GEP Misassembly Tools User Guide

Developed by Wilson Leung and Chris Shaffer

Abstract
The Drosophila ananassae sequence improvement projects derived from the Muller F element appear to have a large number of misassemblies. In response, the Genomics Education Partnership (GEP) has developed a suite of command-line tools to assist GEP faculty and students participating in the sequence improvement of these projects. In many cases, the initial assembly may be missing a large number of reads because of the misassembly and the finisher will need to identify and retrieve additional reads from the NCBI Trace Archive. In addition to providing installation instructions and brief synopses for each tool, this User Guide also provides two case studies that illustrate how you can apply the suite of GEP tools to help resolve misassemblies.

Table of Contents
Abstract .................................................................................................................................................. 1
Introduction ........................................................................................................................................ 3
Resources ........................................................................................................................................... 3
Prerequisites ......................................................................................................................................... 3
Installing the GEP misassembly tools .............................................................................................. 4
Summary of the tools in the GEP misassembly suite ........................................................................ 4
Case Study 1: Resolve a misassembly by finding reads with missing mate pairs .......................... 5
  Using restriction digests to identify potential misassemblies ....................................................... 5
  Identify regions that may have been misassembled ..................................................................... 8
  Tell phrap not to overlap regions with multiple high quality discrepancies ............................... 8
Retrieve missing mate pair reads from the NCBI Trace Archive .................................................. 10
  Step 1: Generate the list of read names to check for mate pairs .................................................. 11
  Step 2: Generate a query for clones with missing mate pairs .................................................... 13
  Step 3: Retrieve the traces from the NCBI Trace Archive ............................................................ 13
  Step 4: Rename and copy the new traces into chromat_dir ....................................................... 14
  Step 5: Incorporate the new reads into our consed database ..................................................... 15
Examining the assembly after incorporating the new reads ............................................................. 15
Case Study 2: Retrieve traces with blastn against the NCBI Trace Archive .................................. 20
  Using restriction digests to identify potential misassemblies .................................................... 20
  Using repeat tags to identify regions that are likely to be misassembled .................................. 22
  Retrieve the missing mate pairs of all unpaired reads ............................................................... 24
Incorporate additional unpaired reads into the project ............................................. 25
Retrieve additional reads through blastn searches against the Trace Archive ................. 28
Determine the locations of all the new reads added to the assembly ............................. 33
Continue with blastn searches to retrieve more reads from the Trace Archive ................. 34
Using the restriction digests to estimate the amount of missing data ......................... 38
Handling blastn results with multiple partial matches ............................................. 39
Use Miniasm to help resolve the misassembly ......................................................... 42
Use restriction digests again to estimate the amount of data still missing .................... 50
Incorporate additional traces into the project with Add New Reads ............................... 51
Confirm the final assembly with restriction digests ................................................... 54
Further investigation into the misassembly ............................................................... 56

Conclusions .............................................................................................................. 58

Appendix 1. Installing the GEP misassembly tools ..................................................... 59
   The install_misassembly_tools.sh script ............................................................... 59
   Compiling the D source files .............................................................................. 60

Appendix 2: Detailed descriptions of the GEP misassembly tools ............................... 61
   add_trace_archive_reads.sh ............................................................................. 61
   rename_traces .................................................................................................. 61
   build_unpaired_cloneid_query.pl ................................................................... 62
   calc_total_fragSizes.pl ..................................................................................... 63
   extract_read_names_from_ace.pl ...................................................................... 64
   find_reads_in_ace_file.pl .................................................................................. 65
   get_mate_pair_ti.sh ........................................................................................... 66
   build_mate_pair_query ....................................................................................... 66
Introduction
Sequence improvement efforts by the Genomics Education Partnership (GEP) during Spring 2012 have revealed that many of the *Drosophila ananassae* Muller F element sequence improvement projects contain major misassemblies. With assistance from the finishers at the Washington University Genome Institute, we found that many of these projects are missing a large number of reads because of the misassemblies. Consequently, there is a need to develop a protocol and a set of tools that will enable GEP faculty and students to identify and retrieve additional sequencing reads from the NCBI Trace Archive ([http://www.ncbi.nlm.nih.gov/Traces/trace.cgi?view=list_arrivals](http://www.ncbi.nlm.nih.gov/Traces/trace.cgi?view=list_arrivals)), and incorporate these additional reads into their sequence improvement projects.

This user guide is designed to provide an overview of the GEP misassembly tools; it includes two case studies that illustrate common usage scenarios for these tools. While these case studies present the process of solving misassemblies as a step-by-step process, the actual process of solving misassemblies generally involves multiple trials. We should also note that this user guide is not a comprehensive reference on solving misassemblies. Please refer to the Identifying and Soring Tandem Duplications and an Inverted Repeat walkthrough for comprehensive explanations of the strategies you could use to tackle misassemblies.

Resources
The installer for the GEP misassembly tools is available at: [http://gander.wustl.edu/~wilson/install_misassembly_tools.sh](http://gander.wustl.edu/~wilson/install_misassembly_tools.sh)

The tutorial files are available for download at the GEP web site (under Curriculum → Washington University → Sequence Improvement → Resolving Misassemblies).

Prerequisites
This user guide assumes you are already familiar with the basic editing and navigation functionalities in *consed*. Before reading this user guide, we recommend that you first complete the *Drosophila Finishing Problem Set* and *A Complex Drosophila Fosmid* exercises ([http://gep.wustl.edu/curriculum/course_materials_WU/sequence_improvement/finishing_exercises](http://gep.wustl.edu/curriculum/course_materials_WU/sequence_improvement/finishing_exercises)).

For the purpose of this tutorial, we further assume that the tutorial package is located inside your home directory (i.e. `~/misassembly_tutorial`).
Installing the GEP misassembly tools

To facilitate the installation of the GEP misassembly tools on your computer, the GEP has developed a script that automates the installation process for systems running recent versions of Mac OS X or Linux. To install this package using the installation script, launch X11 and then copy and paste the following command onto the xterm:

```
curl -L http://gander.wustl.edu/~wilson/install_misassembly_tools.sh | bash
```

The command above will download and execute an installation script on your computer. Once the installation is complete, open a new terminal to access the GEP misassembly tools. To verify that the tools have been installed successfully, open a new xterm and execute the following command:

```
calc_total_fragSizes.pl -h
```

If the installation is successful, you should see a help message that shows the available options for the `calc_total_fragSizes.pl` script.

Please refer to Appendix 1 if you prefer to install the tools manually or if you would like to learn more about the steps involved in installing the GEP misassembly tools.

Summary of the tools in the GEP misassembly suite

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_trace_archive_reads.sh</code></td>
<td>Incorporate new reads from the NCBI Trace Archive into the current project.</td>
</tr>
<tr>
<td><code>build_unpaired_cloneid_query.pl</code></td>
<td>Generate a list of reads with a missing mate pair in the current assembly and construct the corresponding query to retrieve reads from the NCBI Trace Archive.</td>
</tr>
<tr>
<td><code>calc_total_fragSizes.pl</code></td>
<td>Calculate the total fragment sizes for all the restriction digests stored in <code>fragSizes.txt</code></td>
</tr>
<tr>
<td><code>extract_read_names_from_ace.pl</code></td>
<td>Extract read names from specific contigs and regions in the ace file.</td>
</tr>
<tr>
<td><code>find_reads_in_ace_file.pl</code></td>
<td>Find the location of each read and its mate pair in the ace file.</td>
</tr>
<tr>
<td><code>get_mate_pair_ti.sh</code></td>
<td>Generate the query to retrieve the mate pairs of the reads in a NCBI Trace Archive read package.</td>
</tr>
</tbody>
</table>

Many of the GEP Misassembly tools accept additional parameters besides those shown in the user guide. Please refer to Appendix 2 for detailed descriptions of each tool and the parameters available.
Case Study 1: Resolve a misassembly by finding reads with missing mate pairs

In this case study, we will examine a project (2402D22) with a misassembly that can be resolved by identifying the subset of reads that are missing their mate pairs, retrieving those missing reads from the NCBI Trace Archive, and incorporating these reads into the assembly. Assuming the tutorial files are available inside your home directory (i.e. ~/misassembly_tutorial), open an xterm and navigate to the edit_dir of project 2402D22 inside the misassembly_tutorial package. Launch consed and open the ace file 2402D22.fasta.screen.ace.1

cd ~/misassembly_tutorial/2402D22/edit_dir/
consed&

Using restriction digests to identify potential misassemblies

The initial Assembly View shows two contigs, 5 and 4c. To get an initial assessment of the project, we will complement Contig4 and identify the fosmid clone ends (Figure 1). Examining the Aligned Reads Window for Contig5, we can find both sets of fosmid end reads (i.e. reads with the prefix fosmidend) at the beginning and end of Contig5, respectively (Figure 2). This means that either Contig4 contains additional reads that are located outside the boundary of the fosmid or the reads in Contig4 have been improperly placed off the end of Contig5 because of a misassembly.

