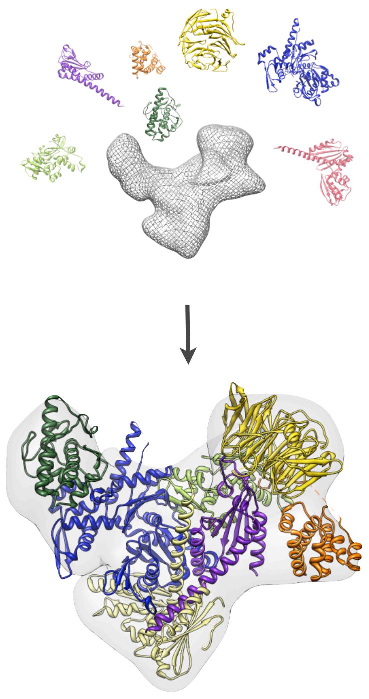
Tutorial on fitting multiple proteins into a cryoEM of their assembly using MultiFit

Keren Lasker, Ben Webb and Andrej Sali

Goal

In this tutorial we will use MultiFit (Lasker et al, JMB, 2009) to determine the protein configuration of the ARP2/3 complex from its individual protein models guided by a density map of the entire complex.



Input

The input to MultiFit consists of a density map of an assembly, and atomic models of the individual subunits. A density map of the ARP2/3 complex was simulated to 2nm from an ARP2/3 crystal structure (PDB access code 1TYQ) using the pdb2mrc command in EMAN. Structures of the 7 proteins (ARP2,ARP3,ARC1-ARC5) were obtained from the PDB. All of the relevant input file are found in $MultiFit_tutorial/data. The tutorial steps are summarized in $MultiFit_tutorial/tutorial_main.sh

Step 1: Generate an assembly input data file

To make use of various automation scripts needed for the next steps, we build an assembly input file. The assembly input file can be generated automatically using the build_assembly_input_file.py script.

The input file consists of a subunits part and an assembly part.

Each line in the subunits part consists of data about one of the subunits:

Field name	Description	Example

name	An id for the subunit	ARP2
protein	Path to the corresponding pdb file	data/models/ ARP2.pdb
pdb_anchor_points	Path to the corresponding anchor points file (generated by gmm_pdb)	results/ARP2.gmm.pdb
num_gaussians	The number of 3G Gaussian components used to describe the subunit	8
transformations	path to a file containing results for fitting the subunit into the density map (generated by gmm_fitting)	results/ARP2.fitting.output
ref filename	Optional parameter: Path to a file containing the native structure of the component. The structure should be positioned as found in the assembly for model RMSD calculations. If no reference structure is provided, leave the field empty.	data/models/ ARP2.fitted.pdb

The assembly part consists of a single line consisting of the following data:

Field name	**Description**	**Example**
map	path to the assembly density map in MRC format	data/1tyq_20.mrc
resolution	The resolution of the density map	20.
spacing	A/pixel	3.
x-origin	X coordinate origin of the density map	0
y-origin	Y coordinate origin of the density map	0
z-origin	Z coordinate origin of the density map	0
coarse_anchor_points	path to the corresponding pdb anchor points file (generated by gmm_em). This file would contain a detailed GMM, which would be used for protein fitting.	results/ 1tyq.fine.gmm.pdb
fine_anchor_points	path to the corresponding pdb anchor points file (generated by gmm_em). This file would contain a corases GMM, with one Gaussian per protein/component. This file would be used in the	results/1tyq.coarse. gmm.pdb

