Structural Variation 3.0 - Beta 2

plugin User Manual

User manual for

Structural Variation Plug-in 3.0 beta 2

Windows, Mac OS X and Linux

February 25, 2013

This software is for research purposes only.

CLC bio
Finlandsgade 10-12
DK-8200 Aarhus N
Denmark

Contents

1 Introduction to the Structural Variation Plug-in **4**

2 The Structural Variation Plug-in Algorithm **5**

2.1 Creating Left- and Right breakpoint signatures 5

2.2 Predicting Structural Variants 7

2.2.1 Procedure for inspecting breakpoint signatures and inferring Structural Variants 8

3 Installation of the Structural Variation Plug-in **12**

4 Uninstall **14**

Chapter 1

Introduction to the Structural Variation Plug-in

The Structural Variation Plug-in tool is designed to identify structural variants such as insertions, deletions, inversions, translocations and tandem duplications in read mappings. It relies *exclusively* on information derived from unaligned ends (also called 'soft clippings') of the reads in the mappings. This means that:

- The tool will detect NO structural variants if there are NO reads with unaligned ends in the read mapping.
- Read mappings made with the CLC 'Map reads to reference' tool with the *'global'* option switched on will have NO unaligned ends and the Structural Variation tool will thus find NO structural variants on these. (The 'global' option means that reads are aligned in their entirety - irrespectively of whether that introduces mismatches towards the ends of the reads. In the 'local' option such reads will be mapped with unaligned ends).
- Read mappings based on really short reads (say, below 35 bp) are not likely to produce many reads with unaligned ends of any useful length, and the tool is thus not likely to produce many structural variant predictions for these read mappings.
- Read mappings generated with the Large Gap Read Mapper are NOT optimal for the detection of structural variants with this tool. This is due to the fact that, the Large Gap Read Mapper will map some reads with (large) gaps, that would be mapped with unaligned ends with standard read mappers, and thus will leave a weaker unaligned end signal in the mappings for the Structural Variation Plug-in tool to work with.

In it's current version the Structural Variation Plug-in has the following known limitations:

- It will only detect intra-chromosomal structural variants.
- There is no reporting of the zygosity of the detected structural variants (there is, however, a 'variant ratio' (explained under 'Add information regarding 'zygosity'' in section 2.2.1), which can be used as a guidance for zygosity).

Chapter 2

The Structural Variation Plug-in Algorithm

The Structural Variation Plug-in algorithm has two steps. First, it identifies positions in the mapping(s) with an *excess* of reads with left (or right) unaligned ends. For each of these, it creates a Left breakpoint (LB) or Right breakpoint (RB) signature. Second, it maps the consensus unaligned ends of the identified LB and RB signatures to selected areas of the references. The mapping patterns of the consensus unaligned ends are examined and structural variant annotations consistent with the mapping patterns are created.

Below we describe the two steps of the algorithm in detail.

2.1 Creating Left- and Right breakpoint signatures

There are typically numerous reads with unaligned ends in read mappings — some are due to structural variants in the sample relative to the reference, others are due to poorly mapped, or poor quality reads. An example is given in figure 2.1. In order to make reliable predictions, attempts must be made to distinguish the unaligned ends caused by noisy read(mappings) from those caused by structural variants, so that the signal from the structural variants comes through as clearly as possible — both in terms of where the 'significant' unaligned ends are and in terms of what they look like.

