ChimeraX MorphOT User Manual

Arthur Ecoffet, Frédéric Poitevin, Khanh Dao Duc

August 2020

Contents

1 Introduction

1.1 What is MorphOT ?

MorphOT is a plugin for *ChimeraX*, which allows users to directly interpolate multiple density maps. It relies on adapting and optimizing for EM maps some recent methods of shape interpolation based on optimal transport (OT), resulting in significant improvement in quality of morphing, compared with the standard command provided with *ChimeraX*. For more details on the method, please see the appendix and the paper accompanying the software [1].

1.2 Development and requirements

MorphOT has been developed using Python 3.7 and is implemented as a plugin for UCSF ChimeraX. A GPU implementation of all *MorphOT* functions is also provided, which requires NVidia GPUs and the Cuda Toolkit to be run.

1.3 Downloading and running ChimeraX-morphOT

To download and install *MorphOT*, first download the source code folder *otmorph-bundle* available at the following Github link.
To install, start ChimeraX and type the following command line:

```
devel build Path/To/Source/Code/otmorph-bundle
```

and

```
devel install Path/To/Source/Code/otmorph-bundle
```

Remark: After review by the UCSF ChimeraX team, the tool will be available in the ChimeraX **Tool** menu (**More tools**), or by searching for *MorphOT* here.

2 General Features

2.1 Input Type

MorphOT takes two ChimeraX density maps as inputs. For the morphing method to perform well, we recommend ensuring that the maps are smooth enough and doing some preprocessing if necessary. Our **Tutorial** (section 3) provides some more details on how to do it.

2.2 Functions

The plugin *MorphOT* is composed of four main functions, which are detailed below.

1. **morphOT**

   MorphOT morphOT *volume-spec* [**start** *start-fraction*] [**playStep** *increment*] [**frames** *N*] [**playDirection 1** | -1] [**playRange** *low-fraction, high-fraction*] [**rate 'linear'** | 'sinusoidal', 'ramp up', 'ramp down'] [**maxsize** *max*][**constantVolume** true | **false**] [**hideOriginalMaps true** | false] [**interpolateColors true** | false] [**niter** *K*] [**reg** *r*] *new-map-options*

   Similar to the "volume morph" function in ChimeraX, **morphOT** creates a trajectory morphed between two maps.For a reasonable result, the inputs maps must have the same grid dimensions. Note volume resample can be used to make a copy of one map that has the same grid as another. A morphing fraction of 0.0 corresponds to the first map and a fraction of 1.0 corresponds to the last, with intermediate maps evenly spaced within that range. The user can also input more than 2 maps $V_1, \dots V_n$, where $n > 2$. In this case, **morphOT** will interpolate between 1 and 2, 2 and 3 etc.

   The morph display will proceed from *start-fraction* (default **0.0**) in steps of *increment* (default **0.04**) for *N* **frames** (default **25**). If the number of frames and step increment exceeds what is needed to reach the **playRange** bounds (default is the entire range : **0.0,1.0**), the morph display will "bounce" back and forth. The **rate** option (default 'linear') has the coefficients change at linear, sinusoidal, ramp up (slow at the beginning, fast at the end) or ramp down rate. The **maxsize** option (default 60) sets the maximum grid size ($maxsize^3$): If the current size of the maps exceed this value, the maps are resized to $maxsize$. The **constantVolume** option allows automatically adjusting the threshold (contour level) to keep the enclosed volume constant. The

hideOriginalMaps option specifies hiding the input maps. The **interpolateColors** option only applies when the maps have the same number of threshold (contour levels for surface/mesh display). The **niter** option (default **20**) specifies the number of iterations in the loop computing each morph frames; the greater this parameter the more accurate each frame is, but with computation cost that scales linearly with this term. In practice, **20** is generally enough for convergence [2]. The **reg** option (default **max_grid_dimension/60**) defines the entropy parameter in the method we used (see Appendix or [2]): convergence of each frame is faster when this parameter is larger, but it also implies more blurred morphing. On the other hand, numerical errors tend to occur if the parameter gets too small. See here for details about *new map options*.

