

Fitting multiple structures into a single cryo-EM map

<http://salilab.org/>

Dr. Benjamin Webb

Sali Lab

University of California San Francisco

Overview

Overview

- Objective: generate an atomic model of a 7-protein assembly

Overview

- Objective: generate an atomic model of a 7-protein assembly
- We don't know the atomic structures of each protein, or how they are assembled

Overview

- Objective: generate an atomic model of a 7-protein assembly
- We don't know the atomic structures of each protein, or how they are assembled
- But: we have the sequence of each protein, the structure of each protein in another species, and a low resolution (20Å) cryo-EM map of the assembly

Overview

- Objective: generate an atomic model of a 7-protein assembly
- We don't know the atomic structures of each protein, or how they are assembled
- But: we have the sequence of each protein, the structure of each protein in another species, and a low resolution (20Å) cryo-EM map of the assembly
- We can use this information with IMP to generate an atomic model

Overview

- Objective: generate an atomic model of a 7-protein assembly
- We don't know the atomic structures of each protein, or how they are assembled
- But: we have the sequence of each protein, the structure of each protein in another species, and a low resolution (20Å) cryo-EM map of the assembly
- We can use this information with IMP to generate an atomic model
- We will use an IMP application called MultiFit to achieve this

Overview

- Objective: generate an atomic model of a 7-protein assembly
- We don't know the atomic structures of each protein, or how they are assembled
- But: we have the sequence of each protein, the structure of each protein in another species, and a low resolution (20Å) cryo-EM map of the assembly
- We can use this information with IMP to generate an atomic model
- We will use an IMP application called MultiFit to achieve this
- “Watch me” style tutorial, since MultiFit does not (currently) work on Windows machines

Prerequisite: protein subunit structures

Prerequisite: protein subunit structures

- Since we know the structure of a related protein, and we know the sequence, we can use Modeller to build structures for each subunit

Prerequisite: protein subunit structures

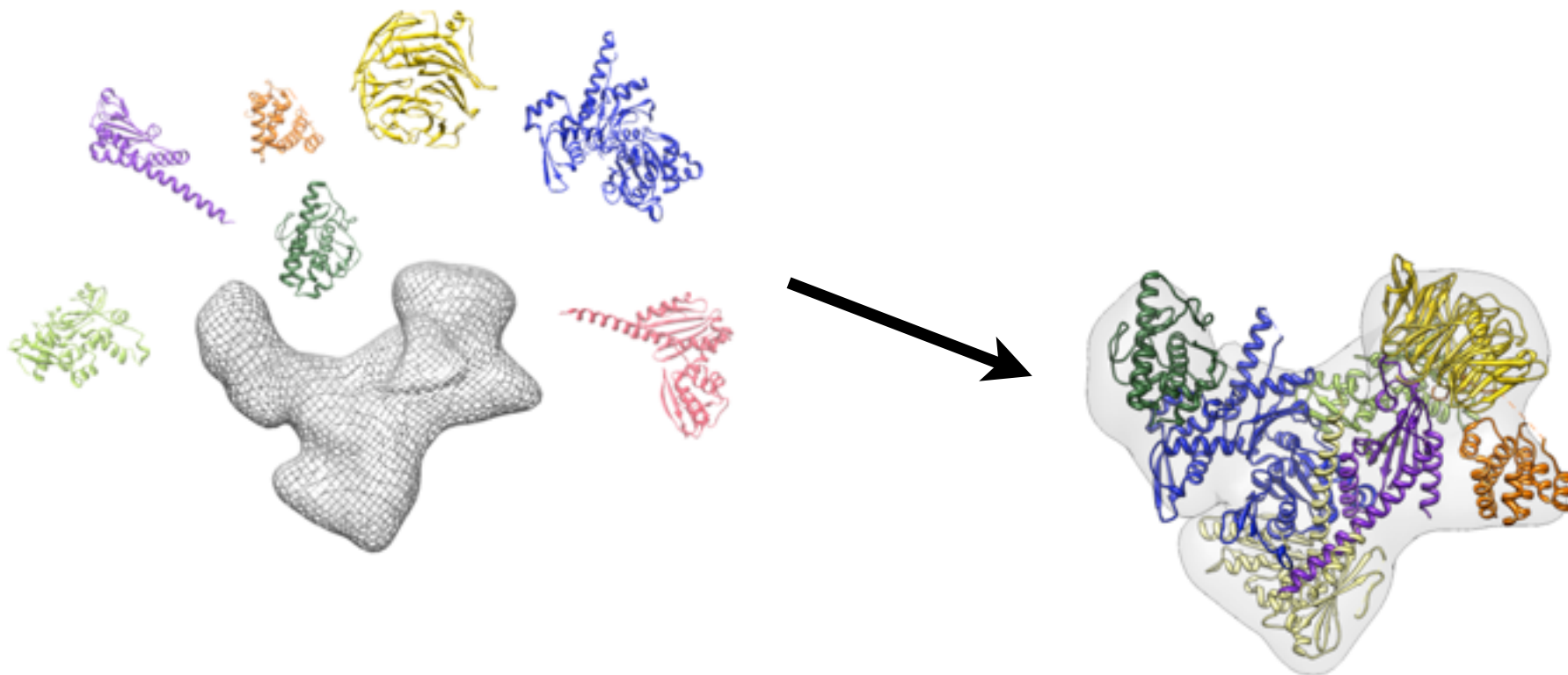
- Since we know the structure of a related protein, and we know the sequence, we can use Modeller to build structures for each subunit
- We won't cover this step here since it is fairly straightforward

