## Information Recorded About PDB Entries

The following database tables, along with their columns, are used to record information about the PDB entries:

* AllPDBEntries
  + entry – The identifier of the PDB entry.
* ChainType
  + chain – The identifier of the PDB chain.
  + chainType – The type of the PDB chain.
* EntryRepresentative
  + nonReprChain – The chain identifier of the non-representative chain.
  + reprChain – The chain identifier of the representative chain.
* ProteinInformation
  + chain – The identifier of the PDB chain.
  + entry – The identifier of the PDB entry.
  + experimentType – The abbreviation for the experimental method used to determine the protein structure.
  + resolution – The resolution of the protein structure.
  + rValueObs – The observed R value for the protein structure.
  + rValueFree – The free R value for the protein structure.
  + alphaCarbonOnly – Whether the recorded protein structure contains only backbone alpha carbon atoms.
  + description – A description of the protein chain.
  + dbName – The name of the external database where the chain can be found.
  + dbCode – The code in the external database that corresponds to the protein chain.
  + organism – The organism that the chain comes from.
  + sequence – The sequence of the chain.
* Representative
  + nonReprChain – The chain identifier of the non-representative chain
  + reprChain – The chain identifier of the representative chain.
* Similarity
  + id – A unique ID for the similarity (used purely as a primary key).
  + chainA – The identifier of one of the chains involved in the similarity.
  + entryA – The identifier of the entry of one of the chains involved in the similarity.
  + chainB – The identifier of one of the chains involved in the similarity.
  + entryB – The identifier of the entry of one of the chains involved in the similarity.
  + similarity – The percentage sequence identity between the chains.
  + matchLength – The length of the match between the two chains.

## Extracting PDB Data

The PDB data is extracted from a subset of the mmCIF files of the PDB structures. The subset is limited to those entries that have had their data updated within the last week. The elements of the records that are extracted are:

* Entry identifiers.
* Chain identifiers.
* The experimental method used to determine the structure.
* Structural resolution.
* R value and free R value.
* Whether the recorded structure contains only backbone alpha carbon atoms.
* External database identifiers for the chains.
* The organism that the chains come from.
* Sequences for the chains.

The mmCIF fields that the data is extracted from are as follows:

* \_entry
  + \_entry.id
    - Used to determine the name of the PDB entry (e.g. 3A0B).
* \_entity
  + \_entity.id
    - The numeric identifier used to identify the specific entity within the entry record.
  + \_entity.pdbx\_description
    - A description of the entity.
* \_entity\_poly
  + \_entity\_poly.entity\_id
    - Used to determine the entity that the rest of the \_entity\_poly information pertains to.
  + \_entity\_poly.type
    - Used to determine the type of the entity (e.g. polypeptide(L)).
  + \_entity\_poly.pdbx\_seq\_one\_letter\_code\_can
    - Used to determine the nucleotide or amino acid sequence of the entity.
  + \_entity\_poly.pdbx\_strand\_id
    - Used to determine the single character codes corresponding to the entity (the 'a', 'A', 'b', '1', etc. that comes after the entry e.g. 3A0Ba, 3A0BA, ...).
* \_exptl
  + \_exptl.method
    - The experimental method that was used to determine the structure.
* \_atom\_site
  + \_atom\_site.label\_atom\_id
    - Records the type of atom. Used to determine if the entity contains only alpha carbon atoms.
  + \_atom\_site.label\_entity\_id
    - Used to determine the entity that the rest of the \_atom\_site information pertains to.
* \_refine
  + \_refine.ls\_d\_res\_high
    - Used to determine the resolution of the structure in an X-ray diffraction experiment.
  + \_refine.ls\_R\_factor\_obs
    - Used to determine the R-value. This is the R-value measurement used when giving an upper bound on the R-value during culling.
  + \_refine.ls\_R\_factor\_R\_free
    - Used to determine the R-free value.
* \_reflns
  + \_reflns.d\_resolution\_high
    - Used to determine the resolution of the structure in an X-ray diffraction experiment.
* \_struct\_ref
  + \_struct\_ref.entity\_id
    - Used to determine the entity that the rest of the \_struct\_ref information pertains to.
  + \_struct\_ref.db\_name
    - Used to determine the name of the external database linked to the entity.
  + \_struct\_ref.db\_code
    - Used to determine the code which can be used to access information about the entity in the external database
* \_entity\_src\_gen
  + \_entity\_src\_gen.entity\_id
    - Used to determine the entity that the rest of the \_entity\_src\_gen information pertains to.
  + \_entity\_src\_gen.pdbx\_gene\_src\_scientific\_name
    - Used to determine the scientific name of the organism that the entity came from.
* \_entity\_src\_nat
  + \_entity\_src\_nat.entity\_id
    - Used to determine the entity that the rest of the \_entity\_src\_nat information pertains to.
  + \_entity\_src\_nat.pdbx\_organism\_scientific
    - Used to determine the scientific name of the organism that the entity came from.
* \_pdbx\_entity\_src\_syn
  + \_pdbx\_entity\_src\_syn.entity\_id
    - Used to determine the entity that the rest of the \_pdbx\_entity\_src\_syn information pertains to.
  + \_pdbx\_entity\_src\_syn.organism\_scientific
    - Used to determine the scientific name of the organism that the entity came from.