The morphing trajectory is created in a new map (volume) model. However, if the modelId of an existing morph map is given, the existing morph will be used instead of a new one being calculated.

2. **semiMorphOT**

   MorphOT semiMorphOT *volume-spec* [**ot_frames** N_{OT}] [**frames** N] *morph-ot-options*

   Instead of computing each frame as in the *morphOT* function, **semimorphOT** computes N_{OT} Wasserstein Barycenters (see Appendix), between which it linearly interpolates. This function allows for finding a good compromise between computation time and accuracy, notably for bigger structures.

   The option **ot_frames** (default 4) determines the number of optimal transport barycenters between which linear interpolation will be displayed. The option **frames** (default 25) is the *total* number of frames displayed.

3. **oneBarycenter**

   MorphOT oneBarycenter *volume-spec weights , w1,w2 ...* [**niter** K] [**reg** r] [**maxsize** *max*] [**interpolateColors true** | false] *new-map-options*

   oneBarycenter computes and displays one weighted barycenter of *two or more* volumes given the **weights** (as many as the number of volumes). The other options are the same as in *morphOT* and *semiMorphOT*.

4. **BarycenterSave**

   MorphOT BarycenterSave *volume-spec folder_path* [**frames** *N*] [**name1 n1**] [**name2 n2**] [**niter** *K*] [**reg** *r*] [**rate 'linear'** | 'sinusoidal', 'ramp up', 'ramp down'] *new-map-options*

   BarycenterSave computes and locally saves the transition pathway displayed by *morphOT*. Options **name1** and **name2** allow the user to call the volume names which are used to save each barycenter. The file name has the form:

   <frame_number>_<name1>_<name2>_weights_<current_frame_weights>.mrc

   Other options are the same as in the aforementioned functions.

3 Tutorial

3.1 Preprocessing the density maps

1. Transport-based morphing requires "smooth" structures with low noise level. We thus begin with a tutorial describing structures pre-processing.

   In this example, we use EMDB Structure 5140. As one can see in Figure 1, a lot of noise is present with low density values

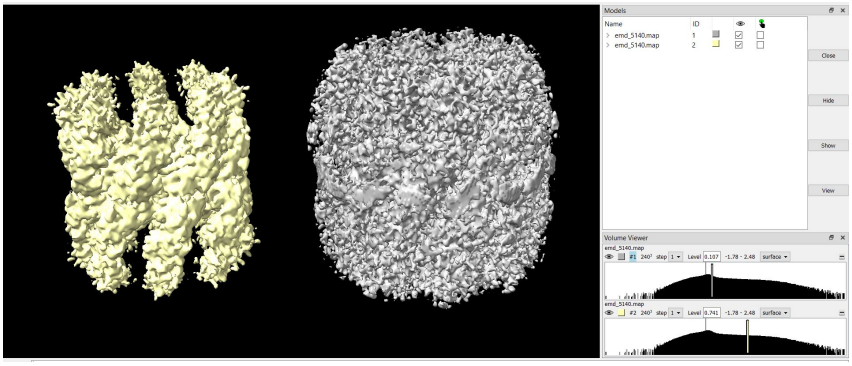

Figure 1: Left: EMDB 5140 with adjusted threshold. We see the shape of the molecule. Right: Same structure with lower threshold, displaying all the noise surrounding the shape

   To smooth the structure, we apply a gaussian blur by typing (see Figure 2):

```
volume gaussian #1 sd 2
```

   (**Remark**: 1 designates volume n1 in ChimeraX volume viewer)