Prerequisite: protein subunit structures

- Since we know the structure of a related protein, and we know the sequence, we can use Modeller to build structures for each subunit
- We won't cover this step here since it is fairly straightforward
- Note however that models should be reasonably high quality, since we want accurate protein-protein interfaces in order to combine subunits into the assembly

MultiFit protocol

Input: components (atomic models or crystal structures, or at low resolution), cryo-EM map

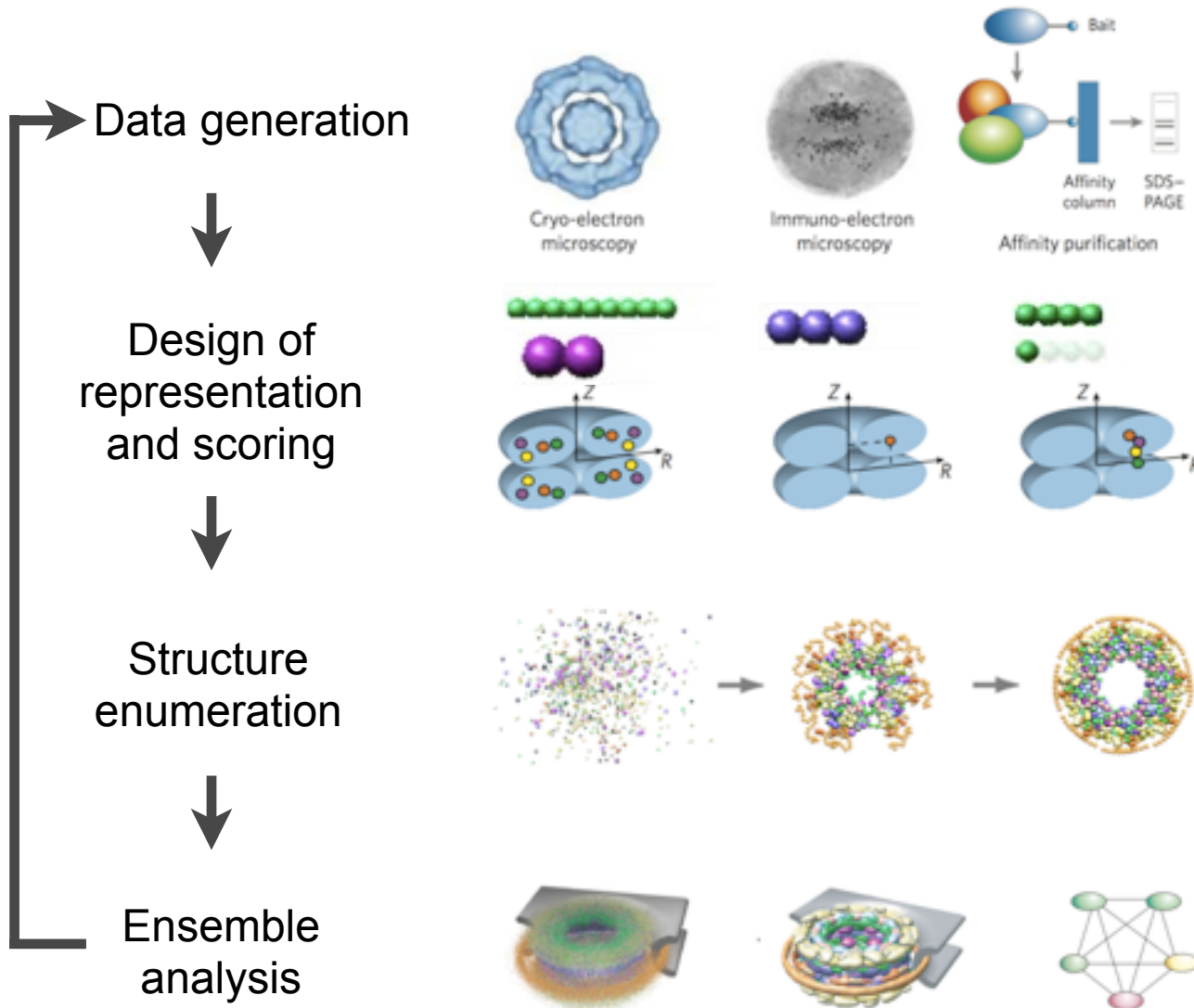


Keren Lasker, JMB **388** (2009)

<http://salilab.org/multifit/>

Output: component
configuration, to
be refined

MultiFit is an IMP application



Alber *et al.* *Nature* 2007 • Robinson, Sali, Baumeister. *Nature* 2007 •
Russel, et al. *Current Opinion in Cell Biology*, 2009

MultiFit is an IMP application

MultiFit is an IMP application

- Representation
 - Components represented as atoms; density map as a 3D voxel grid

MultiFit is an IMP application

- Representation
 - Components represented as atoms; density map as a 3D voxel grid
- Scoring
 - Quality-of-fit of individual components in the density map
 - Protrusion of each component from the map envelope
 - Shape complementarity between pairs of components