	multiple fitting procedure.	

```
name|protein| pdb_anchor_points| transformations|ref filename|
A|$MULTIFIT/benchmarks/em_proteomics/data/1tyq/1tyq_A.pdb|1tyq_A_anchor_points.pdb|8|1tyq_A_fitting.output|$MULTIFIT/benchmarks/em_proteomics/data/
1tyq/1tyq_A.fitted.pdb|
B|$MULTIFIT/benchmarks/em_proteomics/data/1tyq/1tyq_B.pdb|1tyq_B_anchor_points.pdb|5|1tyq_B_fitting.output|$MULTIFIT/benchmarks/em_proteomics/data/
1tyq/1tyq_B.fitted.pdb|
C|$MULTIFIT/benchmarks/em_proteomics/data/1tyq/1tyq_C.pdb|1tyq_C_anchor_points.pdb|7|1tyq_C_fitting.output|$MULTIFIT/benchmarks/em_proteomics/data/
1tyq/1tyq_C.fitted.pdb|
D|$MULTIFIT/benchmarks/em_proteomics/data/1tyq/1tyq_D.pdb|1tyq_D_anchor_points.pdb|6|1tyq_D_fitting.output|$MULTIFIT/benchmarks/em_proteomics/data/
1tyq/1tyq_D.fitted.pdb|
E|$MULTIFIT/benchmarks/em_proteomics/data/1tyq/1tyq_E.pdb|1tyq_E_anchor_points.pdb|4|1tyq_E_fitting.output|$MULTIFIT/benchmarks/em_proteomics/data/
1tyq/1tyq_E.fitted.pdb|
F|$MULTIFIT/benchmarks/em_proteomics/data/1tyq/1tyq_F.pdb|1tyq_F_anchor_points.pdb|4|1tyq_F_fitting.output|$MULTIFIT/benchmarks/em_proteomics/data/1
tyq/1tyq_F.fitted.pdb|
G|$MULTIFIT/benchmarks/em_proteomics/data/1tyq/1tyq_G.pdb|1tyq_G_anchor_points.pdb|3|1tyq_G_fitting.output|$MULTIFIT/benchmarks/em_proteomics/data/
1tyq/1tyq_G.fitted.pdb|
map| resolution| spacing| x-origin| y-origin| z-origin|fine_pdb_anchor_points|coarse_pdb_anchor_points|
$MULTIFIT/benchmarks/em_proteomics/data/1tyq/1tyq_20.mrc| 20.0| 3.| 0.| 0.| 0.|1tyq_20.fine.gmm.pdb|1tyq_20.coarse.gmm.pdb
```

The assembly input file for ARP2/3

Step 2: Learn a reduced representation of the entire assembly and the individual subunits

Learn a Gaussian Mixture Model of the density map

To determine the reduced representation of an assembly density map into *X* 3D Gaussians, simply run:

/opt/multifit/bin/gmm_em density.mrc X dens_threshold density.X.gmm.pdb

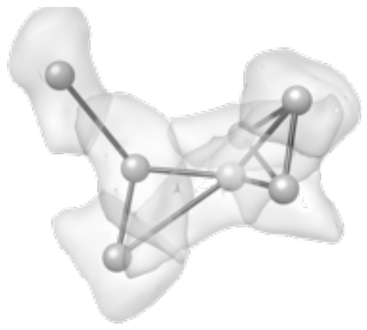
The function will determine a Gaussian Mixture Model of X components that best explain the configuration of all voxels with density value above dens_threshold. The default output of the function is a set of the Gaussians centers encoded as CA atoms. In some cases the function fails to read the a/pix and origin from the mrc file. It is advised to provide them as input parameters as follows:

--apix arg	*the a/pix of the density map*
--x arg	*the X origin of the density map*
--y arg	*the Y origin of the density map*

--z arg *the Z origin of the density map*

The function can provide others types of outputs, such as a .mrc file for each segment. To learn more read the help documentation of the function.
For example, to segment the assembly into densities corresponding to its 7 proteins, simply run:

```
/opt/multifit/bin/gmm_em data/1tyq_20.mrc 7 700 1tyq.dens.7.proteins.pdb –seg 1tyq.dens.protein
```