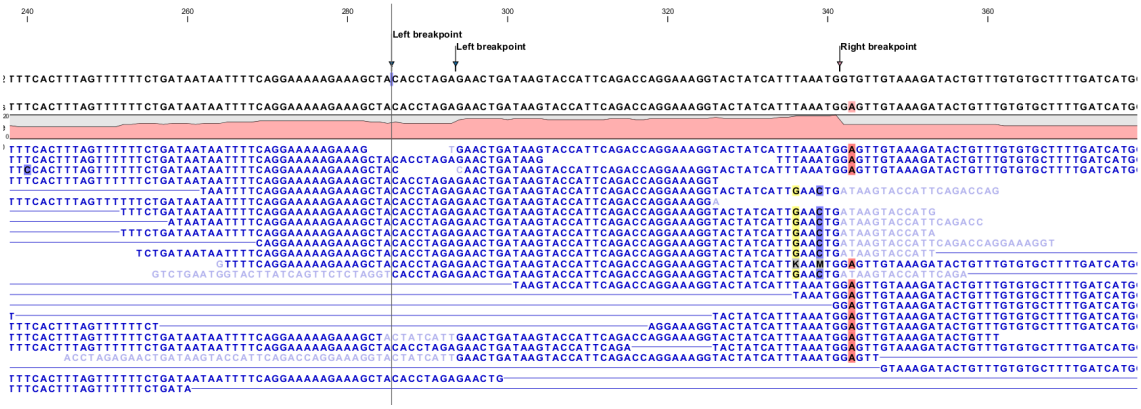

Figure 2.1: *Example of a read mapping containing unaligned ends with three unaligned end signatures.*

To identify positions with a 'significant' portion of 'consistent' unaligned end reads we first

estimate 'null-distributions' of the fractions of left and right unaligned end reads at each position in the read mapping, and subsequently use these distributions to identify positions with an 'excess' of unaligned end reads. In these positions we create a Left (LB) or Right (RB) breakpoint signature. To estimate the null-distributions we:

1. Calculate the coverage, c_i, in each position, i of all uniquely mapped reads (for paired read data sets, only intact paired reads pairs are considered — broken paired reads are ignored).
2. Calculate the coverage in each position of 'valid' reads with a starting left unaligned end, l_i (of minimum consensus length 3bp).
3. Calculate the coverage in each position of 'valid' reads with a starting right unaligned end, r_i (of minimum consensus length 3bp).

We then use the observed fractions of 'Left unaligned ends' ($\sum_i l_i / \sum_i c_i$) and 'Right unaligned ends' ($\sum_i r_i / \sum_i c_i$) as frequencies in binomial distributions of 'Left unaligned end' and 'Right unaligned end' read fractions. We go through each position in the read mapping and examine it for an excess of left (or right) unaligned end reads: if the probability of obtaining the observed number of left (or right) unaligned ends in a position with the observed coverage, is 'small', a Left breakpoint signature (LB), respectively Right breakpoint signature (RB), is created.

There are two user-specified settings, which control the significance of the LBs and RBs: 'The P-value threshold' and the 'Maximum number of mismatches' (figure 2.2). The p-value is used as a cutoff in the binomial distributions estimated above: if the probability of obtaining the observed number of left (or right) unaligned ends in a position with the observed coverage, is smaller than the user-specified cut-off, a Left breakpoint signature (LB), respectively Right breakpoint signature (RB), is created. The 'Maximum number of mis-matches' parameter is used to determine which reads are considered 'valid' unaligned end reads. Only reads that have at most this number of mis-matches in their aligned parts are counted. The higher these two values are set, the more breakpoints will be called. The more breakpoints are called, the larger the search space for the Structural variation detection algorithm, and thus the longer the computation time.