Both the resolution and the scientific name of the organism can be determined from multiple mmCIF fields. When determining the value for the resolution, only one of the \_refine and \_reflns records is used (with preference given to the \_refine record). When determining the organism that the entity originates from, only one of \_entity\_src\_gen, \_entity\_src\_nat or \_pdbx\_entity\_src\_syn is used (with preference given in the order \_entity\_src\_nat, \_entity\_src\_gen and finally \_pdbx\_entity\_src\_syn).

Following the parsing of the mmCIF files, and the extraction of the important data, the extracted data undergoes post-processing. First the type of each chain in the entries is recorded. The possible chain types are:

* DNA
* DNA/polysaccharide
* DNA/polysaccharide/RNA
* DNA/RNA
* NonProtein
* polysaccharide
* polysaccharide/RNA
* Protein
* RNA

The ‘NonProtein’ type is not recorded as such in the mmCIF files. However, if 50% or more of the amino acids in an amino acid sequence are X (i.e. unspecified or unknown), then the entity is marked as a NonProtein, and is not included in the list of proteins. Post-processing of the data for other mmCIF fields is as follows:

* If there is no value recorded for the structural resolution, then the resolution is set to 100.
* If there is no value recorded for the R value, then it is set to 1.
* If there is no value recorded or the free R value, then it is set to 1.
* If there are no non-alpha carbon atoms in the structure, then the chain is marked as containing only backbone alpha carbon atoms.
* The experimental method used to determine the structure is converted into an abbreviated form.

The abbreviations used for the experimental method are:

* ELECTRON MICROSCOPY EM
* FIBER DIFFRACTION FIBER
* INFRARED SPECTROSCOPY FTIR
* NEUTRON DIFFRACTION NEUTRON
* SOLUTION NMR NMR
* SOLID-STATE NMR NMR
* POWDER DIFFRACTION POWDER
* X-RAY DIFFRACTION XRAY
* FLUORESCENCE TRANSFER NA
* ELECTRON CRYSTALLOGRAPHY NA
* SOLUTION SCATTERING NA
* Anything else is marked as NA.

Once all the parsing, extraction and post-processing is complete, the updating of the local PDB data can begin. The first task is to make a first pass at determining the representative chains for the entries. This is only performed for protein chains, as information is not recorded about other types of chains. For each entry the within entry representatives can be determined without reference to any other entries. The chains recorded in an entry are first grouped based on the amino acid sequence of the chain. Then, one chain from each grouping is arbitrarily chosen to be the within entry representative for the grouping. Following this the type of each chain in the subset of parsed PDB entries is recorded.

The next step is to determine which entries in the database, if any, need deleting. An entry needs deleting if the weekly update of the local PDB copy removes the entry. Deleting an entry involves removing the record of the entry, along with all of its chains, their type and the within entry representative information. The similarity and PDB wide representative information held about the entry’s chains also needs updating. The process by which this is done is as follows:

1. Delete all representative data where one of the entry’s chains is the non-representative chain.
2. Determine all of the entry’s chains which are PDB wide representative chains. If there are PDB wide representative chains go to step 3, else delete all similarity data recorded for the entry’s chains.
3. For each PDB wide representative chain REP:
   1. Determine the set of chains, NONREP, that are represented by REP.
   2. Determine which chain, NEWREP, in NONREP should be the new representative chain. Chains where the structure was determined by X-ray diffraction take precedence over other. If there are multiple X-ray diffraction proteins, or none, ties are broken in favour of the chains with the highest resolution structure. Further ties are broken y chains with the best recorded R value. If there are still ties, then one of the chains is chosen arbitrarily to be the new representative chain.
   3. All PDB wide representative records that have the representative chain as REP are updated to have the representative chain be NEWREP.
   4. All similarity records where one of the chains involved is REP, are updated by replacing REP with NEWREP.