![Figure 1: Initial Assembly View for project 2402D22 shows two contigs.](image)

To determine which of these hypotheses is correct, we will examine the restriction digests using only Contig5 (Figure 3). The EcoRV digest shows a single 3.5kb band in the real digest that does not have a corresponding band in the in-silico digest (Figure 4). Examination of the other three digests shows that the in-silico digest is missing approximately 3.5kb compared to the real digest in all cases. To gain more confidence in the real digest and to verify that the size discrepancies cannot be attributed to miscalled bands, we will calculate the total fragment size for each restriction digest to see if the total sizes are consistent with each other and are within the expected size range of 43-52kb. To calculate the total fragment sizes for all digests present in a given project, run the calc_total_fragSizes.pl script in the xterm (from the edit_dir of your project).
Figure 2: Fosmid end reads are found at the beginning and end of Contig5.

Figure 3: Configure the digest window to digest "Entire Single Contig" with Contig5.
The output from the script shows that all four digests have a total size of ~46kb (Figure 5). Because the total fragment sizes are consistent across all of the real digests and are within the expected size range, we cannot attribute the ~3.5kb discrepancies to spurious bands being called in any of the real digests. Hence the discrepant bands are likely genuine and there is approximately 3.5kb of data missing from Contig5. Furthermore, the consistent forward reverse mate pairs between contigs 4 and 5 suggest that there is a misassembly and that Contig4 is likely to be part of this fosmid.

Figure 4: Extra 3.5kb band in the real EcoRV digest is missing from the in-silico digest.

```
~/.misassembly_tutorial/2402D22/edit_dir/ >calc_total_fragSizes.pl
#Enzyme Total_size
EcoRI  45996
EcoRV  46131
HindIII 45698
SacI  45941
```

Figure 5: calc_total_fragSizes.pl shows that all of the real restriction digests have a total size of approximately 46kb.
Identify regions that may have been misassembled

A closer examination of Contig5 in Assembly View reveals two pairs of inconsistent forward-reverse mate pairs at the end of Contig5 (Figure 6). In addition, we can use the high quality discrepancies navigator to help us identify repetitive regions that might have been incorrectly assembled into a single copy by phrap (the assembler). The consed high quality discrepancy navigator identifies a cluster of high quality discrepancies at the end of Contig5 (Figure 7). The consensus sequence in this region is marked with blue repeat tags which indicates that this region contains an rnd-1_family-780#Unknown repeat. The evidence is consistent with the hypothesis that reads derived from multiple repeat copies are incorrectly assembled into a single copy.

![Figure 6: Inconsistent mate pairs found at the end of Contig5.](image1)

![Figure 7: Cluster of high quality discrepancies at the end of Contig5.](image2)

Tell phrap not to overlap regions with multiple high quality discrepancies

To test this hypothesis, we will examine the regions with multiple high quality discrepancies in Contig5 (i.e. at 33,084, 33,588, 33,752, and 34,063). For each discrepancy, we will visually inspect all the traces in the region to verify these base calls. If these differences are judged to be genuine high quality discrepancies, we will add a tag to tell phrap not to overlap at the discrepant position because the restriction digest analysis indicates that there is a misassembly within this region (Figure 8). Had the real and in-silico digests been consistent, we would have tagged the discrepancies as putative polymorphisms instead.

In this case, visual inspection of the traces for the four regions confirms the base calls. Hence we will right-click on the consensus sequence and select the option “Tell phrap to not overlap reads discrepant at this position” at each of these discrepant positions.
After tagging all the positions with multiple high quality discrepancies, we will save the assembly, quit consed, run phredPhrap to reassemble the project, launch consed and open the new ace file. Assembly View now shows three large contigs (4, 5, and 6). Complement the contigs as needed so that they are in the orientation suggested by Assembly View and save the assembly.

We will then run crossmatch to help elucidate the repeat structure in this project. As expected, crossmatch shows the two copies of the repeat (at end of Contig6 and the entire Contig4) are very similar to each other with 99.7% sequence similarity (Figure 9).

Figure 8: Tell phrap not to overlap reads with high quality discrepancies at the end of Contig5.
Figure 9: The new assembly generated by phredPhrap after telling phrap not to overlap the discrepant positions at the end of the contig. The Sequence Matches window shows the highlighted repeats (between contigs 4 and 6) have a sequence similarity of 99.7%.

Retrieve missing mate pair reads from the NCBI Trace Archive

The evidence we have collected so far indicates there was a collapsed repeat in the initial assembly of the project. After separating the two repeat copies, Assembly View and the placement of the fosmid end reads (i.e. the large green triangles) suggest the order of the contigs should be 6-5-4. Furthermore, because crossmatch did not detect any sequence similarity between Contig5 and the other two contigs (Contig4 and Contig6), the gaps among these contigs are likely to be genuine.

Before we design primers to bridge the gaps, we should first ascertain whether there is any additional data available at the NCBI Trace Archive that will allow us to fill in the gap. Typically, the assembler will simply ignore reads that cannot be placed when it generates the whole genome assembly. While many of these reads are unplaced because they are low quality or they are sequencing artifacts, some of these reads are unplaced because of misassemblies. In particular, in cases where the misassembly is caused by a collapsed repeat, reads that belong to the misassembled region have likely been sequenced but are unplaced. The NCBI Trace Archive contains all the reads from the D. ananassae Whole Genome Sequencing (WGS) project irrespective of whether they were placed in the final assembly. Consequently, we will try to recover additional reads at the NCBI Trace Archive and use these reads to close the gaps in our project.

One strategy we can use to identify missing data is through the mate pairs. Except for some reads near the ends of the fosmid, we expect both reads from a mate pair to be included in the assembly. We can therefore identify missing data by looking for unpaired reads in our project. To do this we will need to construct a list of reads that are placed at the end of Contig6 as well as those placed within contigs 4 and 5. We will then check this list to identify any unpaired reads. Finally, we will
construct a query to retrieve both mate pair traces from the NCBI Trace Archive using the clone ID, and then incorporate these reads into our project.

**Step 1: Generate the list of read names to check for mate pairs**

Before we decide on the list of read names to check for missing mate pairs, we should first determine the average insert size for the 3-5kb plasmid library. In the consed Main Window, select “Info -> Show Library Info” and then examine the statistics for the `default` library. We find that the average insert size for this library is 3,347bp (with a standard deviation of 379bp) and that the maximum expected insert size is 4,294bp (Figure 10).

According to the crossmatch results, the direct repeat begins at 32,477 in Contig6 (Figure 9). Hence, we will include all the read names starting slightly upstream from this position (32,000) to the end of Contig6 (34,656). We chose the region slightly upstream of the repeat so that we will include reads that contain both unique and repetitive sequences. Note that based on the insert size statistics, it is possible that there are additional reads with missing mate pairs upstream of this region and we might need to repeat this procedure to retrieve additional reads.

![Figure 10: The 3-5kb plasmid library has an average insert size of 3,347bp and a maximum insert size of 4,294bp.](image)

There are three different ways to generate the list of read names. First, we can highlight each read name in the Aligned Reads window and paste the read name (using middle click) in a text editor inside your `xterm` (e.g. `emacs`, `pico`, `vim`). However, this technique is only useful for creating a list with small number of reads because you will need to do this one read at a time. Second, available in
newer versions of `consed` (>= 20.0), we can highlight all the reads we want and then select "Misc" -> "Save Highlighted Read Names to File" (Figure 11). We can then select “All Contigs” under the “Save reads just in this contig or in all contigs?” field and export the selected read names to a fof file (e.g. `check_mate_pairs.fof`) in the `edit_dir` of your project directory and (Figure 12). Third, we can use the script `extract_read_names_from_ace.pl` (described below). Note that irrespective of the technique you use, we will skip any read names with the prefix `danafos` or `fosmidend` because these are fosmid end reads and we do not expect to find their mate pairs within our project (i.e. the expected insert size for fosmid reads are ~40kb).

![Unlabeled Figure](image1.png)

**Figure 11:** Export all the highlighted read names to a file using the “Save Highlighted Read Names to File” option.

![Unlabeled Figure](image2.png)

**Figure 12:** Export highlighted reads in all contigs to the file `check_mate_pairs.fof`

Because there is a large number of reads that we would like to check, we will use the script `extract_read_names_from_ace.pl` to construct the list of read names. Before running the script, we will first save the current assembly as `2402D22.fasta.screen.check_matepairs.ace.1`. Then issue the following command on the xterm to search this ace file for all the reads placed at the end of Contig6 (32,000-34,656), in Contig4, or in Contig5 and write the read names to the file `check_mate_pairs.fof`:  

```bash
python extract_read_names_from_ace.pl 2402D22.fasta.screen.check_matepairs.ace.1 -q 10 -t 10 -s 100 -o check_mate_pairs.fof
```
extract_read_names_from_ace.pl \
-r Contig6:32000-34656,Contig4,Contig5 \
-i 2402D22.fasta.screen.check_matepairs.ace.1 \
-o check_mate_pairs.fof

For demo purposes, the file with all the read names (check_mate_pairs.fof) is also available inside the supplement directory in your project folder.

**Step 2: Generate a query for clones with missing mate pairs**

Once we have constructed the file with the list of read names to check, we can use the script build_unpaired_cloneid_query.pl to determine if the corresponding mate pair read exists in the chromat_dir. The script will collate the list of reads that are missing their mate pair and generate a query string that can then be used to retrieve the unpaired traces from the NCBI Trace Archive. To run the script, type the following command in the xterm you used to launch consed (i.e. inside edit_dir of your project directory):

build_unpaired_cloneid_query.pl -i check_mate_pairs.fof

This command will examine the read names in the file check_mate_pairs.fof (previously generated by the extract_read_names_from_ace.pl script) and construct a query string listing all of the clones that are missing their mate pair. Select the output beginning from "CLONE_ID" to the closing parenthesis and copy the selected text onto the clipboard (Figure 13 – in black).