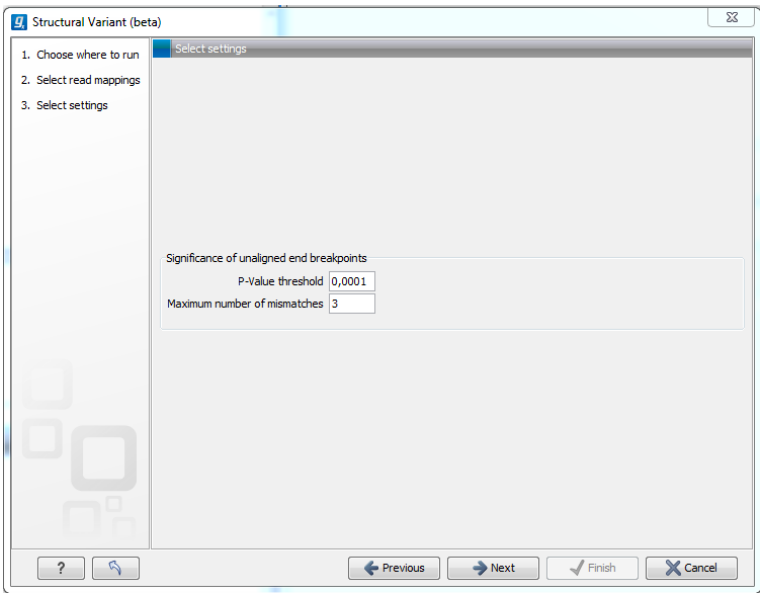

Figure 2.2: *User-specified settings that control the significance of the LBs and RBs.*

In figure 2.1, three unaligned end signatures are shown. The left-most LB signature is called only when the p-value cut-off is chosen high (0.01 as opposed to 0.0001).

2.2 Predicting Structural Variants

Having created breakpoint signatures (LBs and RBs), we use these in a procedure, which inspects the called breakpoint signatures, and attempts to match and combine them to infer possible underlying structural variant signatures. Based on the inferred structural variant signatures, structural variants are predicted, and annotations created. The procedure for inspecting breakpoint signatures and inferring structural variants is described in detail below.

The output of the Structural Variation Plug-in is a 'Structural variants' table and, optionally, a 'Signatures' table and a 'Structural variants report'. The 'Signatures' table contains the LB and RB signature annotations called, and the report gives a summary of the types of unaligned end and structural variant annotations created. When running the analysis on a reads track, the 'Structural variants' and 'Signatures' table outputs are feature tracks. When running the analysis on a stand-alone read mapping, the annotations created are added to the reference sequence of the mapping. The created tables and the mappings are 'linked' allowing the user to easily navigate between the tables and the corresponding positions in the read mapping. Figure 2.3 shows an example of the result of an analysis on a standalone read mapping (to the left) and on a reads track (to the right).