Learn a Gaussian Mixture Model of an atomic component
To determine a reduced representation of a protein into *X* 3D Gaussians, simply run:

```
/opt/multifit/bin/gmm_pdb prot.pdb X gmm_results.pdb
```

The function will learn a Gaussian Mixture Model of *X* components that best describe the configuration of the protein atoms. The function uses all atoms as default but can work on backbone atoms alone by indicating the –backbone flag.
For example, we can produce a reduced representation of ARP3, consisting of 8 Gaussians by running:

```
/opt/multifit/bin/gmm_pdb data/models/1tyq_A.pdb X 1tyq_A.gmm.pdb
```

How to decide the number of Gaussians (*K*) for each protein?
Deciding on the number of Gaussians used to describe a protein is something of an art. Some rules of thumb:

1. Require at least 3, as we need three points for the fitting.

2. Have each Gaussian “covering” the same amount of residues. If you choose, for example, 50 residues per Gaussians, a protein of 170 residues should have 3 Gaussians and one with 260 residues should have 5 Gaussians.

3. The number of Gaussians of the assembly should be equal to the sum of the Gaussians of all of the individual proteins.

4. Advanced: Plot the resulting likelihood function of the GMM clustering procedure for different values of *K* and searching for a point of large drop of the curve.

5. To estimate the number of Gaussians for each protein, run the script:

```
/opt/multifit/utils/anchor_point_estimator.py assembly.input num_of_residues
```

Automation

To run gmm_pdb and gmm_em on all relevant subunits, you can run the script:

```
/opt/multifit/utils/run_anchor_points_detection.py assembly.input
density_threshold
```

Step 3: Fit each protein to the map

We now use the spatial configuration of the calculated GMMs to efficiently fit proteins in the density.

To fit each protein to the density, run the script:

```
/opt/multifit/utils/run_protein_fitting.py assembly.input multifit.par
```

multifit.par – sets values for parameters needed for the fitting procedure (Supplementary materials).

The script calls anchor_point_fitting on each of the proteins:

```
/opt/multifit/bin/anchor_point_fitting
                    Usage: anchor_point_fitting <density.mrc> <a/pix>
                    <resolution> <protein> <density_anchor_points.pdb>
                    <protein_anchor_points.pdb>
--x arg              the X origin of the density map
  --y arg              the Y origin of the density map
  --z arg              the Z origin of the density map
  --local-rad arg       Preform fitting around the centter of the input
                    molecule with a given radius (it is recommended to set
                    the radius to be at least the value of the  resolution)
  --ref arg            a PDB file of the protein fitted to the density map
                    (used for benchmarking)
  --sol arg            all solutions will be printed as PDB format and will be
                    named <sol>_i.pdb
  --output-file arg     output filename
  --param-file arg      parameters used by anchor_point_fitting
```

If the reference structure is known and the method is used for benchmarking, use /opt/multifit/utils/ fetch_best_sampled_transformations.py
The results for our fitting procedure are:

Step 4: Infer the optimum solution

Calculate scores

Next, we calculate all relevant scores. As this is the most time consuming step, we will not run it and just use the scores that are in the scores directory. For completeness, in order to generate all scoring data, run:
/opt/multifit/utils/run_all_scores.py assembly.input

Enumerate configurations

Finally, we efficiently enumerate all possible configurations to find the best scoring assembly models.
/opt/multifit/utils/run_multifit.py assembly.input assembly.jt results/configurations.output
In case you have the native structure and would like to assess RMSD to native, use:

```
for:  data/models/1tyq_A.pdb  best fit of index   0   with rmsd  5.91062
for:  data/models/1tyq_B.pdb  best fit of index  11  with rmsd  3.09233
for:  data/models/1tyq_C.pdb  best fit of index  14  with rmsd  7.9537
for:  data/models/1tyq_D.pdb  best fit of index   1   with rmsd  7.8815
for:  data/models/1tyq_E.pdb  best fit of index   5   with rmsd  3.88972
for:  data/models/1tyq_F.pdb  best fit of index   3   with rmsd  5.61873
for:  data/models/1tyq_G.pdb  best fit of index   8   with rmsd  11.0771
```

```
/opt/multifit/utils/run_multifit.py assembly.input assembly.jt
results/configurations.output data/models/1tyq.fitted.pdb
```

