# ChimeraX MorphOT User Manual

arthurecoffet

July 2020

## Contents

## 1 Introduction

### 1.1 What is MorphOT ?

The main purpose of our bioinformatic tool, *MorphOT*, is to adapt and optimize the OT-based interpolation of shapes for EM maps, as a plugin for *ChimeraX*. *MorphOT* allows users to directly interpolate multiple density maps, with a significant improvement in quality, compared with the standard interpolation command provided with *ChimeraX*. See appendix for more details.

### 1.2 Development and requirements

*MorphOT* has been developed using Python 3.7 for implementation as a plugin into UCSF ChimeraX. A GPU implementation of all *MorphOT* function is provided, the use of GPUs requires NVidia GPUs and the Cuda Toolkit.

### 1.3 Downloading and running ChimeraX-morphOT

While the plugin hasn't been reviewed by UCSF ChimeraX team, one can download the source code folder *ot-morph* on the following github.
Then type into the chimeraX command line :

```
devel build Path/To/Source/Code/ot-morph
```

and

```
devel install Path/To/Source/Code/ot-morph
```

After UCSF ChimeraX team reviewal, the tool will be available either by clicking **More tools** in the ChimeraX **Tool** menu, or on the internet here by searching *MorphOT*.

## 2 General Features

### 2.1 Input Type

*MorphOT* takes as input two ChimeraX density maps. Since the main morph method requires smooth inputs, it is likely that the density maps require preprocessing, see **Tutorial** for more details on how to preprocess data.

The plugin *MorphOT* is composed of four main function, which documentation is detailed below.

### 2.2 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*

  *morphOT* has the same description as previous ChimeraX "volume morph" : Morph between two or more maps. For a reasonable result, the input maps should have the same grids: dimensions, spacing, and numbers of points. 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. There is smooth interpolation between each adjacent pair of maps.

  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 exceed 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$^3$) over which structures are resized to maxsize for efficiency issues. The **constantVolume** option specifies adjusting the threshold (contour level) automatically 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 specifies (default **20**) specifies the number of iteration in the loop computing each morph frames, the greater this parameter the more accurate each frames are, yet computation scales linearly with this term and **20** is generally enough for convergence [1]. The **reg** option (default **max_grid_dimension/60**) defines the entropy parameter in [1], convergence of each frames is faster when this parameter gets bigger yet it also implies a more blurred morphing, on the other hand there tends to be numerical errors if the parameter gets too small. See here for details about *new map options*

The morph 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.

## 2.3 semiMorphOT

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

  An approximation of *morphOT*, instead of computing each frame using Wasserstein Barycenters (OT theory), *semimorphOT* computes $N_{OT}$ Wasserstein Barycenter between which it linearly interpolates. Althoug linear interpolations produces results which don't have any physical interpretation and which can be not satisfying on *big, non linear* transition pathways, the results are satisfying when linearly interpolating between close shapes computed using Wasserstein Barycenters. *semiMorphOT* allows to compute satisfying transition pathways with a good compromise on computation time for bigger structures, thus sometimes allowing to produce prompt transition movies.

  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.

## 2.4 oneBarycenter

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

  *oneBarycenter* computes and display one weighted barycenter of *two or more* volumes given the **weights** (as many as volumes, summing to one). The other options behave as in *morphOT* and *semiMorphOT*.

## 2.5 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 save locally the transition pathway displayed by *morphOT*. Options **name1** and **name2** allow the user to chose the volume names used for saving each barycenter. The file name has form :

  %frame_number%_%name1%_%name2%_weights_%current_frame_weights%.mrc

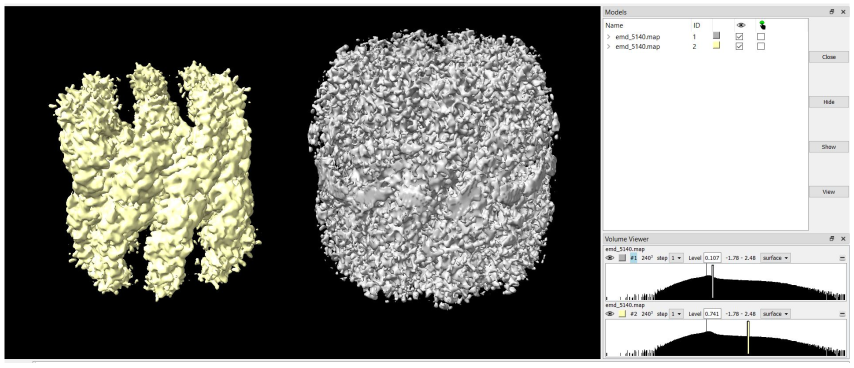
  Other options behave the same as in the previous functions.