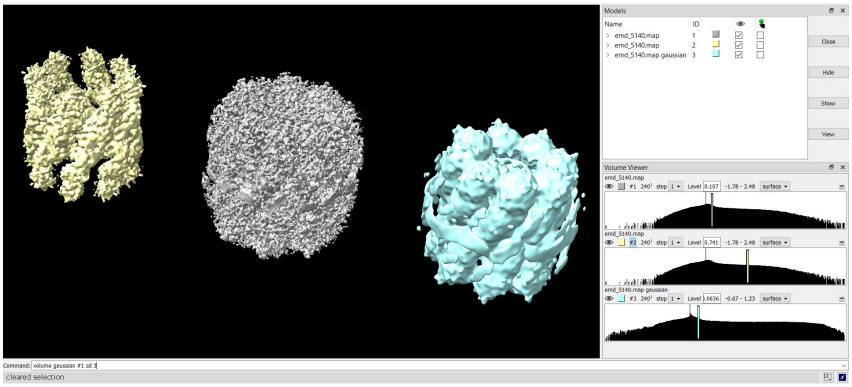

Figure 2: In blue, smoothed version of the yellow map. There still is some noise, appearing as a ring surrounding the density map.

2. Once the noise has been reduced by smoothing, we can also threshold the structure density to remove the remaining noise and only see the structure shape. After finding the critical threshold value where the noise appears (see Figure 3), we threshold the

density maps at this value, and then shift the values so that the minimum is equal to 0. To do this, type in the command line (quoted words designate where the user has to put their own values):

```
volume threshold #'volume_number' min 'threshold_value'
```

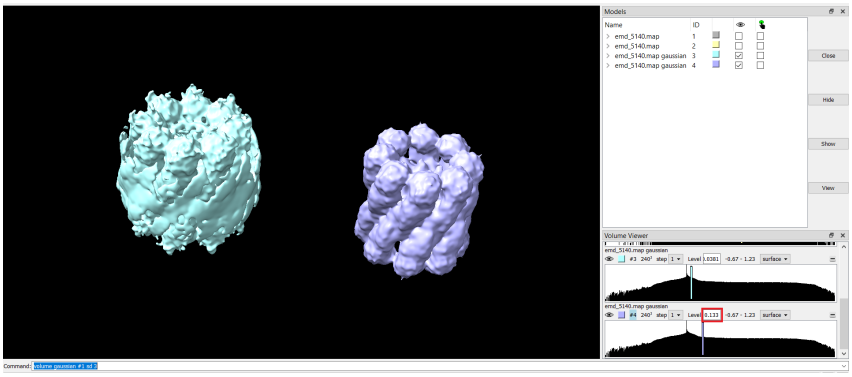

Figure 3: Two smoothed versions of EMDB 5140 with different thresholds. Left: Noise does appear with low thresholding value (0.038). Right: Threshold (see red square on the bottom right) is chosen to be just above the level (0.133) where noise starts appearing.

Finally, to enhance the quality of the interpolation, it is preferable that input volumes have density values comprised between 0 and 1. To re-scale the density, type:

```
volume scale #'volume_number' shift -"threshold_value"
```

Our first structure has now been pre-processed and is ready. To complete the pre-processing, repeat the same steps to structure # 2

3. In the case where structures are not on the same grid, one can fix this by using **volume resample**, and typing (assuming first and second structures have IDs # 1 and # 2)

   ```
   volume resample #2 onGrid #1
   ```

4. Finally, it is to be noted that the threshold values have to be set so that the inputs are *dense*, it is possible that otherwise the displayed result won't be apparent.

As all structures are now pre-processed and have the same size, we can start using *MorphOT*.

3.2 Displaying movie with *morphOT*

To produce a standard transport-based trajectory, type:

```
MorphOT morphOT #1 #2
```

Adding options **reg** and **frames** will have the most effect on how the movie looks like. Alternately, for faster computations and sometimes smoother movie, one can rather use:

```
MorphOT semimorphOT #1 #2 otframes 5 totalframes 25
```

By modifying the **otframes** option, one will change the number of transport-based computed frames, between which linear interpolation is done.