Once any entries that need deleting are deleted, entries that have been updated in the last week can be examined. A newly updated entry could either be a completely new entry, or an entry that already exists and needs updating. If the entry is a new entry, then the steps for updating the database are as follows:

1. Add a record of the entry identifier to the database.
2. For each of the entry’s chains, add a record of the type of the chain. If the chain is a protein:
   1. Add a record for the protein that records the protein’s data (e.g. resolution, R value, experimental method, etc.).
   2. Determine if the chain is recorded as being a within entry representative. If it is, then record all the within entry representative information in the database.
   3. Determine if the chain has the same sequence as an already recorded chain.
   4. If 2.b and 2.c are False, then the updating for the chain is finished. If2.b is True and 2.c is False, then record the within entry information as the PDB wide representative information and record the fact that the sequence found is new. If both 2.b and 2.c are True, then record the within entry representative information and go to step 2.e.
   5. Find the PDB wide representative chain that has the same sequence as the chain being updated. Determine whether the chain being updated or the currently recorded representative should be the representative. The current representative should not stay as the representative in any of the following three situations:
      1. If the structure of the current representative was not determined by X-ray diffraction, and the structure of the chain being updated was.
      2. If neither structure was determined using X-ray diffraction, or both were, but the resolution of the structure of the chain being updated is greater.
      3. If neither structure was determined using X-ray diffraction, or both were, the resolutions are equal but the R value of the chain being updated is better.
      4. If neither structure was determined using X-ray diffraction, or both were, the resolutions are equal, the R values are equal but the identifier of the chain being updated is alphanumerically less than the identifier of the current representative.
   6. If the current representative should stay as the PDB wide representative, then make a record that the PDB wide representative for the chain being updated, and all of the chains within its entry that it represents, are represented by the current PDB wide representative.
   7. If the representative should change, the procedure is as follows:
      1. Update all entries that have the current representative as the representative chain, and make them record the chain currently being updated as their PDB wide representative.
      2. Add an entry that records the fact that the current representative is to be represented by the chain currently being updated.
      3. Add records for all the chains within its entry that it represents, that the chain currently being updated is also the PDB wide representative for those chains.
      4. Alter all similarity records that have the current representative as one of the two chains, so that they have the chain currently being updated in its place.

If the entry already exists in the database, then its record simply needs updating. The first step for each chain in the entry is to update the record of the type of the chain. If the chain is not a protein, then this is all that needs doing. For protein chains, the following further update steps are necessary:

1. Record a copy of the data (e.g. resolution, R value, experimental method, etc.) about the chain that is currently recorded.
2. Update the data recorded about the protein.
3. If there are any within entry representative records where the chain is the representative chain, delete them.
4. If the chain is not a within entry representative, then stop the updating. IF the chain is a within entry representative then go to step 5.
5. Add a record to the database recording the within entry representative information.
6. If the sequence of the chain has changed:
   1. Remove all records in the PDB wide representative table where the non-representative chain is in the same entry as the chain being updated. This is done as the sequence of the chains in the same entry that the chain being updated represent, will also have had their sequence change.
   2. Determine if the chain is a PDB wide representative. If it isn’t, remove all records in the PDB wide representative table where it is recorded as being a non-representative chain and go to step 7, else go to step 6.c.
   3. Determine the PDB wide chains represented by the chain being updated.
   4. Determine which of the chains gathered in step 6.c should be the new representative. Chains where the structure was determined by X-ray diffraction are chosen first. If there is more than one chain which had its structure determined y X-ray diffraction, or none, then the new representative chain will be the one with the highest resolution structure. If the resolution cannot determine which chain should be the representative, then the chain with the best R value will be the new representative. If these three steps are unable to determine one chain to be the representative chain, then the representative is chosen arbitrarily from the equally good chains.
   5. Remove all records in the PDB wide representative table where the chain is recorded as the representative. This is done as the chain can no longer represent the other chains since its sequence has changed.
   6. Alter any similarity records that contain the chain being updated as one of the chains. The records are altered so that the chain being updated is replaced by the newly chosen representative chain.