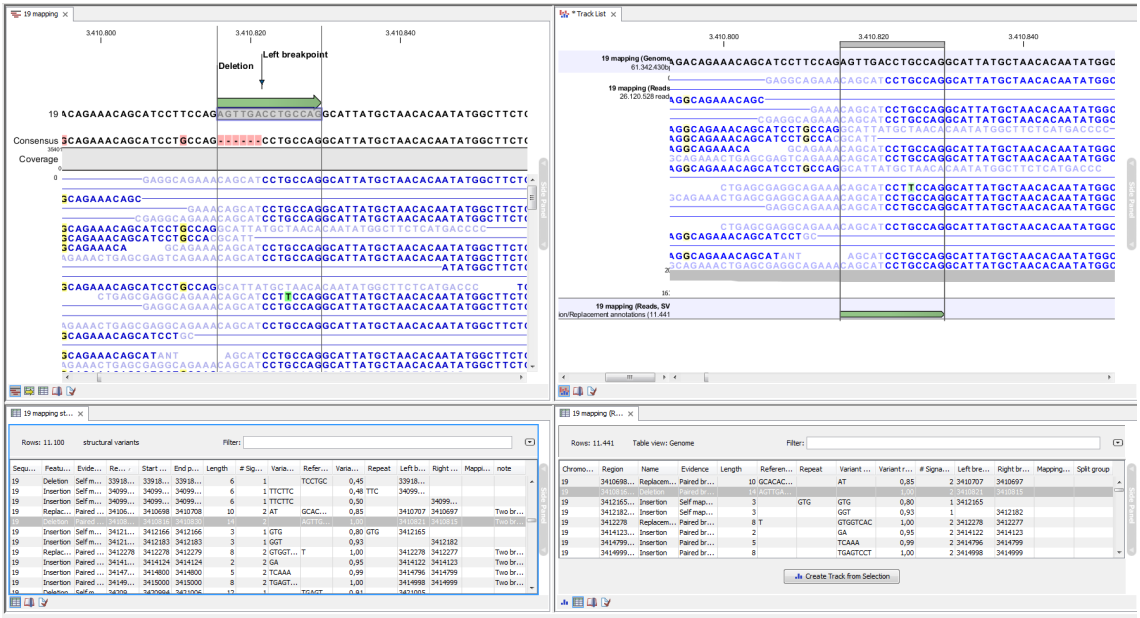

Figure 2.3: *Example of the result of an analysis on a standalone read mapping (to the left) and on a reads track (to the right).*

Typically, there will be called breakpoint signatures that are not found to stem from a structural variant. There may be a number of reasons for that: (1) the unaligned ends from which the breakpoint signature was derived might not be caused by an underlying structural variant, but merely be due to read mapping issues or noise, or (2) the breakpoint(s) which the detected breakpoint should have been matched to was/were not detected, and therefore no matching breakpoint(s) were found. Breakpoints may go un-detected either because of lack of coverage in the breakpoint region or because they are located within regions with exclusively non-uniquely

mapped reads (only unaligned ends of uniquely mapping reads are used).

2.2.1 Procedure for inspecting breakpoint signatures and inferring Structural Variants

The procedure for inspecting the created LBs and RBs and inferring structural variants from them works as follows:

1. **Calculate unaligned end consensus:** For each breakpoint, we calculate the consensus of the unaligned ends. We do this by simple alignment without gaps. Having created the consensus, we exclude the unaligned ends which differ by more than 20% from the consensus, and recalculate the consensus. This prevents 'spuriously' unaligned ends that extend longer than other unaligned ends from impacting the tail of the consensus unaligned end. We say that a consensus unaligned end 'exists' if it is > 4 nucleotides long. For some breakpoints no unaligned end consensus will exist - either because it is too short, or because there are no reads left to calculate the consensus from (e.g. there is no consensus unaligned end for a breakpoint with two reads with completely different unaligned ends, even when they are longer than 4 nucleotides).

2. **'Self-map' the unaligned end consensus:** For each breakpoint, if the consensus unaligned end 'exists' and is long enough to be meaningfully mapped (we use 14nt or longer), we map the consensus unaligned end to the reference surrounding the breakpoint (we search in a window extending from 100 bp upstream to 100 bp downstream of the breakpoint). If it maps, a note is made that it is self-mapped.

3. **'Link' breakpoint signatures:** This step serves to determine which breakpoint signatures originate from the same structural variant. To do this, we go through all breakpoints and create a 'group' of breakpoints consisting of the ones with which the breakpoint is 'linked'. There are two ways in which breakpoints can be linked: either as 'Close' breakpoints or as 'Cross-mapped' breakpoints. These linkings are defined as follows:

   (a) *'Close' breakpoints:* unaligned end breakpoints are linked as 'close breakpoints' if they are either less than 5 nucleotides or the "length of the unaligned end consensus" nucleotides apart.

   (b) *'Cross-mapped' breakpoints:* two unaligned end breakpoints, X and Y, are cross mapped if all of the following apply:

   - They are NOT 'Close' breakpoints.
   - One is an LB and the other an RB.
   - The unaligned end of X maps to the reference sequence around Y (we search in a window extending from 100 bp upstream to 100 bp downstream of the breakpoint Y)
   - The unaligned end of Y maps to the reference sequence around X (we search in a window extending from 100 bp upstream to 100 bp downstream of the breakpoint X)
   - the mappings above are on the same strand.

   Note that 'Cross-mapped' breakpoints are also created if there are multiple hits. E.g.: If we have breakpoints A, B, C, D, E and F, and if A maps B,C and D, and C maps to A, E and F, A and C will be cross mapped.