![Figure 13: Query for unpaired mate pairs generated by the build_unpaired_cloneid_query.pl script.](image)

**Step 3: Retrieve the traces from the NCBI Trace Archive**

Open a web browser and navigate to the NCBI Trace Archive web site at http://www.ncbi.nlm.nih.gov/Traces/trace.cgi?view=list_arrivals. Paste the query string from the clipboard into the “Enter a query string” textbox and then click “Submit” (Figure 14).

![Figure 14: Query the NCBI Trace Archive using the query string generated by the build_unpaired_cloneid_query.pl to retrieve D. ananassae traces by clone ID.](image)
In the search results page, change the archive name in the "Save result of search as" box to unpaired_traces_2402D22. Verify that the "gz file" checkbox is selected, uncheck the "FASTA" checkbox, check the "SCF" checkbox and then click on the "Save query's results" button to save the package (Figure 15). In this tutorial, we assume that the package has been saved in your project directory (i.e. inside ~/misassembly_tutorial/2402D22).

For demo purposes, the read package (unpaired_traces_2402D22.tar.gz) is also available inside the supplement directory in your project folder.

**Figure 15: Select the SCF option to download the trace files from the NCBI Trace Archive.**

**Step 4: Rename and copy the new traces into chromat_dir**

Each trace is assigned a unique identifier by the NCBI Trace Archive. However, this trace name does not encode the mate pair information as expected by consed. Consequently, we need to rename all the traces so that they conform to the read name convention <clone_id>.<orientation>1. Furthermore, even though the package contains reads from both members of each mate pair, we only want to add the subset of reads that are not already in our project. We will use the script add_trace_archive_reads.sh to automate these steps. This script will rename the traces, identify and copy the new traces to the chromat_dir of the project, and create a fof file with only the names of the new traces. Execute the following command in the xterm (i.e. inside edit_dir of your project) to move the new traces to your project:

```bash
add_trace_archive_reads.sh \ 
../unpaired_traces_2402D22.tar.gz unpaired_traces_2402D22.fof
```

The command above will expand the unpaired_traces_2402D22.tar.gz archive located in the parent directory (i.e. the project directory), rename and copy the new traces to chromat_dir and create a new fof file called unpaired_traces_2402D22.fof which contains the names of the new reads (Figure 16).

**Figure 16: Bring the new traces in the unpaired_traces_2402D22.tar.gz package into the current project and create a new fof file called unpaired_traces_2402D22.fof inside edit_dir.**
Step 5: Incorporate the new reads into our consed database

The new traces are now in chromat_dir and the new fof file is in edit_dir. However, consed currently does not know about the new reads until we either use Add New Reads or phredPhrap to incorporate these reads into a new assembly. Please refer to the GEP wiki for detailed discussions of these two techniques and on the rationale for choosing one of these techniques over the other (http://gep.wustledu/wiki/index.php/Finishing_Questions). We only did a minimal amount of manual assembly (“sorting”) on this project, so we will use phredPhrap to incorporate the new reads. Once phredPhrap is complete, launch consed and examine the new assembly.

Examining the assembly after incorporating the new reads

From the Assembly View, we see a single contig (Contig4) in our new assembly. However, because we have retrieved all the reads at the end of the contig that have missing mate pairs, we have probably included extra reads that are placed beyond the end of the fosmid. To determine the actual fosmid end, we will search for read names that contain the prefix fosmidend (Figure 17).

Figure 17: Search for fosmid end reads in the new assembly.
The search result shows that the fosmid end read generated by Agencourt Biosciences (fosmidendac2402D22.g1) is placed at 33,868-34,658 while the fosmid end read generated by the Washington University Genome Institute (fosmidendwgc2402D22.g1) is placed at 37,453-38,360. Because Contig5 in our original assembly is approximately 34kb, and based on the digest data we have previously concluded that around 3.5kb was missing, we expect a properly placed end read to be at about 37-38kb. Consequently, the fosmid end read fosmidendac2402D22.g1 is likely to be misplaced. We will pull this read out of the assembly and then use the standard “Search for String” and “Compare Contig” technique to join it back to the contig at around 37-38kb (Figure 18).

Figure 18: Put the fosmid end read fosmidendac2402D22.g1 into its own contig.

When we compare the fosmidendac2402D22.g1 sequence against the 37kb region, we find a single high quality discrepancy: there is a G at position 197 in fosmidendac2402D22.g1 versus an A at position 37,652 in Contig4 (Figure 19).

Figure 19: Discrepant base between the Agencourt fosmid end read (Contig5) and the second repeat copy in Contig4.
A closer examination of the trace at this discrepant base position in Contig5 shows a higher A peak that overlaps with a lower G peak (Figure 20). Given that the two sets of fosmid end traces came from sequencing the same piece of DNA, the fosmid end sequences generated by Agencourt and the Washington University Genome Institute should be identical. Consequently, we will edit this discrepant base to a low quality A and then join the two contigs together. This will create Contig6 with the properly placed end reads.

![Figure 20: Discrepant base in the Agencourt read shows overlapping G and A peaks.](image)

After we have joined the two contigs together, we find that there are additional reads that extend beyond the end of the fosmid end reads (Figure 21). When we compare these read names against the list of new reads we have just added to the project (in unpaired_traces_2402D22.fof), we find that all of these traces correspond to some of reads we have just added to the project.

![Figure 21: The fosmid end read fosmidendwgc2402D22c.g1 shows that the insert-vector junction is at position 38,254 in Contig6.](image)
Because we can see vector sequences in the *fosmidendwgc2402D22c.g1* read (i.e., bases that have been masked by `consed` with X's), the fosmid insert likely ends at 38,254 in Contig6. To verify that we have identified the proper clone ends and that the project has been correctly assembled, we will check the restriction digest with "Just Part of Clone" for Contig6 from 1 to 38,254 (Figure 22).

Figure 22: Tell `consed` to digest Contig6 from 1 to 38,254 to verify we have correctly identified the fosmid clone ends and that the project is correctly assembled.
Figure 23: EcoRV digest shows that the real 3.5kb band is now accounted for in the *in-silico* digest.

All four *in-silico* digests are now consistent with the real DNA digests (Figure 23). The digest results are consistent with the interpretation that the misassembly has been resolved, and that the real fosmid insert DNA sequence can be found in Contig6 from 1 to 38,254. Consequently, we will add a clone end tag at position 38,254 (Figure 24). We will also add a comment tag to the consensus from 38,255 to the end of the Contig6 to demarcate the extra reads from the adjacent genomic region that we have incorporated into the project. We are now ready to go through the Finishing Checklist and resolve the remaining problems in this project.

Figure 24: Add clone end tag at 38,254 in Contig6 to denote the end of the fosmid.
Case Study 2: Retrieve traces with blastn against the NCBI Trace Archive

In this case study, we will examine another project (2391D16) with a major misassembly that could only be identified using restriction digests. This case study illustrates how you can identify additional reads that belong to a project by performing blastn searches against the *D. ananassae* - WGS database at the NCBI Trace Archive, using the contig consensus sequence as the query.

In this walkthrough, we assume that the tutorial files are available inside your home directory (i.e. ~/misassembly_tutorial). Open a new xterm and navigate to the edit_dir of the project 2391D16 and launch consed. Open the ace file 2391D16.fasta.screen.ace.1

```
   cd ~/misassembly_tutorial/2391D16/edit_dir/
   consed&
```

**Using restriction digests to identify potential misassemblies**

Initial assessment of the project using Assembly View shows that the project consists of two major contigs (Figure 25). Examining the Aligned Reads window for Contig4, we noticed that both sets of fosmid end reads are found within Contig4. In addition, the read 31172311015.b1 appears to protrude beyond the fosmid end (Figure 26).

![Assembly View](image1)

*Figure 25: Initial assembly view shows project 2391D16 consists of two major contigs.*

![Aligned Reads](image2)

*Figure 26: Read 31172311015.b1 extends beyond the end of the fosmid insert.*
Because we can see the masked vector sequences (X's) in the fosmidendwgc2391016c.bl1 read, the fosmid end should be at 38,228. To verify the locations of the fosmid ends, we will perform an in-silico digest of Contig4 from 1-38,228 (Figure 27). Because all the in-silico fragments at the end of the contig are consistent with the real digest fragments, we are confident that the right end of the fosmid insert should be at 38,228. However, all four restriction digests show major discrepancies at the beginning of the Contig4 (Figure 28), consistent with the hypothesis that a large amount of data is missing from this project. For example, the EcoRV digest shows that there is an extra 4kb in the real digest compared to the in-silico digest (Figure 29). Despite these discrepancies, we will add the cloneEnd tag at positions 1 and 38,228 to demarcate the positions of the fosmid end reads.

Figure 27: Digest Contig4 (1-38228) based on the position of the fosmid end reads.

Figure 28: Assembly View shows discrepant (red) digest bands at the beginning of the fosmid.
Figure 29: EcoRV digest shows that the real fragment (11,842) is ~4.5kb larger than the in-silico fragment (7,340) at the beginning of Contig4.

