicted between 1427 and 4074. As expected, scores for the valid breakpoints decreased almost monotonically with β_p (Figure 2b). We also analyzed the number of vertices visited by the algorithm (visit count) for each valid path, as a proxy for the time spent on the search. Interestingly, the distribution of visit counts for valid paths was bimodel for lower values of β_p, indicating that some proportion of the paths would take significantly longer to be identified at these β_p values. At a value of $\beta_p = 5296$, the distribution of visit counts becomes unimodel, and is at a local minima. We selected $\beta_p = 6884$, maximum visit count of 300,000, and maximum score of 30 based, as an adequate balance of sensitivity and balancing sensitivity and running time. Interestingly, the value we selected for β_p is close to the estimate of 6082 gained by fitting the distribution of intron lengths to an exponential distribution.



(a)



(b)



(c)

Figure 2: Statistics for paths composed of previously validated breakpoints (valid paths), as identified using a range of values for lambda. (a) Number of valid paths identified. (b) Distribution of scores for valid paths. (c) Distribution of the number of vertices visited by the algorithm for valid paths.

References

Bignell, G. R., Santarius, T., Pole, J. C., Butler, A. P., Perry, J., Pleasance, E., Greenman, C., Menzies, A., Taylor, S., Edkins, S., *et al.*, 2007. Architectures of somatic genomic rearrangement in human cancer amplicons at sequence-level resolution. *Genome Res*, **17**(9):1296–1303.

Brown, J. R., 1974. Shortest alternating path algorithms. *Networks*, **4**:311–334.

Elkan, C. and Noto, K., 2008. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 213–220, New York, NY, USA. ACM.

Galante, P., Parmigiani, R., Zhao, Q., Caballero, O., de Souza, J., Navarro, F., Gerber, A., Nicolás, M., Salim, A., Silva, A., *et al.*, 2011. Distinct patterns of somatic alterations in a lymphoblastoid and a tumor genome derived from the same individual. *Nucleic Acids Research*, **39**(14):6056–6068.

Gotoh, O., 1982. An improved algorithm for matching biological sequences. *Journal of molecular biology*, **162**(3):705–708.

Hormozdiari, F., Alkan, C., Eichler, E. E., and Sahinalp, S. C., 2009. Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes. *Genome Res*, **19**(7):1270–1278.

Langmead, B. and Salzberg, S., 2012. Fast gapped-read alignment with bowtie 2. *Nat Meth*, **9**(4):357–359.

Lee, C.-H., Ou, W.-B., Mariño-Enriquez, A., Zhu, M., Mayeda, M., Wang, Y., Guo, X., Brunner, A., Amant, F., French, C., *et al.*, 2012. 14-3-3 fusion oncogenes in high-grade endometrial stromal sarcoma. *Proceedings of the National Academy of Sciences of the United States of America*, **109**(3):929–934.

McPherson, A., Hormozdiari, F., Zayed, A., Giuliany, R., Ha, G., Sun, M., Griffith, M., Heravi Moussavi, A., Senz, J., Melnyk, N., *et al.*, 2011a. defuse: An algorithm for gene fusion discovery in tumor rna-seq data. *PLoS Comput Biol*, **7**(5):e1001138.

McPherson, A., Wu, C., Hajirasouliha, I., Hormozdiari, F., Hach, F., Lapuk, A., Volik, S., Shah, S., Collins, C., and Sahinalp, C., *et al.*, 2011b. Comrad: detection of expressed rearrangements by integrated analysis of rna-seq and low coverage genome sequence data. *Bioinformatics*, **27**(11):1481–1488.

Stephens, P. J., McBride, D. J., Lin, M. L., Varela, I., Pleasance, E. D., Simpson, J. T., Stebbings, L. A., Leroy, C., Edkins, S., Mudie, L. J., *et al.*, 2009. Complex landscapes of somatic rearrangement in human breast cancer genomes. *Nature*, **462**(7276):1005–1010.

Tuzun, E., Sharp, A. J., Bailey, J. A., Kaul, R., Morrison, V. A., Pertz, L. M., Haugen, E., Hayden, H., Albertson, D., Pinkel, D., *et al.*, 2005. Fine-scale structural variation of the human genome. *Nat Genet*, **37**(7):727–732.

Volik, S., Zhao, S., Chin, K., Brebner, J. H., Herndon, D. R., Tao, Q., Kowbel, D., Huang, G., Lapuk, A., Kuo, W. L., *et al.*, 2003. End-sequence profiling: sequence-based analysis of aberrant genomes. *Proc Natl Acad Sci U S A*, **100**(13):7696–7701.