MultiFit is an IMP application

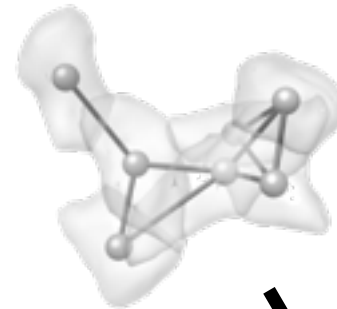
- Representation
 - Components represented as atoms; density map as a 3D voxel grid
- Scoring
 - Quality-of-fit of individual components in the density map
 - Protrusion of each component from the map envelope
 - Shape complementarity between pairs of components
- Structure enumeration
 - Determine approximate positions (anchor points)
 - Sampling of components into anchor points using a divide-and-conquer optimizer (DOMINO)

MultiFit protocol

1. Input: components, map

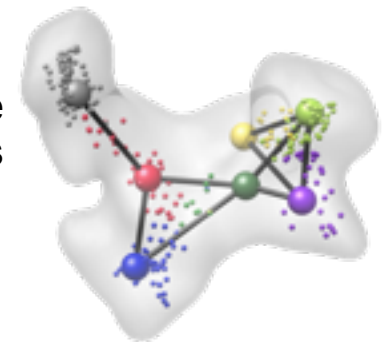


Map segmented into anchor graph

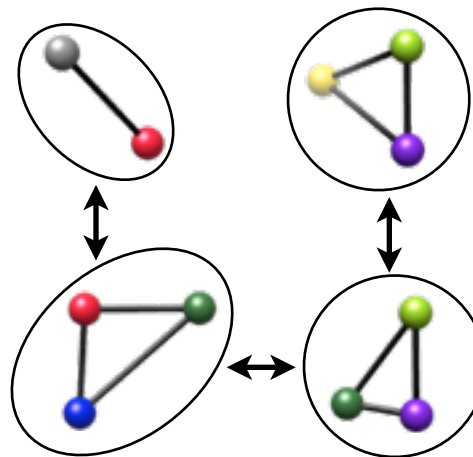


2. Discretize map and components

3. Fit proteins into the map

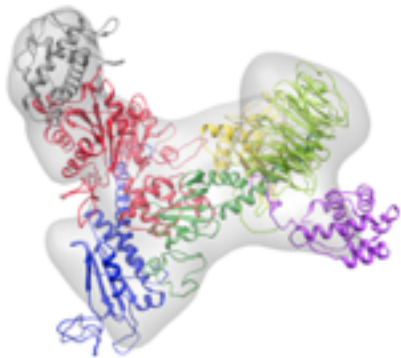


Decompose into subsets



4. Sample subsets semi-independently

Gather subset solutions into best global solutions



Component fits in vicinity of anchor nodes

Example

- Build a model of the ARP2/3 complex
- Consists of 7 subunits (ARP2, ARP3, ARC1-5)
- We have a cryo-EM map at 20Å resolution
- We can tell how well we did at the end, because there is actually a crystal structure of ARP2/3 (PDB code 1tyq)

Step 1: generate MultiFit input file

- File assembly.input contains data on all the subunits followed by assembly information
- Subunit portion:

```
ARP3|data/models/1tyq_A.pdb|results/1tyq_A_anchor_points.pdb|  
8|results/1tyq_A_fitting.output|data/models/1tyq_A.fitted.pdb|  
ARP2|data/models/1tyq_B.pdb|results/1tyq_B_anchor_points.pdb|  
5|results/1tyq_B_fitting.output|data/models/1tyq_B.fitted.pdb|  
...
```

Step 1: generate MultiFit input file

- File assembly.input contains data on all the subunits followed by assembly information
- Subunit portion:

```
ARP3|data/models/1tyq_A.pdb|results/1tyq_A_anchor_points.pdb|  
8|results/1tyq_A_fitting.output|data/models/1tyq_A.fitted.pdb|  
ARP2|data/models/1tyq_B.pdb|results/1tyq_B_anchor_points.pdb|  
5|results/1tyq_B_fitting.output|data/models/1tyq_B.fitted.pdb|  
...
```



A human-readable ID for the subunit

Step 1: generate MultiFit input file

- File assembly.input contains data on all the subunits followed by assembly information
- Subunit portion:

```
ARP3|data/models/1tyq_A.pdb|results/1tyq_A_anchor_points.pdb|  
8|results/1tyq_A_fitting.output|data/models/1tyq_A.fitted.pdb|  
ARP2|data/models/1tyq_B.pdb|results/1tyq_B_anchor_points.pdb|  
5|results/1tyq_B_fitting.output|data/models/1tyq_B.fitted.pdb|  
...
```

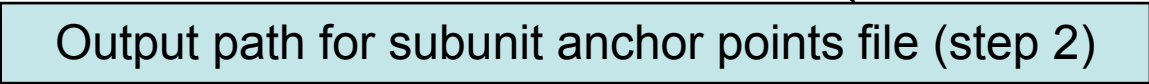


PDB file containing subunit structure

Step 1: generate MultiFit input file

- File assembly.input contains data on all the subunits followed by assembly information
- Subunit portion:

```
ARP3|data/models/1tyq_A.pdb|results/1tyq_A_anchor_points.pdb|  
8|results/1tyq_A_fitting.output|data/models/1tyq_A.fitted.pdb|  
ARP2|data/models/1tyq_B.pdb|results/1tyq_B_anchor_points.pdb|  
5|results/1tyq_B_fitting.output|data/models/1tyq_B.fitted.pdb|  
...
```