As in the first case study, we will use the `calc_total_fragSizes.pl` script to check the quality of the real digests. We find that all the restriction digests have a size of ~50kb, well within the expected size range (Figure 30). The EcoRI digest is approximately 1kb smaller than the other digests. However, comparison of the real versus the in-silico digests shows that there is a single 1kb band in the real EcoRI digest while the in-silico digest has two 1kb bands. Hence the discrepancy in the total size of the EcoRI digest can likely be attributed to this miscalled doublet.

Given the vector sequence has a total length of approximately 8kb, we expect the entire fosmid insert to have a total size of ~42kb. Because the total length of Contig4 is only 38kb, our analysis is consistent with the hypothesis that there is 4kb of data missing from our assembly. The missing data is likely caused by a collapsed repeat because there are no gaps present in the assembly.

Using repeat tags to identify regions that are likely to be misassembled

To test the hypothesis that the missing data is caused by a collapsed repeat, we will use the high quality discrepancy navigator to identify regions with clusters of high quality discrepancies (as we have done in the first case study). Unfortunately, all of the high quality discrepancies in this project can be attributed to either unclipped subclone vector sequences or miscalled bases (Figure 31). Hence we cannot use high quality discrepancies to help us identify the region with the collapsed repeat (Figure 31, bottom).

The 1,762bp discrepant band in the HindIII in-silico digest suggests at least part of the misassembly could be attributed to the first 1.7kb of the contig. Further evidence that there is a collapsed repeat within the first 1.7kb of the contig can be obtained by examining this region in the Aligned Reads window. In this region we find that there are three blue (repeat) tags in the consensus sequence.
(added by the script `tagRepeats.pl`) that are immediately adjacent to each other. These tags correspond to fragments derived from two different variants of an LTR/Pao transposon and an unknown repeat. For example, the repeat that extends from 417 to 1256 is an LTR/Pao transposon (Figure 32).

Original list of high quality discrepancies:

<table>
<thead>
<tr>
<th>Contig</th>
<th>Read Name</th>
<th>Consensus Name</th>
<th>Consensus Positions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contig4</td>
<td>24867111D15.bl</td>
<td>2122</td>
<td>high quality base disagrees with consensus</td>
<td></td>
</tr>
<tr>
<td>Contig4</td>
<td>24867111D15.bl</td>
<td>2125</td>
<td>high quality base disagrees with consensus</td>
<td></td>
</tr>
<tr>
<td>Contig4</td>
<td>damafos1817.bl</td>
<td>2663</td>
<td>high quality base disagrees with consensus</td>
<td></td>
</tr>
<tr>
<td>Contig4</td>
<td>250399111111.bl</td>
<td>4986</td>
<td>high quality base disagrees with consensus</td>
<td></td>
</tr>
<tr>
<td>Contig4</td>
<td>343396111108.bl</td>
<td>6924</td>
<td>high quality base disagrees with consensus</td>
<td></td>
</tr>
<tr>
<td>Contig4</td>
<td>34320411101081.bl</td>
<td>9470</td>
<td>high quality base disagrees with consensus</td>
<td></td>
</tr>
<tr>
<td>Contig4</td>
<td>2518891111111.bl</td>
<td>17571</td>
<td>high quality base disagrees with consensus</td>
<td></td>
</tr>
<tr>
<td>Contig4</td>
<td>25188911111110.bl</td>
<td>23430</td>
<td>high quality base disagrees with consensus</td>
<td></td>
</tr>
</tbody>
</table>

Unclipped vector:

Miscalled bases:

All high quality discrepancies have been resolved:

Figure 31: All the high quality discrepancies in the project can be attributed to unclipped vector and incorrect base calls.
Figure 32: The beginning of Contig6 contains multiple LTR fragments and unknown transposons.

Retrieve the missing mate pairs of all unpaired reads

Given our current hypothesis that there is a misassembly at the beginning of the contig that results in a substantial amount of missing data, we will begin our analysis by retrieving missing mate pair reads from the NCBI Trace Archive (using the protocol we have previously described in the first case study). However, instead of creating a list of specific reads to check for missing mate pairs, we will identify the subset of reads that are unpaired in our entire project and retrieve each corresponding mate pair from the NCBI Trace Archive. We can search through all the reads in chromat_dir and create the query string in one step by executing the following command on the xterm (inside edit_dir of the project).

```
ls ../chromat_dir | build_unpaired_cloneid_query.pl
```

(The "|" bar symbol is usually found in the upper right of the keyboard above the backslash "\". As with the first case study, we can use the output from this command (Figure 33) as a query string (by copy/paste) to retrieve the mate pairs from the NCBI Trace Archive (Figure 34). We will save the new trace package with the name unpaired_traces_2391D16.tar.gz.

For demo purposes, the unpaired_traces_2391D16.tar.gz archive is available inside the supplement directory in your project folder.
Figure 33: Identify all the unpaired reads in the current assembly and generate the corresponding CLONE_ID query for the NCBI Trace Archive.

Incorporate additional unpaired reads into the project

Assuming that the read package is in the project directory, we can incorporate the new reads into our project using the `add_trace_archive_reads.sh` script and then run `phredPhrap` to create a new assembly with the new reads:

```
add_trace_archive_reads.sh \
  ../unpaired_traces_2391D16.tar.gz unpaired_traces_2391D16.fof
```

After launching `consed` with the new assembly, open Assembly View. Complement the contigs in the orientation as suggested by Assembly View and save the assembly. We find that the new reads have extended the beginning of Contig4 (Figure 35). The extra reads have also extended Contig3 and created an additional contig (Contig2). This change in the assembly is expected because about half of the reads at the end of the contig should have mate pairs outside of the region defined by the fosmid ends.
Figure 35: Additional reads (newly paired) from the Trace Archive extend the beginning of Contig4.

However, the restriction digest analysis above indicated an expected fosmid insert size of 42kb, but the distance between the fosmid end reads is less than 40kb. This difference suggests that the current assembly is still missing about 2kb of data, a conclusion supported by a comparison of the real and in-silico EcoRV digests (Figure 36). It could be that the fosmid end reads (i.e. reads with the prefix fosmidend) are misplaced or that there is still a collapsed repeat in this new assembly. We will use the high quality discrepancy navigator again to look for evidence of a collapsed repeat.

The high quality discrepancies navigator identified two positions in the consensus sequence (932 and 942) with multiple high quality discrepancies. Because visual inspection of the traces confirmed that these are genuine high quality discrepancies and that there are discrepancies in multiple reads, we should be concerned about the possibility of a collapsed repeat in this region. Therefore, we will tell phrap not to overlap reads that are discrepant at these positions (Figure 37).

Furthermore, we will manually tear this Contig4 based on these high quality discrepancies. Using the discrepancy at base 942 (A versus C) as our guide, we will send all the reads with a C at position 942 to the new right contig and all the reads with an A to the new left contig (Figure 38).

We made this decision based on the location of the discrepancy within each read. We can quickly assess the location by noting the proximity of a discrepant read to the yellow line that divides the forward and reverse strand reads in the Aligned Reads window. In the Aligned Reads window, reads are stacked vertically according to their starting position: reads that are closer to the yellow line starts later than reads that are further way from the yellow line.

Consequently, reads with the A at consensus position 942 extend further to the left than the reads with the C at this consensus position. In addition, because we generally only want to rely on high quality reads when sorting a project, we will only move reads with a high quality A at the consensus position 942 to the new left contig (i.e. keep the read 00334884D10.g1 in the new right contig).
Figure 36: EcoRV digest (Contig4:1-40937) indicates that the project is still missing about 2kb of data.

Figure 37: Tell phrap not overlap the high quality discrepant bases at 932 and 942.
Retrieve additional reads through blastn searches against the Trace Archive

At this point, we have recovered all of the reads we could identify based on missing mate pairs (that is, the original assembly had one of two reads from the mate pair) but the restriction digests indicate that our assembly is still missing a substantial amount of data (~2kb). Because we have already added data by finding those reads without a mate and incorporating the mates into the project, the remaining data that are missing from our assembly are likely in reads where both members of the mate pairs are unplaced or they are placed in the wrong part of the whole genome assembly. In order to recover these reads, we will use the consensus sequence at the beginning of
Contig6 to query the NCBI Trace Archive for additional *D. ananassae* WGS read pairs that might belong in this region.

To perform the blastn search against the *D. ananassae* WGS reads at the NCBI Trace Archive, we will first export the Contig6 consensus sequence from 130 to 330 (i.e. the first 200 high quality bases at the beginning of Contig6) and save it as Contig6_begin.fasta inside the edit_dir of your project directory (Figure 39).

![Image](image.png)

**Figure 39:** Extract the beginning of the consensus sequence of Contig6 as the query for a blastn search against the *D. ananassae* WGS database at the NCBI Trace Archive.