The configuration.output file contains the final models. For example a result can be:

```
|configuration index|score|rmsd to ref|configuration file|
|A:17,B:5,C:18,D:8,E:1,F:0,G:0|16.8515000343|10.1418685913|conf.0.pdb|
```

Step 5: Reranking by proteomics data

For challenging molecules, a score composed of fitting and geometric complementarity is not sufficient for an unambiguous determination of the assembly architecture and additional restraints should be provided. Here, we demonstrate how proteomics data can be converted into spatial restraints.
For illustrate we chose two models, sampled by MultiFit, both ranked similarly and show how additional restraint derived from proteomics data can resolve the ambiguity.

```
/opt/multifit/utils/rescoring_by_proteomics.py ARP23.wrong.model.xml
restraints.xml
/opt/multifit/utils/rescoring_by_proteomics.py ARP23.good.model.xml
restraints.xml
```

Supplementary materials

Advanced Parameters

Advanced parameters can be tuned in multifit.par file.

Tuning parameters for Anchor point detection

Parameter name	**Default value**	**Description**
NUM_RESIDUES_FOR_ANCHOR_POINT	50	We should have at least 5 anchor points for each protein. If the number of residues of the protein is lower than 250, adjust this parameter accordingly.

MAX_EXC_VOL	2.0	Maximum excluded volume allowed
NUM_FITTING_SOLS	20	number of fitting solutions to consider for each protein
FRACTION_MIN_CENTER_DIST	0.66	The distance between the proteins centroids should be at least 2/3 of their radii sum
FRACTION_MAX_CENTER_DIST	1.33	The distance between the proteins centroids should be at most 4/3 of their radii sum
PROTEIN_1_CENTROID		the centroid of the second protein should be close to this point
PROTEIN_2_CENTROID		the centroid of the first protein should be close to this point
DISTANCE_FROM_CENTROID	5.0	only consider transformation that center the protein within this value from the provided centroids

Tuning parameters for fitting by point alignment

Parameter name	**Default value**	**Description**
CUBE_SIZE	3.	
QUERY_RADIUS	20.	
MAX_RMSD_IN_ALIGN	15	only consider matching for which the alignment rmsd is below this value
MAX_ANGLE_DIFF	0.1	Maximum rotational difference between solutions in the same cluster
MIN_CLUSTER_SIZE	3	Clusters with less members are not considers as candidates for good fits
MAX_TRANS_DIFF	5.	Maximum translational difference between solutions in the same cluster

MultiFit in a nutshell

Motivation

Models of macromolecular assemblies are essential for a mechanistic description of cellular processes. Such models are increasingly obtained by fitting atomic-resolution structures of components into a density map of the whole assembly. Yet, current density-fitting techniques are frequently insufficient for an unambiguous determination of the positions and orientations of all components.