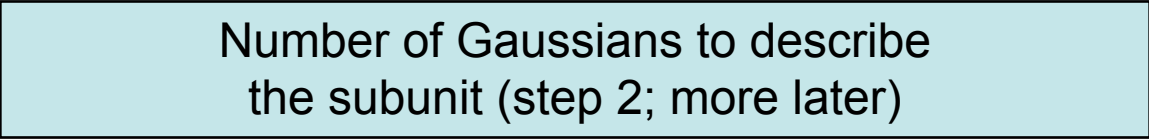


Output path for subunit anchor points file (step 2)

Step 1: generate MultiFit input file

- File assembly.input contains data on all the subunits followed by assembly information
- Subunit portion:

```
ARP3|data/models/1tyq_A.pdb|results/1tyq_A_anchor_points.pdb|  
8|results/1tyq_A_fitting.output|data/models/1tyq_A.fitted.pdb|  
ARP2|data/models/1tyq_B.pdb|results/1tyq_B_anchor_points.pdb|  
5|results/1tyq_B_fitting.output|data/models/1tyq_B.fitted.pdb|  
...
```

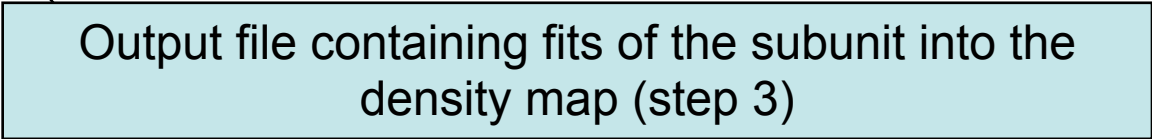


Number of Gaussians to describe
the subunit (step 2; more later)

Step 1: generate MultiFit input file

- File assembly.input contains data on all the subunits followed by assembly information
- Subunit portion:

```
ARP3|data/models/1tyq_A.pdb|results/1tyq_A_anchor_points.pdb|  
8|results/1tyq_A_fitting.output|data/models/1tyq_A.fitted.pdb|  
ARP2|data/models/1tyq_B.pdb|results/1tyq_B_anchor_points.pdb|  
5|results/1tyq_B_fitting.output|data/models/1tyq_B.fitted.pdb|  
...
```

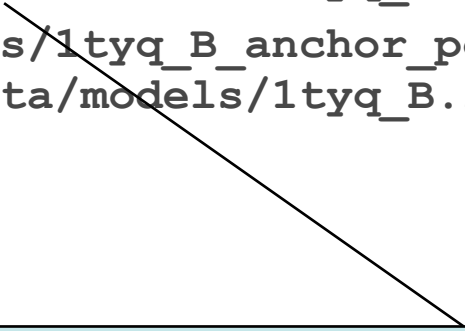


Output file containing fits of the subunit into the
density map (step 3)

Step 1: generate MultiFit input file

- File assembly.input contains data on all the subunits followed by assembly information
- Subunit portion:

```
ARP3|data/models/1tyq_A.pdb|results/1tyq_A_anchor_points.pdb|  
8|results/1tyq_A_fitting.output|data/models/1tyq_A.fitted.pdb|  
ARP2|data/models/1tyq_B.pdb|results/1tyq_B_anchor_points.pdb|  
5|results/1tyq_B_fitting.output|data/models/1tyq_B.fitted.pdb|  
...
```



Optional PDB file containing the real crystal structure of the subunit in the correct position in the map (for benchmarking the method)

Step 1: generate MultiFit input file

- File assembly.input contains data on all the subunits followed by assembly information
- Subunit portion:

```
ARP3|data/models/1tyq_A.pdb|results/1tyq_A_anchor_points.pdb|  
8|results/1tyq_A_fitting.output|data/models/1tyq_A.fitted.pdb|  
ARP2|data/models/1tyq_B.pdb|results/1tyq_B_anchor_points.pdb|  
5|results/1tyq_B_fitting.output|data/models/1tyq_B.fitted.pdb|  
...
```

Step 1: generate MultiFit input file

- Assembly portion:

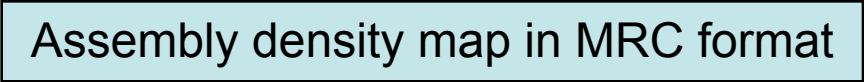
```
data/1tyq_20.mrc| 20.0| 3.| 0.| 0.| 0.|
```

```
results/1tyq_20.fine.gmm.pdb|results/1tyq_20.coarse.gmm.pdb|
```

Step 1: generate MultiFit input file

- Assembly portion:

```
data/1tyq_20.mrc| 20.0| 3.| 0.| 0.| 0.|  
results/1tyq_20.fine.gmm.pdb|results/1tyq_20.coarse.gmm.pdb|
```




Assembly density map in MRC format

Step 1: generate MultiFit input file

- Assembly portion:

```
data/1tyq_20.mrc| 20.0| 3.| 0.| 0.| 0.|  
results/1tyq_20.fine.gmm.pdb|results/1tyq_20.coarse.gmm.pdb|
```