Navigate to the NCBI Trace Archive BLAST page at [http://blast.ncbi.nlm.nih.gov/Blastcg?PAGE=Nucleotides&PROGRAM=blastn&BLAST_SPEC=Trace Archive&BLAST_PROGRAMS=blastn&PAGE_TYPE=BlastSearch](http://blast.ncbi.nlm.nih.gov/Blastcg?PAGE=Nucleotides&PROGRAM=blastn&BLAST_SPEC=Trace Archive&BLAST_PROGRAMS=blastn&PAGE_TYPE=BlastSearch), select the Contig6_begin.fasta sequence file as our query under the “Enter Query Sequence” section, and select “*Drosophila ananassae* – WGS” under the “Database” field. We will also change the BLAST program to “Somewhat similar sequences (blastn)” under the “Program Selection” field, set the Expect threshold to “1e-10”, and turn off the low complexity filter under the “Algorithm parameters” section (Figure 40).
Figure 40: Configure our blastn search against the *D. ananassae* – WGS database, using the beginning of Contig6 as our query, and turning off the low complexity filter.

The blastn results show a large number of hits to our query sequence (i.e. greater than 100, the maximum number of alignments BLAST will show by default). This result is not surprising because the query region has sequence similarity to multiple transposable elements. Because we know that the *D. ananassae* assembly has an overall coverage of ~8x, we expect only 8 reads on average to be placed at a given region. Because of the large number of (likely spurious) hits, we will apply a more stringent filter on the blastn results by requiring all the reads to match perfectly against our query (i.e. the beginning of Contig6).
Expand the “Formatting options” section and change the “Percent Identity Min” field to 100, then click “Reformat” (Figure 41). There are 10 reads that align perfectly with the query, a much more reasonable number considering that the genome assembly have ~8X coverage on average.

Figure 41: Change the Formatting options to show only reads that match perfectly to our consensus.

To download all the traces that match perfectly to the beginning of Contig6, scroll down to the “Descriptions” section of the blastn output and click on the “All” link (next to “Select”), then click on the link “Trace” on the main toolbar (Figure 42).

Figure 42: Select all the reads with perfect matches to our consensus query sequence, then click on the link “Get selected sequences” to download all the traces.

On the Trace Archive search results screen, change the archive name to traces_Contig6_begin, check the “gz file” box, check the box next to drop-down list labeled “SCF” and uncheck the “FASTA” checkbox (Figure 43). Click on the button “Save query’s results” and save the package in the project directory.

For demo purposes, the traces_Contig6_begin.tar.gz package is also available inside the supplement folder in the project directory.

We will also retrieve the mate pair traces for all the reads we found in our blastn search. We can use the placement of the mate pair reads in conjunction with any high quality discrepancies in the read to help us verify that the reads we have added actually belong to the project. For example, multiple high quality discrepancies or inconsistent mate pairs would indicate that the reads under examination have been misplaced. These reads may either belong to another part of the project or they may not belong to this project at all.
Figure 43: Configure the NCBI Trace Archive results page to download the trace files for all the selected reads from our blastn search results.

We will use the `get_mate_pair_ti.sh` script to help us retrieve the mate pairs of the 10 reads we have downloaded previously. Execute the following command on the xterm we used to launch `consed` (i.e. inside the `edit_dir` of the project):

```
get_mate_pair_ti.sh ..//traces_Contig6_begin.tar.gz
```

Copy the output from the `get_mate_pair_ti.sh` script into the NCBI Trace Archive search box and click “Submit” (Figure 44). Set the archive name to `traces_Contig6_begin_mate`, select the “.gz file” box, unselect the FASTA checkbox and select the SCF checkbox (as we have done previously). Click on the “Save query’s result” button and save the new package inside the project directory.

For demo purposes, the `traces_Contig6_begin_mate.tar.gz` package is also available inside the supplement folder in the project directory.

Figure 44: Obtain the mate pair traces for all the reads we have downloaded previously.
We can incorporate both sets of reads into our project by issuing the following commands inside the edit_dir of your project:

```
add_trace_archive_reads.sh \
  ../traces_Contig6_begin.tar.gz traces_Contig6_begin.fof

add_trace_archive_reads.sh \
  ../traces_Contig6_begin_mate.tar.gz traces_Contig6_begin_mate.fof
```

The `traces_Contig6_begin.fof` and `traces_Contig6_begin_mate.fof` contain the two sets of new reads that have not been incorporated into the assembly previously. We could incorporate these new reads into the assembly using either phredPhrap or Add New Reads depending on the state of the assembly. In this example, we will run phredPhrap to incorporate the new reads because we have only done a single tear and we have also tagged the region at the beginning of the contig to tell phrap not to overlap reads that are discrepant.

After running phredPhrap, open the new ace file and then open Assembly View. We will complement the contigs in the orientation suggested by Assembly View and note the two inconsistent forward reverse mate pairs between contigs 7 and 5. After running crossmatch, we also see that there is an identical copy of the repeat between contigs 4 and 7 (Figure 45). Hence one potential explanation for the misassembly is that the fosmid end should be placed in this other copy of the repeat, in Contig4 instead of Contig7.

![Assembly View](image)

**Figure 45:** Assembly View after incorporating the additional reads retrieved from blastn searches against the *D. ananassae* – WGS database at the NCBI Trace Archive.

**Determine the locations of all the new reads added to the assembly**

Unlike Add New Reads, phredPhrap does not create a custom navigation file with the locations of all the new reads we have just added to the assembly. While you could search for individual reads or read names that match a specific pattern using the “Find reads containing” field in the consed Main Window, there is no convenient way to determine the locations multiple reads in a list. The `find_reads_in_ace_file.pl` script is designed to address this limitation. For each read name in a fof file, this script will search for the read in the ace file specified by the user and report the placement of the read and its mate pair.
In this case, we would like to determine the placement of all of the new reads we have just added to our project in the current assembly. Because the script will automatically search for the mate pair of each read, we only need to provide one of the fof files (i.e. traces_Contig6_begin.fof) that have been previously generated by the add_trace_archive_reads.sh script. Before running the script, we will first save the assembly as 2391D16.fasta.screen.find_reads.ace.1. Then execute the following command on the xterm inside edit_dir of the project:

```
find_reads_in_ace_file.pl \ 
-i 2391D16.fasta.screen.find_reads.ace.1 \ 
-f traces_Contig6_begin.fof \ 
-o traces_Contig6_begin_locations.csv
```

This command will create a new file called traces_Contig6_begin_locations.csv in comma-separated values (CSV) format. This file can be imported directly into most spreadsheet programs (e.g. Microsoft Excel, Figure 46) for subsequent analysis. The CSV file consists of seven columns: the clone ID, the contig name, the left-most consensus position, and the orientation of the forward (b) and reverse (g) reads, respectively. Reads not found in the ace file are denoted by the value NA.

![Figure 46](image)

**Figure 46: Import the read placement information for all the reads in the traces_Contig6_begin.fof file and their mate pairs into Microsoft Excel.**

The results show that there are three clones where only one of its mate pairs has been incorporated into the assembly (i.e. 24988111A20, 25027811B17, and 25102111C07). All of the reads are placed in either Contig4 or Contig7 and the forward reverse mate pairs are pointing in the opposite direction (as expected). Furthermore, because there are neither inconsistent forward reverse mate pairs nor any high quality discrepancies in either contigs 4 or 7, we have no evidence that would suggest that any of these new reads are misplaced. Consequently, we will keep all of these new reads in the current assembly.

**Continue with blastn searches to retrieve more reads from the Trace Archive**

Our prior analysis of the restriction digests indicates that the fosmid insert should have a total size of ~42kb. Because Contig7 is only ~40 kb and we cannot find a join between Contig4 and Contig7 using “Search for String” or crossmatch (results not shown), we will continue to retrieve reads from the NCBI Trace Archive through blastn searches against the *D. ananassae* WGS database.
Open the Aligned Reads window and navigate to the beginning of Contig7. We see that the first high quality base is at position 22. (Note that the high quality bases upstream of this region in the read 25027811B17.b1 are vector sequences because the discrepant bases are at the beginning of the read and the discrepant bases start with the EcoRI cut site GAATTC). We will export the consensus from 22 to 222 and save the sequence file as Contig7_begin.fasta inside edit_dir (Figure 47).

![Image of Aligned Reads window]

**Figure 47: Export the beginning of Contig7 from 22-222 to the file Contig7_begin.fasta**

Using the protocol described above (see pages 28-33), we will use the Contig7_begin.fasta sequence to look for additional reads. The key steps in the protocol are summarized below:

1. Perform a blastn search at the NCBI Trace Archive (against the *D. ananassae* – WGS database), using Contig7_begin.fasta as the query

2. Filter the results to retain only the reads with 100% sequence identity.

3. Download the trace package (traces_Contig7_begin.tar.gz) into the project directory

4. Construct the query for missing mate pair traces with the get_mate_pair_ti.sh script:

```
get_mate_pair_ti.sh ../traces_Contig7_begin.tar.gz
```

5. Execute the query at the NCBI Trace Archive and download the mate pair traces into the project directory with the package name traces_Contig7_begin_mate.tar.gz

6. Use the add_trace_archive_reads.sh script to rename the new traces and move the new traces into the appropriate directories:

```
add_trace_archive_reads.sh \
../traces_Contig7_begin.tar.gz traces_Contig7_begin.fof

add_trace_archive_reads.sh \
../traces_Contig7_begin_mate.tar.gz traces_Contig7_begin_mate.fof
```
For demo purposes, both of these read packages are also available inside the supplement directory in the project directory.