4. **'Process breakpoint groups':** Next we go through all the 'groups' of breakpoints created in the linking step above. Depending on the number (whether there is a single, two or more than two) and on the types of breakpoint signatures in a group, we do the following:

   - **One breakpoint:**
     - If the breakpoint is 'Self-mapped', create an 'Insertion (mapped)' or a 'Deletion (mapped)' signature. Whether an insertion or deletion is made, depends upon how the unaligned end self-maps to the reference.
     - If the breakpoint is not self mapped, do not create any signature.
   - **Two 'close' breakpoints:**
     - If the two breakpoints are NOT an LB and an RB: de-group the breakpoints and process the breakpoints as two single breakpoints.
     - If the two breakpoints ARE an LB and an RB: extend the two unaligned end sequences to also include 50 bp of the reference sequence respectively upstream and downstream of the unaligned end starting points. Align the resulting sequences (using the "Merge Overlapping Paired Read" aligner). Consider the consensus of this alignment:
       * if no consensus could be constructed: create an 'Insertion (close breakpoints)' signature.
       * if a consensus could be constructed: align the consensus to the reference sequence, and create a structural variant signature, depending on how the resulting alignment looks. There are three cases:
         · The alignment contains gaps in the reference: we create an 'Insertion (paired breakpoints)' signature.
         · The alignment contains gaps in the consensus: we create a 'Deletion (close breakpoints)' signature.
         · The alignment contains mismatches: we create a 'Replacement (paired breakpoints)' If the 'Replacement" has the same length in the reference and consesus and the length is significant. If the replacements is a reverse complement of itself: we create 'Inversion (paired breakpoints)' signature.
         · if the consensus is smaller than the reference: create a 'Deletion (close breakpoints)' signature.
         · if the consensus is larger than the reference: create an 'Insertion (paired breakpoints)' signature.
         · if a part of the consensus doesn't match the reference and the consensus is a perfect and 'significant' inversion of the reference: create an 'Inversion (paired breakpoints)' signature.
         · if a part of the consensus doesn't match the reference and the consensus is NOT a perfect and 'significant' inversion of the reference: create a 'Replacement (paired breakpoints)' signature.

       (For 'significance' we use the length of the inverted sequence n, and estimate the probability that the replaced bases occur in exactly the 'inversion' order as $1/n!$ If this value is smaller than the user-specified p-value cut-off, the inversion is said to be 'significant'. The default p-value of 0.0001 requires the length of the inverted sequence to be at least 8, in order to deem it 'significant' and thus call an 'inversion' rather than a 'replacement').

- **Two 'cross mapped' breakpoints:**
  - If the RB lies left of the LB we create a 'Deletion (cross-mapped)' signature.
  - If the LB lies left of the RB we create a 'Tandem duplication' signature.
  - If the mappings are found on the reverse complement strand we create an 'Inversion' signature.
- **One close and one cross mapped breakpoint:** Not possible as we do not attempt to cross map breakpoints to close breakpoints.
- **More than two breakpoints:** Groups with more than two breakpoints represent structural variants with more complicated unaligned end breakpoint signatures. Each group is processed as follows:
  - If there are two or more 'cross mapped' breakpoints in the group, we take the 'cross mapped' pair that has the shortest distance between them and process them as described under 'Two cross mapped breakpoints'. Others are ignored. This will create a 'Deletion (cross mapped)', 'Tandem duplication' or 'Inversion' signature. Add this to the group.
  - All pairs of 'close' breakpoints in the group — EXCEPT those (if any) that were also the two chosen 'cross' mapped breakpoints in above — are processed as 'Two close breakpoints'. This will create deletion, insertion or replacement signatures. Add these to the group.
  - If the group, after the above processing, has a deletion and an insertion signature: create a 'Translocation' signature.
  - For remaining groups: create a 'Complex variant' signature.

5. **Create structural variant features:** For each structural variant signature present after step 4 - except those that extend over too large a region! - , create a structural variant feature. For those extending over too large a region, visualization is challenging and we instead add multiple features - one for each 'end' of the variant. To allow the user to see which of these 'split features' belong together, we give features that belong to the same structural variant a common 'split group' identifier.