Resolution of the density map (Å)

Step 1: generate MultiFit input file

- Assembly portion:

```
data/1tyq_20.mrc| 20.0| 3.| 0.| 0.| 0.|  
results/1tyq_20.fine.gmm.pdb|results/1tyq_20.coarse.gmm.pdb|
```



Spacing/voxel size, in Å/pixel

Step 1: generate MultiFit input file

- Assembly portion:

```
data/1tyq_20.mrc| 20.0| 3.| 0.| 0.| 0.|  
results/1tyq_20.fine.gmm.pdb|results/1tyq_20.coarse.gmm.pdb|
```




Origin of the density map

Step 1: generate MultiFit input file

- Assembly portion:

```
data/1tyq_20.mrc| 20.0| 3.| 0.| 0.| 0.|  
results/1tyq_20.fine.gmm.pdb|results/1tyq_20.coarse.gmm.pdb|
```



Output anchor points files (step 2)

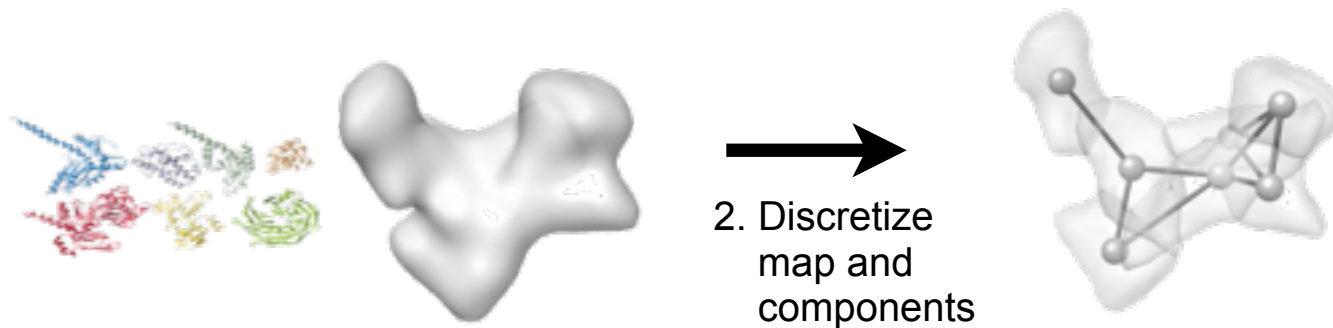
Step 1: generate MultiFit input file

- Assembly portion:

```
data/1tyq_20.mrc| 20.0| 3.| 0.| 0.| 0.|
```

```
results/1tyq_20.fine.gmm.pdb|results/1tyq_20.coarse.gmm.pdb|
```

Step 2: discretize map and subunits



```
/opt/multifit/utils/run_anchor_points_detection.py  
assembly.input 700
```

This runs two MultiFit utilities:

- gmm_em: determines a reduced representation of 3D Gaussians (Gaussian Mixture Model) of the EM map that best reproduces the configuration of all voxels with density above 700
- gmm_pdb: determines a set of N 3D Gaussians that best reproduces the atom configuration, for each subunit, where N is specified in assembly.input

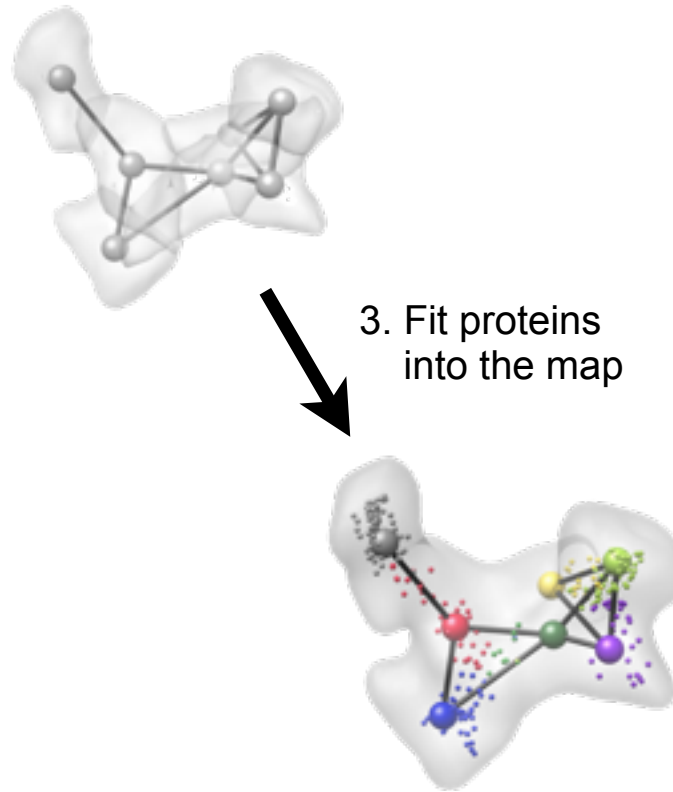
What's the best value of N?