In the case of *morphOT* and *semimorphOT* one can export a movie by typing the following few commands :

```
movie record
MorphOT morphOT [morphOT options]
movie stop
movie encode [path]
```

4 Appendix

4.1 Running on GPU

GPU computing is only available in USCF ChimeraX for GPUs with CUDA cores (most NVIDIA GPUS). To have MorphOT running with GPU's, follow these extra steps:

1. If you do not already have it, install Cuda toolkit (link here)
2. Note your Cuda toolkit number (10.0,10.1 ...) and open file **bundle_info.xml** in the **otmorph-bundle** directory
3. Uncomment line <Dependency name="cupy-cuda102" version=">=0.1"/ >¿ and replace 'cuda102' by your cuda version.
4. Open ChimeraX and do "devel build ..." + "devel install" as explained in section 1.3 ("Downloading and Installing OTMorph").

4.2 Details on the algorithm

Before stating the algorithm implemented in *MorphOT*, we need to introduce the mathematical framework and tools underlying it.

Definition of the interpolant: First, to produce trajectories between two EM density maps V_0 and V_1, we use the following interpolant

$$V_t = \underset{V}{\operatorname{argmin}} \left[(1-t)\mathcal{W}_2^2(V_0, V) + t\mathcal{W}_2^2(V, V_1)\right], \tag{1}$$

where $\mathcal{W}_2^2$ is defined as

$$\mathcal{W}_2^2(\mu_0, \mu_1) \overset{\text{def.}}{=} \inf_{\pi \in \Pi(\mu_0, \mu_1)} \int_{X \times Y} d^2(x, y)\ \mathrm{d}\pi(x, y), \tag{2}$$

Equation 1 simply defines a *weighted barycenter* problem between maps V_0 and V_1 with respect to $\mathcal{W}_2^2$.

Computation of $\mathcal{W}_2^2$: $\mathcal{W}_2^2$ is not a trivial metric, and finding efficient ways of computing it is an active topic of research. Here, we followed the recent method introduced by Solomon *et al.* [2], summarized as follows: First, $\mathcal{W}_2^2$ is regularized with the entropy $H(\pi)$ of the *transportation plan* π (as inspired by Cuturi [3]), defined as

$$H(\pi) \stackrel{\text{def.}}{=} -\iint_{X\times Y} \pi(x,y)\ln\pi(x,y)\,\mathrm{d}x\,\mathrm{d}y. \tag{3}$$

This term yields the *entropy-regularized 2-Wasserstein distance*

$$\mathcal{W}_{2,\gamma}^2(\mu_0,\mu_1) \stackrel{\text{def.}}{=} \inf_{\pi\in\Pi(\mu_0,\mu_1)} \left[\int_{X\times Y} d^2(x,y)\ \mathrm{d}\pi(x,y) - \gamma H(\pi)\right], \tag{4}$$

where $\gamma > 0$ is the *entropy parameter* (as γ increases, more spread-out solutions are promoted [2]). This regularization simplifies the original problem by making it *strictly convex*, ensuring the existence of a unique solution.

In practice, solving this optimization problem over a large domain by computing and storing the matrix of pairwise distance $d^2(x,y)$ is computationally expensive. To overcome this issue Solomon *et al.* have proposed in addition to approximate $d^2(x,y)$ using Varadhan's formula [4]: Considering the heat transfer from x to y over a short period of time,

$$d(x,y)^2 = \lim_{t\to 0}[-2t\ln\mathcal{H}_t(x,y)],$$

where $\mathcal{H}_t(x,y) = e^{-2d^2(x,y)/t}$ is the *Heat Kernel*. For $\gamma \ll 1$ and setting $t \stackrel{\text{def.}}{=} \gamma/2$, we obtain

$$d^2(x,y) \approx -\gamma\ln\left(e^{-d^2(x,y)/\gamma}\right).$$

Combining this approximation with (4), we can build another approximation of $\mathcal{W}_2^2$ in terms of a *projection problem* with respect to the Kullback-Leibler divergence, another common metric on densities. Finally, the metric we compute as an approximation of $\mathcal{W}_2^2$ is

$$\mathcal{W}_{2,\mathcal{H}_{\gamma/2}}^2(\mu_0,\mu_1) \stackrel{\text{def.}}{=} \gamma\left[1 + \min_{\pi\in\Pi(\mu_0,\mu_1)} \mathrm{KL}(\pi\,|\,\mathcal{H}_{\gamma/2})\right]. \tag{5}$$

After natural discretization of the problem (densities encoded as vectors, joint distributions and operators as matrices), and with a basic use of *Lagrangian optimization*. We obtain a solution π in closed form which can be efficiently computed using Sinkhorn-Knopp matrix scaling algorithm. [5]. We refer to [2] and its supplemental material for more details.