Run phredPhrap again to incorporate the new traces from our blastn search into the new assembly. We can see that the additional reads have extended Contig7 but that there are a few more inconsistent forward reverse mate pairs at the beginning of Contig7 (Figure 48). Note that in the newer versions of consed (>= 22.0), the “only filtered” parameters for inconsistent mate pairs have changed, and the new default parameters are optimized for detecting inconsistent mate pairs in short read assemblies. Consequently, we need to change the parameter “pairs how close (both mates) to confirm” from 500 to 2000 (under “What to Show” -> “Fwd/Rev Pairs”, Figure 49). This parameter controls how far consed will look to find an additional inconsistent mate pair when there is already an inconsistent mate pair.

![Figure 48: Assembly View after incorporating additional reads from blastn search with the beginning of Contig7 against the NCBI Trace Archive.](image)

![Figure 49: For newer versions of consed (>= 22.0), change the “pairs how close (both mates) to confirm” parameter from 500 to 2000 to see all the inconsistent forward reverse mate pairs.](image)

Clicking on the inconsistent forward reverse mate pairs in Assembly View (Figure 50), we can determine the clone IDs of the inconsistent reads and see if they correspond to the reads we have just added to the assembly.
We can verify the origin of these reads by issuing the following commands on the xterm inside the edit_dir of your project directory (Figure 51):

```bash
grep '25038911N08' traces_*.fof
grep '22352011F11' traces_*.fof
grep '35227511M19' traces_*.fof
```

Figure 51: Using grep to determine if the inconsistent mate pairs come from the set of new reads we have just added into the project.

The commands above use the UNIX tool `grep` to search for the clone ID of the three pairs of inconsistent reads in the fof files we have created previously (which all have the prefix `traces_` and the suffix `.fof`). The search results show that all the inconsistent read names can be found in the files `traces_Chip47_begin.fof` and `traces_Chip47_begin_mate.fof`, which means that they correspond to a subset of the reads that we have just added to the project. The inconsistent forward reverse mate pairs indicate that these reads are placed at an incorrect location in the assembly. We will keep these reads in their current positions for now because they do not affect the consensus sequence, but we will need to resolve these inconsistent mate pairs later in the sorting process.
Using the restriction digests to estimate the amount of missing data

Next, we will use the restriction digests to ascertain the amount of data that is still missing from the project (Figure 52). Note that we will set the end of Contig7 to 41,163 because we have previously established that the read 31172311015.b1 extends beyond the end of the fosmid (see pp. 20-21).

The EcoRV digests shows that we are making progress in reducing the amount of missing data but the project is still missing approximately 1.6kb of data. Given that the average read length is ~700bp and we are only looking for reads that have 100% identity over a 200bp query, we expect the amount of missing data will shrink by no more than ~500 bp with each iteration of the blastn search against the NCBI Trace Archive. Consequently, given that the project is missing ~2kb of data, we need to perform multiple rounds of blastn searches to recover all the missing data, essentially walking along the fosmid. We will use the high quality sequence at the beginning of the new Contig7 to continue our blastn searches against the *D. ananassae* – WGS database at the NCBI Trace Archive.
Open the Aligned Reads window for Contig7 and extract the consensus from 40-240 (i.e. the first high quality base after the vector sequence in the read 24878411L16.g1). We will call this new sequence file Contig7_begin_rd2.fasta (Figure 53).

**Handling blastn results with multiple partial matches**

Repeat the procedure as described above to perform the blastn search against the *D. ananassae* - WGS database at the NCBI Trace Archive with the Contig7_begin_rd2.fasta sequence. Set the “Percent Identity Min” threshold to “100” to retain only the perfect blastn matches.

Unlike the previous sets of blastn results, we find that there are many matches to just the beginning (from bases 1 to 75) of the sequence (Figure 54). While these blastn hits have 100% sequence identity, they do not match across the entire 200 bases so these reads are unlikely to belong to the project. To filter out these spurious hits, we will add an E-value cutoff of 1e-50 into the “Expect Max” field under “Formatting options” (Figure 55). The last read we will include has the accession number ti:450540097 (i.e. the last red BLAST hit in the original BLAST Graphic Summary).
Figure 54: blastn identified many matches between *D. ananassae* traces and the beginning of Contig7.

Figure 55: Apply an additional filter to retrieve only reads with E-values less than 1e-50.
As before, we will use the following protocol to incorporate these reads and their mate pairs into our project:

1. Select “All” the reads and download the package (traces_Contig7_begin_rd2.tar.gz) into the project directory

2. Construct the query for missing mate pair traces with the get_mate_pair_ti.sh script:
   
   ```
   get_mate_pair_ti.sh ../traces_Contig7_begin_rd2.tar.gz
   ```

3. Execute the query at the NCBI Trace Archive and download the mate pair traces into the project directory with the package name traces_Contig7_begin_rd2_mate.tar.gz

4. Use the add_trace_archive_reads.sh script to rename the new traces and move the new traces into the appropriate directories:
   
   ```
   add_trace_archive_reads.sh \
   ../traces_Contig7_begin_rd2.tar.gz traces_Contig7_begin_rd2.fof
   ```
   ```
   add_trace_archive_reads.sh \
   ../traces_Contig7_begin_rd2_mate.tar.gz \ 
   traces_Contig7_begin_rd2_mate.fof
   ```

For demo purposes, these read packages are available in the supplement directory inside the project directory.

We will incorporate the new reads into the assembly using phredPhrap and launch consed. The consed Main Window shows that Contig7 is approximately 42kb (Figure 56). Because our original analysis of the restriction digest shows that the total size of the fosmid insert is approximately 42kb, we might have recovered all of the reads that we need to resolve this misassembly.

![Consed Main Window](image)

**Figure 56:** Main contig window shows that the new Contig7 is approximately 42kb.
Use Miniassembly to help resolve the misassembly

Open Assembly View and run crossmatch. Assembly View shows that there are additional inconsistent mate pairs that were caused by the new set of reads we have just added (Figure 57), consistent with the hypothesis that the beginning of Contig7 has been misassembled. (Remember to change the “pairs how close (both mates) to confirm” field from 500 to 2000 if you are using consed version 22.0 or above in order to see the entire set of inconsistent mate pairs.) Consequently, we should do some manual sorting of this region first before we continue to use blastn searches to retrieve additional reads from the NCBI Trace Archive.

Figure 57: New assembly shows additional inconsistent mate pairs at the beginning of Contig7.

Because we know that the fosmid end reads should be placed at the ends of the fosmid insert, we will first delete the clone end tag at consensus position 3,562 in Contig7 and then pull both of these fosmid reads out of this contig (Figure 58). Save the assembly and then run Miniassembly to assemble the two contigs (Contig8 and Contig9) with the fosmid end reads together (Figure 59). (Note: We decided to use Miniassembly instead of manually joining the two fosmid end reads together to ensure that the vector sequences are clipped properly.)

Figure 58: Put the fosmid end reads into their own contigs.
Figure 59: Run Miniassembly on the fosmid end reads to create a new contig.

Figure 60: Add a cloneEnd tag to beginning of Contig8 with the fosmid end reads.

Open the Aligned Reads window for Contig8 and add a cloneEnd tag to the beginning of the new Contig8 that was just created by Miniassembly (Figure 60). Navigate to the end of the high quality area in this contig (around base 500). Select the consensus from 480-500 and perform a “Search for String” using this sequence against the entire assembly. We find that this sequence matches the beginning of Contig4 (385-405) and matches in Contig7 (4041-4061) (Figure 61).
Figure 61: Search for String results using the consensus sequence in Contig8 from 480-500.

Because we just pulled the fosmid end reads out of Contig7, we will try to join Contig8 with Contig4 instead. Open the Aligned Reads window for both Contig4 and Contig8 using the “Searching Contigs” dialog box. Then click on the “Compare Cont” button to align the two regions against each other (Figure 62). We see that the unaligned region at the beginning of Contig4 could likely be explained by unclipped vector (notice the characteristic GAATTC EcoRI recognition sequence at the start of the unaligned region in Contig4 from base 29 to 34). Based on this alignment, we will join the two contigs together. Save the assembly and then open Assembly View.

Figure 62: Compare Contig4 with Contig8 shows that the discrepancies between the two contigs are caused by either unclipped vector or low quality sequences.

From the new Assembly View, we see that there are a few inconsistent forward reverse mate pairs at the beginning of Contig7. We will move the mouse over the red triangles so that they turn yellow and then click on these highlighted triangles with the left mouse button. In the “Clicked Forward/Reverse Pairs” dialog box, click on the “Pull Out Reads” button. Then select all the reads in the “Put Reads Into Their Own Contigs” window and click on the “Remove Highlighted Reads” button to put all of these reads into their own contigs (Figure 63). Save the assembly.
Figure 63: Pull the inconsistent mate pairs out of the beginning of Contig7 and put each read into its own contig.

Next, we will use Miniassembly to see if the reads we have just pulled out of the beginning of Contig7 can be assembled together with consistent forward/reverse pairs. After Miniassembly is complete, we find that it created two contigs (10 and 11), each with 6 reads (Figure 64).
Figure 64: Running Miniassembly on the reads that we have just pulled out of the beginning of Contig7 results in two contigs.

After saving the assembly, open Assembly View again. By default, Assembly View will not show the small contigs that were just created by Miniassembly. To see these smaller contigs, select “What to Show” -> “In/exclude Contigs” and change the “exclude contig if this many reads or less” field to 5 (Figure 65). Then click on the “Apply and Restart Assembly View” button. Complement the contigs as needed to show the contigs in the orientation suggested by Assembly View.
Figure 65: Change the In/exclude Contig settings to show the smaller contigs in Assembly View.