- Some rules of thumb:
 - Require at least 3, as we need three points for the fitting.
 - Each Gaussian should cover the same amount of residues. If you choose, for example, 50 residues per Gaussian, a protein of 170 residues should have 3 Gaussians and one with 260 residues should have 5.
 - The number of Gaussians of the assembly should be equal to the sum of the Gaussians of all of the individual proteins.
 - Advanced: Plot the resulting likelihood function of the GMM clustering procedure for different values of N and look for a point of where this is a large drop.
- `anchor_point_estimator.py` can estimate the number of Gaussians for each protein

Outputs

- Set of 3D Gaussian centers
- Encoded in PDB files as fake C α atoms
- The density map's anchor graph segments the map into regions that correspond approximately to locations of the components in their assembly
 - At this stage, the assignment of components to regions is unknown

Step 3. fit proteins into the map



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

multifit.par sets various MultiFit optimization parameters;
see <http://salilab.org/multifit/> for more details

Outputs

- Set of candidate fits of individual subunits into the map
- Encoded as PDB files in the 'fits' subdirectory

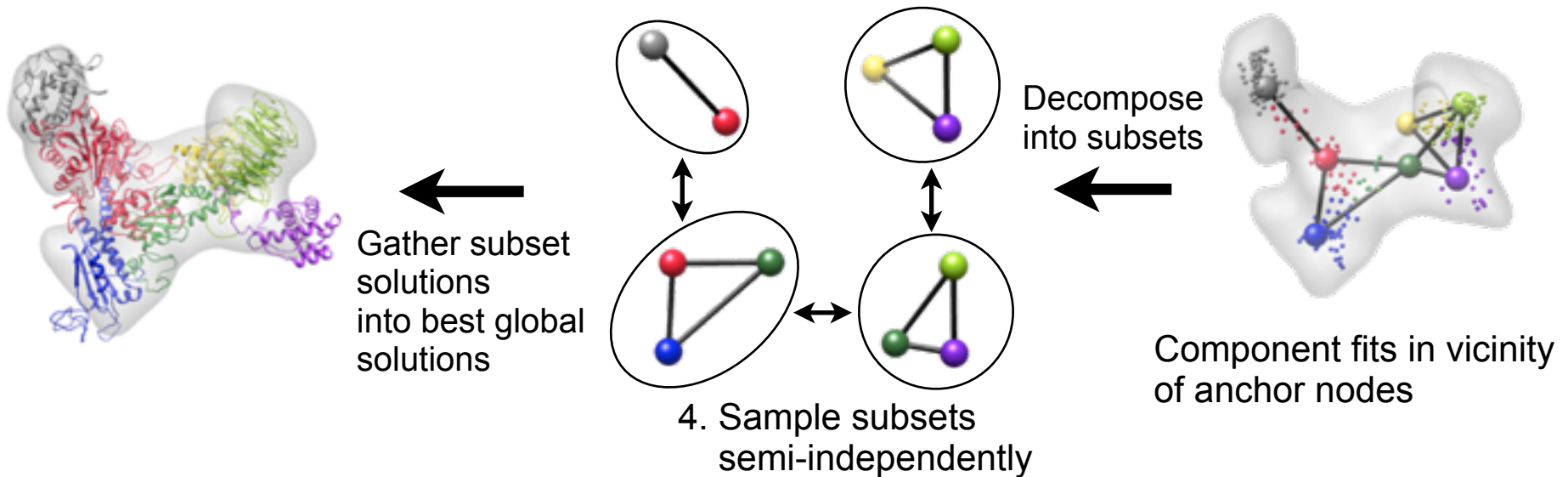
Step 3. fit proteins into the map

- Since we have native structures, we can see how well we did in this step:

```
/opt/multifit/utils/fetch_best_sampled_transformations.py  
assembly.input
```

```
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
```