Computation of V_t: As Equation 5 provides a good approximation of $\mathcal{W}_2^2$, we can re-evaluate V_t as

$$V_t = \operatorname*{argmin}_V \left[(1-t)\mathcal{W}_{2,\mathcal{H}_{\gamma/2}}^2(V_0,V) + t\mathcal{W}_{2,\mathcal{H}_{\gamma/2}}^2(V,V_1)\right] \tag{6}$$

$$= \operatorname*{argmin}_V \left[(1-t)\min_{\pi\in\Pi(V_0,V)} \mathrm{KL}(\pi\,|\,\mathcal{H}_{\gamma/2}) + t\min_{\pi\in\Pi(V,V_1)} \mathrm{KL}(\pi\,|\,\mathcal{H}_{\gamma/2})(V,V_1)\right]. \tag{7}$$

After discretization, equation 7 reformulates as a *projection problem* with respect to the KL-divergence on an *intersection* of convex sets. It is a harder problem than computing $\mathcal{W}_2^2$,

but it can be solved using *iterated Bregman projections*, which state that we can project on an intersection of set by iteratively projecting on one set then on the other [6].

This finally yields the following algorithm, returning the barycenter $\boldsymbol{\mu}$, for any number of input maps $\{\boldsymbol{\mu_i}\}_i$ and given a set of weights $\{\alpha_i\}_i$:

Algorithm 1 Iterated Bregman Projection for Wasserstein barycenters

Require: $\{\boldsymbol{\mu_i}\}$, $\{\alpha_i\}$, $\mathbf{H_t}$

$\mathbf{v_1}, \cdots, \mathbf{v_k} \leftarrow \mathbf{1}$
$\mathbf{w_1}, \cdots, \mathbf{w_k} \leftarrow \mathbf{1}$
for j = 1,2,3, $\cdots$ **do**
  $\boldsymbol{\mu} \leftarrow \mathbf{1}$
  for i = 1, $\cdots$, k **do**
    $\mathbf{w_i} \leftarrow \boldsymbol{\mu_i} \oslash \mathbf{H_t}(\mathbf{v_i})$
    $\mathbf{d_i} \leftarrow \boldsymbol{v_i} \otimes \mathbf{H_t}(\mathbf{w_i})$
    $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} \otimes \mathbf{d_i}$
  end for
  for i = 1, $\cdots$, k **do**
    $\mathbf{v_i} \leftarrow \mathbf{v_i} \otimes \boldsymbol{\mu} \oslash \mathbf{d_i}$
  end for
end for
return $\boldsymbol{\mu}$

References

[1] Ecoffet A, Poitevin F, Dao Duc K. MorphOT: Transport-based interpolation between EM maps with UCSF ChimeraX. *(submitted)* (2020).

[2] Solomon J, De Goes F, Peyré G, Cuturi M, Butscher A, Nguyen A, et al. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (TOG)* **34**, 1–11 (2015).

[3] Cuturi M. Sinkhorn distances: Lightspeed computation of optimal transport. In: Advances in neural information processing systems. (2013).. p. 2292–2300.

[4] Varadhan SRS. On the behavior of the fundamental solution of the heat equation with variable coefficients. *Communications on Pure and Applied Mathematics* **20**, 431–455 (1967).

[5] Sinkhorn R, Knopp P. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific J Math* **21**, 343–348 (1967).

[6] Bregman LM. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics* **7**, 200 – 217 (1967).