Delete the clone end tag at the end of Contig10 (navigate by tags, then select “cloneEnd”). Pull out the inconsistent forward reverse mate pairs from Contig7 that connect contigs 5 and 7 (Figure 66). Save the assembly and assemble all of these reads into their own contigs with Miniassembly (Contig12). After opening Assembly View, complementing the contigs as suggested by Assembly View and running crossmatch, we find that the new contig (Contig12) has sequence similarity to both Contig7 and Contig9 (Figure 67).

Figure 66: Pull the inconsistent forward reverse mate pairs from Contig7.
Figure 67: Assembling the inconsistent mate pair reads in Contig7 with Miniassembly created Contig12, which has sequence similarity to both Contig7 and Contig9.

Because the mate pairs in Contig12 were inconsistent when they were placed in Contig7, we will try to join Contig12 with Contig9 by selecting the match between Contig9 and Contig12 in the “Sequence Matches” dialog. Click on the “Show Alignment” button so that we can inspect the alignment between these two contigs. The alignment looks good overall so we will join the two contigs and then save the assembly (Figure 68).

Figure 68: Compare Contigs window shows a good alignment between Contig9 and Contig12.
Following the join we note in Assembly View (after running crossmatch) that there is a 100% match between the beginning of Contig7 and Contig11 (Figure 69). Select this match and then click on the "Show Alignment" button, examine the alignment and then join the two contigs together. Save the assembly.

![Assembly View](image)

**Figure 69:** crossmatch shows a perfect match between the beginning of Contig7 and Contig11.

The resulting Assembly View is shown in Figure 70. Based on the cloneEnd tags, we would like to join Contig13 with Contig14. However, crossmatch and manual “Search for String” failed to identify regions where we can join these two contigs together, which suggests that some data might still be missing between the two contigs.

![Assembly View](image)

**Figure 70:** Assembly View after joining Contig7 with Contig11; this reveals a unique region at the end of Contig13.
Use restriction digests again to estimate the amount of data still missing

To get an idea of how much data is still missing, we will run the restriction digest with Contig13 and Contig14. As we have previously mentioned, there is a read at the end of Contig14 that protrudes beyond the end of the fosmid so we will digest “Just Part of Clone” and specify the end position of Contig14 at 41,789 (Figure 71). The EcoRV, EcoRI, and ScaI digests are now consistent between the in-silico and the real digests. However the HindIII digest indicates that there is a gap of about 200 bases (Figure 72). Given that we cannot find a good join, we will again look for more reads at the NCBI Trace Archive that might enable us to connect these two contigs together. One could consider calling for a sequencing reaction to cover the gap, but the high repeat density might make it difficult to design unique primers in this region.

Figure 71: Estimate gap size between Contig13 and Contig14 using restriction digests.
**Figure 72: HindIII digest suggests that the gap between Contig13 and Contig14 is ~200bp.**

**Incorporate additional traces into the project with Add New Reads**

Using the protocol described above, we will extract the high quality consensus sequence near the end of Contig13 (1240-1440) and use it as the query (Contig13_end.fasta) to search against the *D. ananassae* – WGS database at the NCBI Trace Archive. Retain only the reads with alignments that have 100% sequence identity. Download the traces and name it traces_Contig13_end.tar.gz. Use `get_mate_pair_ti.sh` to construct the query to retrieve the mate pair reads and then use this query to perform a search of the NCBI Trace Archive. Retrieve the corresponding mate pair traces and name this file traces_Contig13_end_mate.tar.gz.

For demo purposes, both of these read packages are also available inside the supplement directory in the project directory.

**Incorporate the new sets of reads into the project using `add_trace_archive_reads.sh`**

```
add_trace_archive_reads.sh \
  ../traces_Contig13_end.tar.gz traces_Contig13_end.fof
```

```
add_trace_archive_reads.sh \
  ../traces_Contig13_end_mate.tar.gz traces_Contig13_end_mate.fof
```
Because we have done quite a few manual tears and joins, we do not want to run phredPhrap to generate a new assembly (as we would likely have to redo all of those tears and joins). Instead, we will use Add New Reads with the fof files generated by the add_trace_archive_reads.sh script to incorporate the new reads into our assembly.

Returning to consed, save the assembly and then click on the “Add New Reads” button and select the traces Contig13 end.fof file to add the first set of reads (Figure 73). Save the assembly and then repeat the same process to add the reads in the traces Contig13 end mate.fof file.

Examine the Aligned Reads window for Contig13, we notice that even though there are many reads that extend beyond the end of the consensus sequence, all of these bases are shown as discrepant (Figure 74). This is a consequence of the “Add New Reads” protocol, which does not always properly update the consensus as expected. Instead of manually extending the consensus sequence, we will run Miniassembly on Contig13, as this protocol will correctly extend the consensus sequence (Figure 75). After running Miniassembly, the new Contig17 should consist of 1,945 bases (Figure 76). Save the assembly.
Figure 74: Add New Reads failed to extend the consensus sequence at the end of Contig13.

Figure 75: Use Miniassembly on Contig13 to update the consensus sequence.

Figure 76: Miniassembly extends the end of Contig13 and renames it to Contig17
Open Assembly View and run crossmatch. We see that the end of Contig17 matches with the beginning of Contig14 (Figure 77). Click on the “Show Alignment” button, examine the alignment, and then join the contigs together and save the assembly.

Figure 77: Sequence matches between the end of Contig17 and the beginning of Contig14.

**Confirm the final assembly with restriction digests**

We will now use the restriction digest to confirm this join. Before we run the digest, we will pull the read 31172311015.b1 from the assembly into its own contig, because our previous analyses have shown that this read protrudes beyond the end of the fosmid. Save the assembly and then run the digest against the “Entire Single Contig” with Contig18 (Figure 78).
Figure 78: After pulling out the read 31172311015.b1, run digest against Contig18.

All the major fragments in the *in-silico* digests are now consistent with the real digest. The only exception is the *Eco*RI digest, where there should be two 1kb bands in the real digest according to our previous analysis of the total fragment sizes. In particular, the previously discrepant band (at around 4.8kb) in the real *Hind*III digest is now consistent with the *in-silico* digest (Figure 79).
Further investigation into the misassembly

At this point, the digest data suggests we have resolved the misassembly. However, when you open Assembly View, you will notice two pairs of inconsistent forward reverse mate pairs between Contig18 and Contig10; these mate pair reads are too far apart (Figure 80). (Note that, as previously described, if you are using consed version >= 22.0, you will need to change the “pairs how close (both mates) to confirm” setting to 2500 in order to see both inconsistent reads).

Figure 79: In-silico digest using HindIII matches the real digest after joining the contigs.

Figure 80: Two inconsistent mate pairs between Contig18 and Contig10
After running crossmatch, we see that the end of Contig10 has high degree of sequence similarity with Contig18, which suggests these regions could be joined together (Figure 81). Furthermore, crossmatch also suggests that the beginning of Contig10 overlaps with the end of Contig5.

Because Contig10 extends beyond the end of the fosmid as identified by the fosmid end clone sequences, we would normally not join Contig10 with the end of Contig18, as the sequences in Contig10 are not part of the fosmid. However, for instructional purposes we will try to join these contigs together to get a clearer view of this region and to gain a better understanding of the cause of the misassembly.

After joining these contigs together and running crossmatch, we find that there is a 3.4kb direct repeat that extends beyond the beginning of the fosmid; the two repeat copies have a 99.8% sequence similarity (Figure 82). In the original published assembly, these two repeats were incorrectly assembled, collapsed together as a single copy. The fact that the fosmid actually ended in the middle of this direct repeat introduces another confounding factor as these reads were incorrectly placed in the right hand copy of the repeat.

Figure 81: crossmatch shows overlap between the end of Contig10 and beginning of Contig18
Figure 82: After joining all the contigs together, we find that there is a 3.4kb direct repeat that extends beyond the beginning of the fosmid.

Conclusions

The two cases described in this user guide demonstrate how the GEP misassembly tools can be used to help solve misassemblies and retrieve missing data in the GEP *D. ananassae* sequence improvement projects. While this user guide presents the process of solving misassemblies as a step-by-step process, the actual process of understanding and resolving misassemblies is generally not a linear process and may require much trial and error. Please do not hesitate to contact the GEP staff if you have any questions during the course of resolving a misassembly in your GEP sequence improvement project.
Appendix 1. Installing the GEP misassembly tools

The install_misassembly_tools.sh script

The installation script install_misassembly_tools.sh attempts to detect your platform (i.e. Mac OS X or Linux) and shell (i.e. [t]csh, bash, zsh), download the pre-compiled package, and update the PATH environment variable to include the new directory that contains the misassembly tools.

By default, the script will create a new folder in ~/bin called misassembly_tools where all the tools will be installed. You can provide an optional parameter to the install_misassembly_tools.sh script to install the tools at an alternate location.