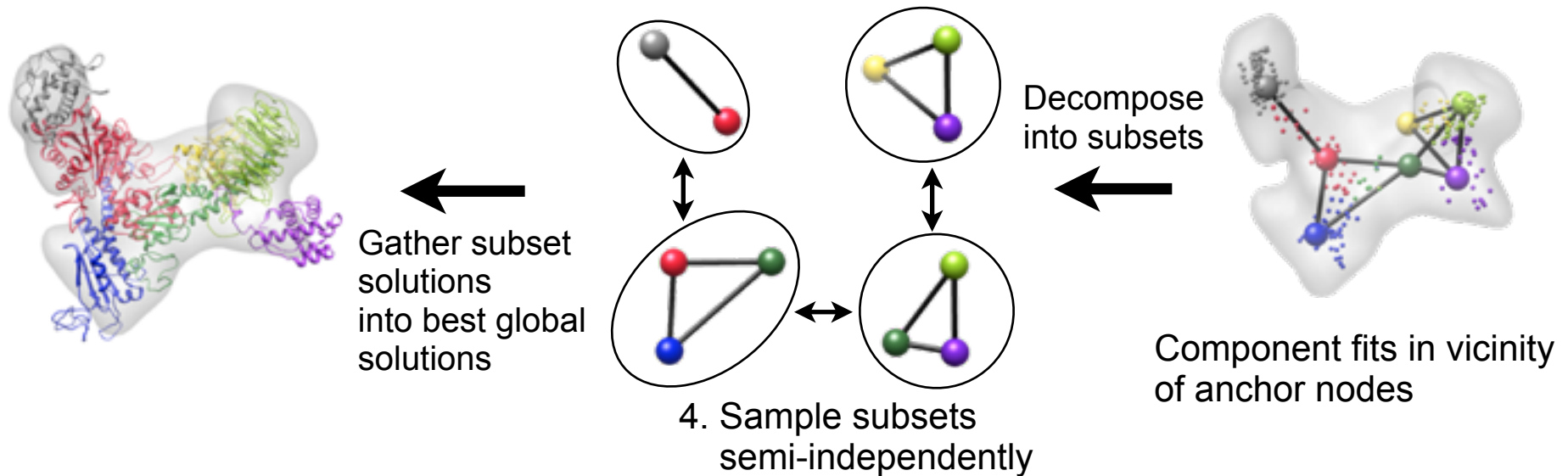
Step 4. Infer the optimum solution



First, we calculate all relevant scores:

```
/opt/multifit/utils/run_all_scores.py  
assembly.input > scores.log
```

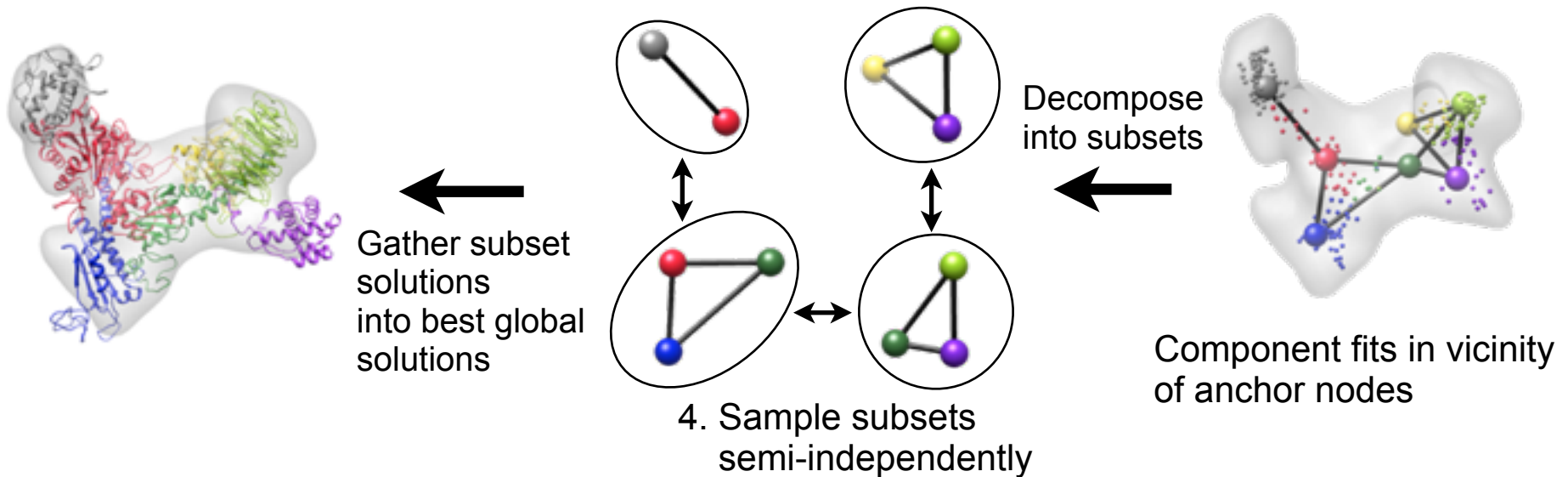
Step 4. Infer the optimum solution



Finally, we efficiently* enumerate all configurations (mappings of subunits to anchor points) to find the best scoring assembly models:

```
/opt/multifit/utils/run_multifit.py assembly.input  
assembly.jt assembly_configurations.output  
data/models/1tyq.fitted.pdb > multifit.log
```

Step 4. Infer the optimum solution



Finally, we efficiently* enumerate all configurations (mappings of subunits to anchor points) to find the best scoring assembly models:

```
/opt/multifit/utils/run_multifit.py assembly.input  
assembly.jt assembly_configurations.output  
data/models/1tyq.fitted.pdb > multifit.log
```

* Branch-and-bound optimization using the DOMINO divide-and-conquer message-parsing algorithm



DOMINO: Divide-and-Conquer

Represent the scoring function as a restraint graph.

Decompose the set of components into relatively decoupled subsets (a junction tree algorithm from graph theory).

Optimize each subset independently by a traditional optimizer, to get the optimal and a number of suboptimal solutions (restrained fitting for configuration stage and restrained docking for refinement stage).

Gather subset solutions into the best possible global solutions (message passing algorithms from graph theory; eg, belief-propagation).