The algorithm

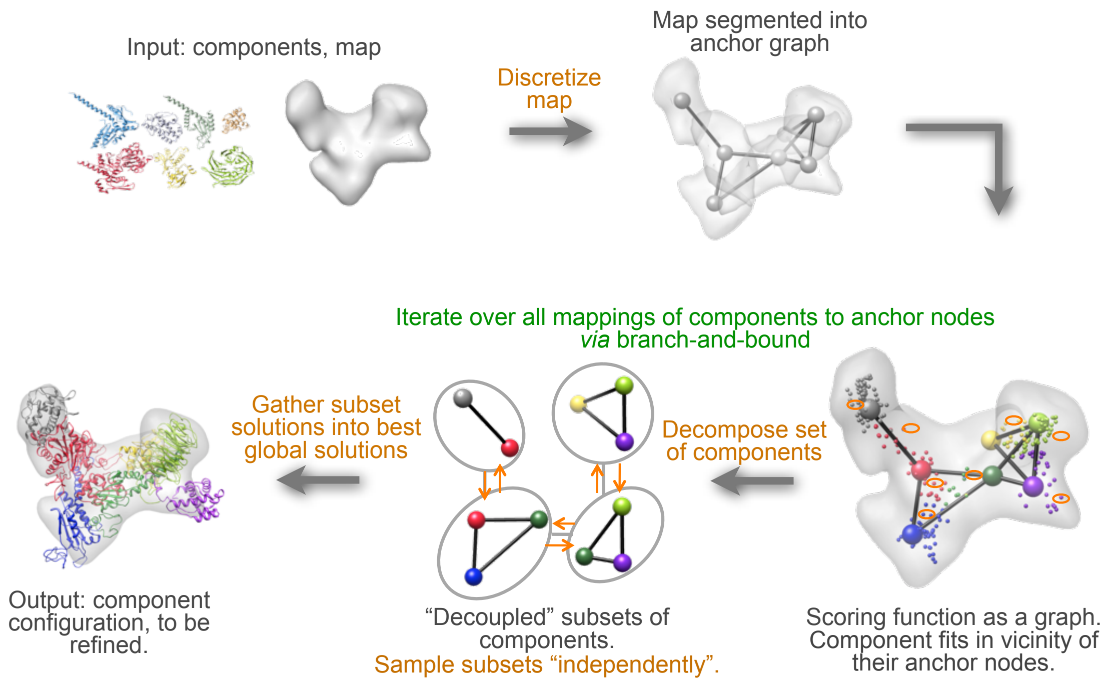
MultiFit is a computational method for simultaneously fitting atomic structures of components into their assembly density map at resolutions as low as 25 Å. The component positions and orientations are optimized with respect to a scoring function that includes the quality-of-fit of components in the map, the protrusion of components from the map envelope, as well as the shape complementarity between pairs of components. The scoring function is optimized by an exact inference optimizer DOMINO that efficiently finds the global minimum in a discrete sampling space.

We express this structure characterization challenge as a combinatorial optimization problem. Next, we outline a representation of the modeled system, a scoring function, and an optimization algorithm.

Representation. The assembly density map is represented by a three-dimensional (3D) grid, in which every voxel is assigned an estimated density value. The components are represented by their atoms and remain rigid throughout the entire optimization process.

Scoring. Potential configurations are evaluated based on the quality-of-fit of individual components in the density map, the protrusion of each component from the map envelope, as well as the shape complementarity between pairs of components.

Optimization. The component configuration that optimizes the scoring function is identified by a combinatorial optimization protocol, consisting of three stages: (i) anchor graph construction, (ii) coarse-grained sampling, and (iii) fine-grained sampling. In anchor graph construction, the density map is discretized into regions and the connectivity between them is calculated. In coarse-grained sampling, the sampling space is first discretized by fitting each of the components into each of the map regions and selecting a number of top-ranking placements for each component in each region. Next, a branch-and-bound search through all mappings of components to regions combined with DOMINO (a divide and conquer sampler based on message passing algorithm on graphs finds top 20 scoring configurations. In fine-grained sampling, each of these top configurations is refined by DOMINO; a refined sampling space is generated for each coarse configuration by docking pairs of its interacting components and selecting only those placements that are approximately consistent with the initial coarse configuration.


Input: components, map
Discretize map
Map segmented into anchor graph
Iterate over all mappings of components to anchor nodes *via* branch-and-bound
Decompose set of components
Scoring function as a graph. Component fits in vicinity of their anchor nodes.
"Decoupled" subsets of components.
Sample subsets "independently".
Gather subset solutions into best global solutions
Output: component configuration, to be refined.