6. **Add repeat information:** Augment the predicted structural variants with repeat information: For structural variants for which we have identified the variant sequence (there are some, e.g. larger insertions, for which this is not possible) we attempt to identify if the variant sequence contains (perfect) repeats. We do this by searching the region around the structural variant for perfect repeat sequences. The region searched is 3 times the length of variant around the insertion/deletion point.

7. **Add information regarding 'zygosity':** Augment the predicted breakpoints and structural variants with information related to zygosity: For all unaligned end breakpoints we examine all the reads that cover the breakpoint. We count the number of reads that are mapped across this position and have either an unaligned end or insertions or deletions in their mappings. We call these the 'Non-perfect mapped reads'. The remaining mapped reads that cover the position (that is, those that are mapped in their entire length, and for which there are no insertions or deletions in the alignment) are called 'Perfect mapped reads'. For breakpoints we report the fraction of the 'Non-perfect mapped' reads among all mapped reads (that is: 'Non-perfect mapped'/('Non-perfect mapped'+'Perfect mapped')). For a Structural variant that is generated from more breakpoints, we sum the number of reads for the individual breakpoints and report the fraction of these in the 'Variant ratio' column.

This fraction is intended to give some idea of the zygosity of the breakpoint or structural variant. The closer the value to 1, the higher the likelihood that the variant is homozygous. However, as the 'Non-perfect mapped reads' include all reads with any unaligned end or any insertion or deletion, and not only the ones that are in perfect accordance with the breakpoint or structural variant called, it is NOT a perfect indicator of zygosity. It does, however, often give a good indication.

Chapter 3

Installation of the Structural Variation Plug-in

The Structural Variation Plug-in is installed as a plugin. Plug-ins are installed using the plug-in manager[1]:

Help in the Menu Bar | Plug-ins and Resources... ()

or **Plug-ins () in the Toolbar**

The plug-in manager has four tabs at the top:

- **Manage Plug-ins.** This is an overview of plug-ins that are installed.
- **Download Plug-ins.** This is an overview of available plug-ins on CLC bio's server.
- **Manage Resources.** This is an overview of resources that are installed.
- **Download Resources.** This is an overview of available resources on CLC bio's server.

To install a plug-in, click the **Download Plug-ins** tab. This will display an overview of the plug-ins that are available for download and installation (see figure 3.1).

Clicking a plug-in will display additional information at the right side of the dialog. This will also display a button: **Download and Install**.

Click the Structural Variation Plug-in and press **Download and Install**. A dialog displaying progress is now shown, and the plug-in is downloaded and installed.

If the Structural Variation Plug-in is not shown on the server, and you have it on your computer (e.g. if you have downloaded it from our web-site), you can install it by clicking the **Install from File** button at the bottom of the dialog. This will open a dialog where you can browse for the plug-in. The plug-in file should be a file of the type ".cpa".

When you close the dialog, you will be asked whether you wish to restart the CLC Workbench. The plug-in will not be ready for use before you have restarted.

[1] In order to install plug-ins on Windows Vista, the Workbench must be run in administrator mode: Right-click the program shortcut and choose "Run as Administrator". Then follow the procedure described below.