The next step for protein chains, is to check if the sequence of the chain is the same as any other chain already stored in the database. If it isn’t, then if the chain is a within entry representative, the within entry representative information for the chain is recorded as PDB wide representative information. If the sequence of the chain is new, e.g. the chain’s sequence was updated to become a sequence that didn’t already exist in the database, then the chain is recorded as having a sequence that is new to the database. If the chain’s sequence is already in the database, then the following steps are carried out:

1. Find the PDB wide representative chain that has the same sequence as the chain being updated. Determine whether the chain being updated or the currently recorded representative should be the representative. The current representative should not stay as the representative in any of the following three situations:
   1. If the structure of the current representative was not determined by X-ray diffraction, and the structure of the chain being updated was.
   2. If neither structure was determined using X-ray diffraction, or both were, but the resolution of the structure of the chain being updated is greater.
   3. If neither structure was determined using X-ray diffraction, or both were, the resolutions are equal but the R value of the chain being updated is better.
   4. If neither structure was determined using X-ray diffraction, or both were, the resolutions are equal, the R values are equal but the identifier of the chain being updated is alphanumerically less than the identifier of the current representative.
2. If the representative chain should not be changed, then record the current representative as being the PDB wide representative for the chain being updated and all of the chains it represents within its entry.
3. If the representative chain should be changed, then:
   1. Update all PDB wide representative records where the representative chain is recorded as being the current representative. These records should be updated so that the representative chain is the chain being updated.
   2. Add a record that the current representative is now represented by the chain being updated.
   3. Update all similarity records where one of the two chains is the current representative. The chain in the similarity record that is the current representative should be updated to be the chain being updated (i.e. the new PDB wide representative).
   4. Record the within entry representative information for the chain being updated (i.e. the new PDB wide representative) as also being PDB wide representative information.

Once all the updating has been performed, it is possible that the pairwise sequence identity between individual chains needs to be calculated. The steps involved in this are as follows:

1. Find the representative chains of the newly added/updated chains that have a sequence that is not already in the database.
2. Extract all the sequences of the representative chains that were already stored in the database.
3. Create the FASTA format files.
   1. A file containing all representative chains of the newly added/updated chains with new sequences.
   2. A file containing all the representative chains that were already stored in the database.
   3. A file containing all the chains in 3.a and 3.b.
4. First, perform an all against all BLAST operation with the FASTA file dataset produced in 3.a against the one produced in 3.c. This will generate the pairwise sequence identity for the new sequences against both the new and old sequences.
5. The second step is to perform an all against all BLAST operation of the FASTA file dataset produced in 3.b against the one produced in 3.a. This will generate the pairwise sequence identities for all the already stored chains against the newly added chains. This step is necessary as BLAST is not symmetrical. For example, BLASTing protein A against protein B may give different results than BLASTing protein B against protein A.
6. Following this step the new sequence identities generated are processed, and recorded in the database. Processing is done to ensure that the greatest sequence identity is recorded for each pair of chains.

## Culling By Chains

The user input is first parsed to determine the chains that have been entered, and to determine the entries that have been entered. If there are any invalid chains or entries the program is aborted, and returns an error message. Following this, the program steps are as follows, with numbers corresponding to the numbered flowchart process:

1. The first step involves determining if there are any entries in the user supplied input. All entries in the input are replaced by the entry’s chains. For example, if there is an entry XXXX that has chains A and B, then XXXX in the user input is replaced by chains XXXXA and XXXXB. This step produces a set of chains, VVVV, which consists of all the valid user input chains.
2. The next step is to generate a second set of chains by removing any chains that do not meet the user supplied structural/chain criteria. For example, any chains where the resolution is too low or high, the R value is too low or high, etc. are not included in the list of chains to be culled. It is possible that no chains in VVVV meet the structural/chain criteria. In this case the non-redundant dataset generated by the program will contain no chains. The result of this step is a set of non-representative PDB chains, NNNN.
3. The chains that reach this step will all be valid PDB chains, and will meet the structural/chain criteria. The next step is to generate a set RRRR, of representative chains. This involves selecting the fewest representative chains, so that all the chains in NNNN have a chain which represents them in RRRR. Any representative chain in NNNN will be copied directly to RRRR, while any non-representative chain in NNNN will have a chain which represents it in RRRR. In addition to determining the representative set RRRR, this step is also used to generate a one to many mapping between the elements of RRRR and NNNN. This mapping is used when determining which chains are to be kept following the culling by Leaf.
4. Once the representative set RRRR has been generated, the percentage sequence identity between the chains in RRRR can be determined.
5. The culling is performed using Leaf and the sequence identities generated in step 4. This step returns a set of chains, CCCC, which contains the chains which need to be removed from the dataset to make it non-redundant. However, the set CCCC is composed of a subset of the representative chains from RRRR, and further processing is required to determine the user input chains that should be kept and removed.
6. The user input chains to keep, KKKK, are determined using RRRR, CCCC and the mapping between RRRR and NNNN calculated in step 3. First, the set KKRR, the set of chains in RRRR but not CCCC, is calculated. KKRR comprises the representative proteins that make up the non-redundant dataset. In order to determine the user input chains that should make up the non-redundant dataset, the chains in KKRR are mapped to chains in NNNN using the RRRR to NNNN mapping. As this mapping is a one to many mapping, it is possible that a chain in KKRR will map to multiple chains in NNNN. The mapping is therefore done as follows:
   * If the chain in KKRR maps to only one chain in NNNN, then that one chain is added to KKKK.
   * If the chain in KKRR maps to more than one chain in NNNN, then one of the chains mapped to chains is arbitrarily added to KKKK.
7. At this point the set KKKK contains all the user input chains that should be in the non-redundant dataset. In order to determine the user input chains which should be removed from the dataset, the chains in KKKK are removed from the chains in VVVV. This gives the final set of chains to be removed, REM=VVVV-KKKK.

The two sets of chains, KKKK and REM, can be output, along with statistics about the chains in KKKK.

### Example

1. Shows the validated user input, and the conversion of an entry input to its constituent chains.
2. Shows the input chains that remain after the structural/chain criteria are taken into account. In this case chains 2, 5 and 9 did not meet the user specified structural/chain criteria, and so are not included in the further redundancy removal.
3. Shows the mapping of the non-representative chains to representative ones, in black arrows, and the reverse mapping of representative chains to the non-representative chains that they represent, in blue arrows. User input chain 1 is a representative chain in its own right, and therefore there is no representative mapping recorded for it.
4. Shows the sequence identity between the representative chains.
5. Shows the result of the Leaf redundancy removal method. The white nodes are the ones that need to be removed from the dataset to make it non-redundant.
6. Shows the mapping from the representative chains that should be kept, to the user input chains that should be kept. The chains to keep are in red, while those to remove are in white. The blue arrows between representative chains and non-representative chains are from the reverse mapping calculated in step 3, Again chain 1 does not need to be mapped to any other chain, as it is itself a representative chain. The choice of whether to keep chain 3 or 4, and whether to keep chain 9 or 10, is arbitrary. Only one can be kept as they mapped to the same representative chain, and are therefore have identical sequences.
7. Shows the final non-redundant dataset, the red chains, and the chains that were removed from the dataset, the white chain.

## Culling By Entries

The user input is first parsed to determine the entries that have been entered. If there are any invalid entries the program is aborted, and returns an error message. Following this, the program steps are as follows, with numbers corresponding to the numbered flowchart process:

1. The first step is to create a set, VVVV, of the chains of all the user input entries. For example, if one of the entries input by the user is XXXX, and this entry has two chains A and B, then VVVV will contain the chains XXXXA and XXXXB. This is done for every user input entry.
2. The next step is to generate a second set of chains by removing any chains that do not meet the user supplied structural/chain criteria. For example, any chains where the resolution is too low or high, the R value is too low or high, etc. are not included in the list of chains to be culled. It is possible that no chains in VVVV meet the structural/chain criteria. In this case the non-redundant dataset generated by the program will contain no entries. The result of this step is a set of non-representative PDB chains, NNNN.
3. The chains that reach this step will all be valid PDB chains, and will meet the structural/chain criteria. The next step is to generate a set RRRR, of representative chains. This involves selecting the fewest representative chains, so that all the chains in NNNN have a chain which represents them in RRRR. Any representative chain in NNNN will be copied directly to RRRR, while any non-representative chain in NNNN will have a chain that represents it in RRRR. In addition to determining the representative set RRRR, this step is also used to generate a one to many mapping between the elements of RRRR and NNNN. This mapping is used when determining the similarity between entries.
4. Once the representative set RRRR has been generated, the percentage sequence identity between the chains in RRRR can be determined.
5. Once the sequence identity between the representative chains is known, the similarity between user input entries can be determined. First the mapping from RRRR to NNNN is used to determine the entries that the representative chains map to. This can be done as the first four characters of a chain identifier is the identifier of the entry that the chain belongs to. Therefore, rather than mapping from the chains in RRRR to the chains in NNNN, the mapping is done from the chains in RRRR to the entries of the chains in NNNN. For example, if chain XXXXA in RRRR maps to (i.e. represents) chain ZZZZA in NNNN, then XXXXA would map to entry ZZZZ. As the percent sequence identity between chains in RRRR is known, the percent sequence identity between two entries can be determined at the same time as the mapping from chains in RRRR to entries in NNNN occurs. The sequence identity between any two entries E and F, is the maximum of the sequence identity between any of the chains in E and any of the chains in F. If two chains in RRRR have a sequence identity greater than the threshold supplied, and the two chains map to the same entry, then there is no self loop induced in the protein similarity graph. An example of this step that clarifies the method of mapping sequence identities from RRRR to entries can be seen BELOW.
6. The culling is performed using Leaf and the sequence identities generated in step 5. This step returns a set of entries, EEEE, which contains the entries that need to be removed from the dataset to make non-redundant.
7. The user input entries to keep, KKKK, are calculated as KKKK=VVVV-EEEE. If intra-entry culling is not going to be performed, then jump to step 9, else go to step 8.
8. Intra-entry culling is used to make sure that no two chains in the same entry have a sequence identity greater than a given percentage. For each entry, the process is as follows:
   * Determine the chains in the entry.
   * Determine the representative chains of the chains in the entry.
   * Determine the sequence identity between the representative chains.
   * Use Leaf to determine which representative chains to remove.
   * Determine the non-representative chains, i.e. the chains in the entry, that should be kept.
9. The chains to keep, CCCC, are calculated in one of two ways:
   * If intra-entry culling was performed, then CCCC is comprised of the chains in each entry that were not removed.
   * If intra-entry culling was not performed, then CCCC is comprised of every chain from every entry in KKKK.

The two sets of entries, KKKK and EEEE, can be output, along with statistics about the entries in KKKK and a FASTA file of the chains in CCCC.

### Example

1. Shows the mapping of the valid user input entries to their constituent chains.
2. Shows the removal of chains 3, 4, 7 and 8 for not meeting the user’s structural/chain criteria. Chains 1 and 2 can meet the structural/chain criteria even though chain 3 does not. This is because the sequence of chain 3 can be of a different length to chains 1 and 2, and the length of chain 3 may not be within the user specified sequence length range.
3. Shows the mapping of the non-representative chains to representative ones, in black arrows, and the reverse mapping of representative chains to the non-representative chains that they represent, in blue arrows.
4. Shows the sequence identity between the representative chains.
5. Shows the process of mapping of representative chains to non-representative chains to entries. The blue arrows are from the mappings of representative chains to non-representative, calculated in step 3. The non-representative chains are then mapped to the entries they come from. The sequence identities between representative chains are also mapped to the entries. There are two demonstrations of special cases in this step.
   1. E3 and E5 have two different sequence identity values that could be assigned to them. This is because chains 11 and 12 have 40% sequence identity with chain 5, while chains 9 and 10 have 50% sequence identity to chain 6. In this case the greater of the two sequence identities is chosen to be the sequence identity between E3 and E5.
   2. E6 and E7 are given a sequence identity of 100%. This cannot be inferred from the sequence identities between representative chains, as for this to be the case R6 would need an edge to itself with 100% sequence identity. However, whenever the chains of two different entries map to the same representative chain, the entries should be given a 100% sequence identity. This is because some of their chains have a 100% sequence identity, as they are represented by the same chain.
6. Shows the results of using Leaf to remove redundancy. The white entries are to be removed.
7. Shows the final non-redundant dataset of entries. Red entries are to be kept in the on-redundant dataset, while white entries should be removed.
8. Shows two different methods of determining which chains are to be presented as being a non-redundant dataset.
   1. In this approach all chains from the non-redundant entries are recorded as being in a non-redundant dataset. This method corresponds to having no intra-entry culling.
   2. In this method intra-entry culling is performed. Chain 2 is removed as it has the same sequence as chain 1, while chain 15 is removed as it has the same sequence as chain 14. The removal of chain 6 depends on both its pairwise sequence identity with chain 5, and the percentage sequence identity threshold given for the intra-entry culling.