For example, use the following command to install the misassembly tools in ~/some_directory.

curl -L http://gander.wustl.edu/~wilson/install_misassembly_tools.sh | \ 
  bash -s ~/some_directory

In order to enable the user to execute the misassembly tools without providing the full path, the installer will also try to append the path to the misassembly_tools directory to the environment PATH variable (if the path does not already exist). Depending on your shell, the installer will update different startup files in order to modify the PATH variable:

<table>
<thead>
<tr>
<th>Shell</th>
<th>Modified startup file</th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>.profile</td>
</tr>
<tr>
<td>csh / tcsh</td>
<td>.cshrc</td>
</tr>
<tr>
<td>zsh</td>
<td>.zprofile</td>
</tr>
</tbody>
</table>

The pre-built package should be compatible with Mac OS 10.6 or above and with most Linux distributions (both 32 and 64 bit systems). Please contact Wilson (wleung@wustl.edu) if the package is incompatible with your platform or if you encounter any installation issues with the GEP misassembly tools.
Manual install

1. Download the appropriate package for your platform:

Mac OS X: http://gander.wustl.edu/~wilson/misassembly_tools/misassembly_tools_mac.tgz

Linux: http://gander.wustl.edu/~wilson/misassembly_tools/misassembly_tools_linux.tgz

2. Create the directory where the GEP misassembly tools should be installed and expand the package. In this example, we assume that the package is available in the current directory and we will install the Mac OS X package in ~/bin

   mkdir -p ~/bin

   tar -xzf misassembly_tools_mac.tgz -C ~/bin

3. Append the path to the misassembly_tools folder to your shell startup file. Depending on the shell you are using, you will need to update a different shell startup file. You can determine the shell you are using with the following command:

   echo $SHELL

   In this example, we assume that the path to the misassembly tools is ~/bin/misassembly_tools

   tcsh or csh:
   echo 'set path = ( ${path} ${HOME}/bin/misassembly_tools )' >> ~/.cshrc

   bash:
   echo 'export PATH=${PATH}:${HOME}/bin/misassembly_tools' >> ~/.profile

   zsh:
   echo 'export PATH=${PATH}:${HOME}/bin/misassembly_tools' >> ~/.zprofile

Compiling the D source files
The GEP misassembly tools include two pre-compiled binary files (build_mate_pair_query and rename_traces). These programs are used to process the TRACEINFO.xml file that contains the metadata associated with each read in the package you download from the NCBI Trace Archive.

These programs are developed using the D programming language (http://dlang.org/) and the source code for both programs is included as part of the installation package. Hence, you can compile these programs yourself using the D compiler. The installers for the D language compilers and libraries are available at http://dlang.org/download.html.
Appendix 2: Detailed descriptions of the GEP misassembly tools

add_trace_archive_reads.sh

Synopsis:
add_trace_archive_reads.sh <traces.tar.gz> <reads_to_add.fof>
  traces.tar.gz  = Read package downloaded from the NCBI Trace Archive
  reads_to_add.fof = File with the new read names

Incorporate reads from the NCBI Trace Archive into the current project.

Description:
This script should be executed inside the edit_dir of the project directory. This script will first expand the read package into the /tmp directory and then rename the traces according to the clone_id field in TRACEINFO.xml using the program rename_traces. The renamed traces are then compared against the files in the chromat_dir of the project directory. Any renamed reads not in the chromat_dir directory are copied from the temporary directory to the chromat_dir and the read name is added to the fof file specified by the user.

rename_traces

Synopsis:
rename_traces -i <infile> -o <outdir>
  infile = Path to TRACEINFO.xml file
  outdir = Path to the output directory for the renamed traces

Rename traces using the metadata in TRACEINFO.xml and place the output files in outdir.

Description:
This is a helper script for add_trace_archive_reads.sh. It parses the TRACEINFO.xml file that is included in the read package downloaded from the NCBI Trace Archive to determine the name mapping between the trace name and the St. Louis read naming convention used by consed. It then renames all the traces using this name mapping.

The new read names are based on the clone ID and the orientation of the read (b = forward, g = reverse). For example, the read name 24750511M06.b1 means that the read has the clone ID 24750511M06 and it is in the forward orientation.
**build_unpaired_cloneid_query.pl**

**Synopsis:**
build_unpaired_cloneid_query.pl
[-i <read_names.fof>]
[-c <chromat_dir>]
[-o <outfile>]

-i,--infile <read_names.fof>
   File with list of read names to check
   [Default = stdin]

-o,--outfile <output_file>
   Output file to write the Trace Archive query
   [Default = stdout]

-c,--chromat <path to chromat_dir>
   Path to chromat_dir where the traces are stored
   [Default = ../chromat_dir]

Construct query to retrieve unpaired reads (i.e. reads with missing mate pair) in the current project from the NCBI Trace Archive.

**Description:**
This script searches the list of read names specified in the read_names.fof file against the reads in the chromat_dir directory to determine if both mate pairs are present in the chromat_dir directory. The script then generates a query using the CLONE_ID field to enable the user to retrieve all the unpaired reads from the NCBI Trace Archive.
calc_total_fragSizes.pl

Synopsis:
calc_total_fragSizes.pl [-i <fragSizes.txt>] [-o <output file>]
   -i,--infile <fragSizes.txt>
       The fragment sizes file used by Consed

   -o,--outfile <output file>
       Output file with total fragment sizes
       [Default = stdout]

Calculate the total fragment sizes for all the restriction digests stored in fragSizes.txt

Description:
Each GEP sequence improvement project includes a file called fragSizes.txt, which contains the sizes of the real restriction fragments. In some cases, the automated band callers may under-call or over-call bands in the real digests. This tool reports the total fragment sizes for each digest specified in fragSizes.txt.
**extract_read_names_from_ace.pl**

**Synopsis:**
```
extract_read_names_from_ace.pl \
  -r <contig:start-end> [-i <ace_file>] [-o <outfile>]

-r,--region <contig:start-end>
    Specify the region to extract read names. The start and end fields are
    optional. For example, `-r Contig1` will extract all the reads in Contig1.
    Multiple regions from different contigs can be specified using a comma-
    separated list or with multiple `-r` flags.

-i,--infile <ace file>
    Ace file to analyze
    [Default = stdin]

-o,--outfile <output file>
    Output file of filenames with all the reads from the specified region
    [Default = stdout]
```

Extract read names from specific contigs and regions in the ace file.

**Description:**
This script examines an ace file and identifies the subset of reads that
belongs to an entire contig or regions within a contig specified by the user.
For example, you can use the following command to create the list of read
names from 1 to 1000 in Contig6 from the ace file project.screen.ace.1:

```
extract_read_names_from_ace.pl -r Contig6:1-1000 -i project.screen.ace.1
```

Multiple regions can be specified by a comma-delimited list or by using
multiple `-r` flags. For example, you can use the following command to extract
all the reads in Contig5 and reads in Contig6 from 1-1000 in the ace file
project.screen.ace.1:

```
extract_read_names_from_ace.pl \
  -r Contig6:1-1000,Contig5 -i project.screen.ace.1
```

Both the start and end positions are optional when defining a region:

- **Contig6:1000-** will extract read names from 1000 to the end of Contig6
- **Contig6:-1000** will extract read names from the beginning of Contig6 to 1000
**find_reads_in_ace_file.pl**

**Synopsis:**
find_reads_in_ace_file.pl -i <ace_file> [-f <read_names.fof>] [-o <outfile>]

- **-i, --infile <ace file>**
  Ace file to analyze

- **-f, --foffile <fof file>**
  File with the list of read length to check
  [Default = stdin]

- **-o, --outfile <output file>**
  Output file with the list of read locations (in CSV format)
  [Default = stdout]

Find the location of each read and its mate pair in the ace file.

**Description:**
This script lists the locations of reads in the read_names.fof file and its corresponding mate pair based on the ace file provided by the user. Read names that cannot be found in the ace file are denoted by **NA**.

The output file is in CSV (comma-separated values) format, which can be imported into Microsoft Excel (or other spreadsheet programs). The output file consists of 7 columns:

- **clone_id**: Clone ID / template
- **b_contig**: Contig where the b1 (forward) read is found
- **b_start**: Left-most position of the b1 read in the consensus
- **b_orientation**: Orientation of the b1 read relative to the consensus
- **g_contig**: Contig where the g1 (reverse) read is found
- **g_start**: Left-most position of the g1 read in the consensus
- **g_orientation**: Orientation of the g1 read relative to the consensus
**get_mate_pair_ti.sh**

**Synopsis:**
get_mate_pair_ti.sh <traces.tar.gz> [query.txt]
  traces.tar.gz = Package from the NCBI Trace Archive
  query.txt = Output File with the Trace Archive query
  [Default = stdout]
Construct query to retrieve mate pairs of the reads in a read package from the NCBI Trace Archive.

**Description:**
This script temporarily expands the read package from the NCBI Trace Archive. It then examines the TRACEINFO.xml file in the read package to determine the TI numbers of the mate pairs for the reads in the package. Using these TI numbers the script generates the corresponding query to facilitate the retrieval of the mate pair reads from the NCBI Trace Archive.

**build_mate_pair_query**

**Synopsis:**
build_mate_pair_query -i <infile> [-o <outfile>]
  infile = Path to TRACEINFO.xml file
  outfile = Path to output file where the query should be written
  [Default = stdout]

**Description:**
A helper script for get_mate_pair_ti.sh, this script parses the TRACEINFO.xml file to extract the list of TI numbers that corresponds to the mate pairs of the reads in the read package. It then generates the corresponding NCBI Trace Archive query and writes the results to the output file.