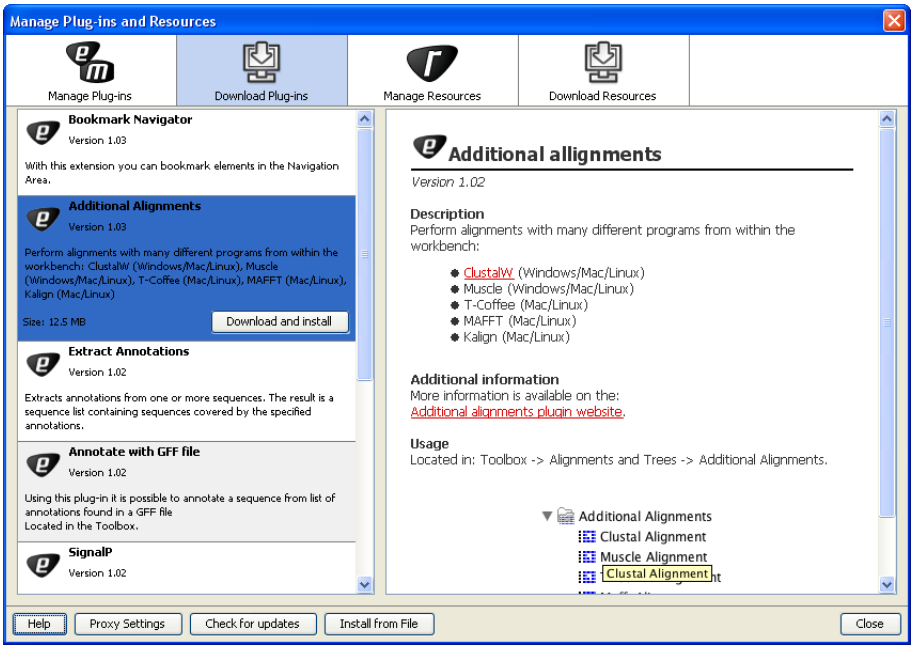

Figure 3.1: *The plug-ins that are available for download.*

Chapter 4

Uninstall

Plug-ins are uninstalled using the plug-in manager:

Help in the Menu Bar | **Plug-ins and Resources... (**  **)**

or **Plug-ins (**  **) in the Toolbar**

This will open the dialog shown in figure 4.1.

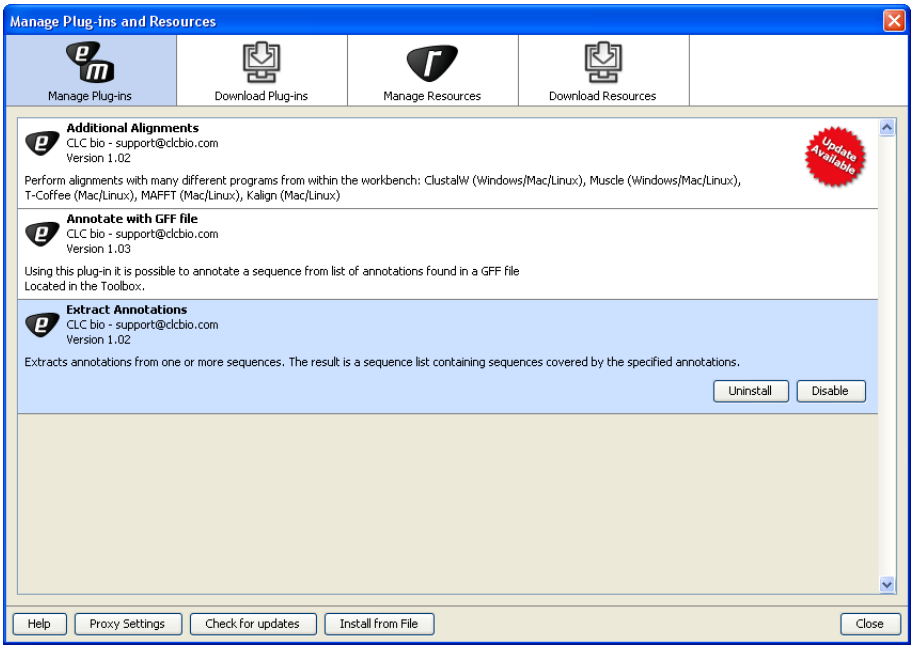

Figure 4.1: *The plug-in manager with plug-ins installed.*

The installed plug-ins are shown in this dialog. To uninstall:

Click the Structural Variation Plug-in | **Uninstall**

If you do not wish to completely uninstall the plug-in but you don't want it to be used next time you start the Workbench, click the **Disable** button.

When you close the dialog, you will be asked whether you wish to restart the workbench. The plug-in will not be uninstalled before the workbench is restarted.