1. Pairwise Graphical Model of Scoring Function

$$F(y_1, \dots, y_8) = \alpha_2(y_2) + \alpha_6(y_6) + \alpha_7(y_7) \\ + \beta_{12}(y_1, y_2) + \beta_{13}(y_1, y_3) + \beta_{14}(y_1, y_4) + \beta_{15}(y_1, y_5) \\ + \beta_{23}(y_2, y_3) + \beta_{24}(y_2, y_4) + \beta_{36}(y_3, y_6) + \beta_{38}(y_3, y_8) \\ + \beta_{45}(y_4, y_5) + \beta_{56}(y_5, y_6) + \beta_{58}(y_5, y_8)$$

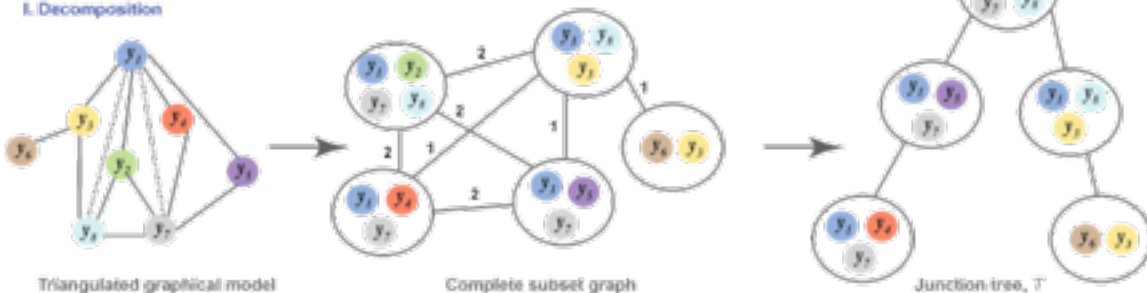
Scoring function, S



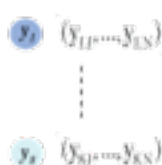
Graphical model, G

2. Divide-and-Conquer Approach

I. Decomposition



II. Variable sampling



III. Gathering



Final output

- `assembly_configurations.output:`

`ARP3,0|ARP2,14|ARC1,3|ARC2,24|ARC3,19|ARC4,11|ARC5,13|
(17.5593729019) (rmsd:29.2637996674) (conf.0.pdb)`

`ARP3,5|ARP2,13|ARC1,9|ARC2,24|ARC3,19|ARC4,4|ARC5,13|
(18.3258602619) (rmsd:11.997220993) (conf.1.pdb)`

`...`

Final output

- `assembly_configurations.output:`

`ARP3,0|ARP2,14|ARC1,3|ARC2,24|ARC3,19|ARC4,11|ARC5,13|`
`(17.5593729019) (rmsd:29.2637996674) (conf.0.pdb)`

`ARP3,5|ARP2,13|ARC1,9|ARC2,24|ARC3,19|ARC4,4|ARC5,13|`
`(18.3258602619) (rmsd:11.997220993) (conf.1.pdb)`

...

Chosen fits for each subunit

Final output

- `assembly_configurations.output:`

ARP3,0|ARP2,14|ARC1,3|ARC2,24|ARC3,19|ARC4,11|ARC5,13|
(17.5593729019) (rmsd:29.2637996674) (conf.0.pdb)

ARP3,5|ARP2,13|ARC1,9|ARC2,24|ARC3,19|ARC4,4|ARC5,13|
(18.3258602619) (rmsd:11.997220993) (conf.1.pdb)

...



Score of the assembly

Final output

- `assembly_configurations.output:`

`ARP3,0|ARP2,14|ARC1,3|ARC2,24|ARC3,19|ARC4,11|ARC5,13|`
`(17.5593729019) (rmsd:29.2637996674) (conf.0.pdb)`

`ARP3,5|ARP2,13|ARC1,9|ARC2,24|ARC3,19|ARC4,4|ARC5,13|`
`(18.3258602619) (rmsd:11.997220993) (conf.1.pdb)`

...



Fit against known crystal structure

Final output

- `assembly_configurations.output:`

`ARP3,0|ARP2,14|ARC1,3|ARC2,24|ARC3,19|ARC4,11|ARC5,13|
(17.5593729019) (rmsd:29.2637996674) (conf.0.pdb)`

`ARP3,5|ARP2,13|ARC1,9|ARC2,24|ARC3,19|ARC4,4|ARC5,13|
(18.3258602619) (rmsd:11.997220993) (conf.1.pdb)`

`...`



File name of output assembly

Final output

- `assembly_configurations.output:`

`ARP3,0|ARP2,14|ARC1,3|ARC2,24|ARC3,19|ARC4,11|ARC5,13|
(17.5593729019) (rmsd:29.2637996674) (conf.0.pdb)`

`ARP3,5|ARP2,13|ARC1,9|ARC2,24|ARC3,19|ARC4,4|ARC5,13|
(18.3258602619) (rmsd:11.997220993) (conf.1.pdb)`

`...`

Reranking by proteomics data

- For challenging molecules, the MultiFit score is not sufficient
- If we have proteomics information, we can use IMP to combine this with the EM fit to rescore the final solutions
- This can be used to distinguish a “good” model from a “bad” one when they are ranked similarly
- Here, we use Restrainer and Y2H data:



```
cd proteomics
```

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

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

Summary

Summary

- MultiFit fits multiple protein subunits into a density map of the complete assembly

Summary

- MultiFit fits multiple protein subunits into a density map of the complete assembly
- In our example, subunits are all fitted into the density and the architecture is probably correct

Summary

- MultiFit fits multiple protein subunits into a density map of the complete assembly
- In our example, subunits are all fitted into the density and the architecture is probably correct
- However, protein-protein interactions are probably not correct

Summary

- MultiFit fits multiple protein subunits into a density map of the complete assembly
- In our example, subunits are all fitted into the density and the architecture is probably correct
- However, protein-protein interactions are probably not correct
- Can further refine using a pairwise docking algorithm (e.g. PatchDock) combined with additional local sampling