# 3 Tutorial

## 3.1 Preprocessing the density maps

1. First, optimal transport morphing requires ”smooth” structures with little noise. Let’s begin with a preprocessing structures tutorial.

   We use on this example EMDB Structure 5140. As one can see on the screenshot below, a lot of noise is present with low density values



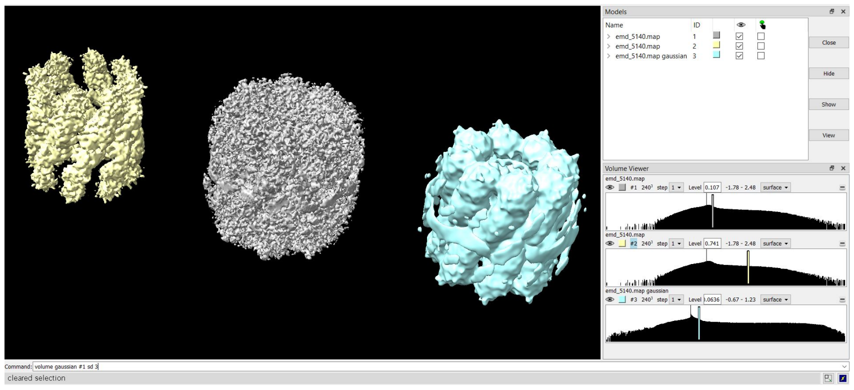
| Name | ID |
|---|---|
| emd_5140.map | 1 |
| emd_5140.map | 2 |

```
emd_5140.map
#1 240³ step 1 Level 0.107 -1.78 - 2.48 surface
emd_5140.map
#2 240³ step 1 Level 0.741 -1.78 - 2.48 surface
```

Figure 1: Left : EMDB 5140 with adjusted threshold, we see the shape of the molecule
Right : EMDB 5140 with lower threshold to display all the noise surrounding the shape

To smooth the structure we apply a gaussian blur by typing :

```
        volume gaussian #1 sd 2 \{Comment : #1 designate volume n1 in ChimeraX v
```



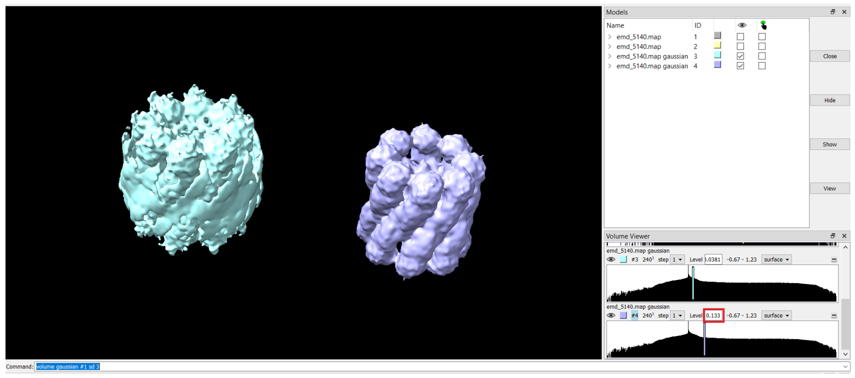
| Name | ID |
|---|---|
| emd_5140.map | 1 |
| emd_5140.map | 2 |
| emd_5140.map gaussian | 3 |

```
emd_5140.map
#1 240³ step 1 Level 0.107 -1.78 - 2.48 surface
emd_5140.map
#2 240³ step 1 Level 0.741 -1.78 - 2.48 surface
emd_5140.map gaussian
#3 240³ step 1 Level ).0636 -0.67 - 1.23 surface
Command: volume gaussian #1 sd 3
cleared selection
```

Figure 2: On the right, smoothed version of the previous map, there still is some noise as seen with the ring surrounding the density maps. We will threshold the map to remove it

2. Once the noise has been reduced by smoothing, we can also threshold the structure density to only get the structure shape.

   In the following we will threshold the density maps at this value and then shift the values so that the minimum is equal to 0. We can do this by typing the following in



| Name | ID | | 👁 | |
|---|---|---|---|---|
| emd_5140.map | 1 | ■ | ☐ | ☐ |
| emd_5140.map | 2 | ■ | ☐ | ☐ |
| emd_5140.map gaussian | 3 | ■ | ☑ | ☐ |
| emd_5140.map gaussian | 4 | ■ | ☑ | ☐ |

```
emd_5140.map gaussian
#3 240³ step 1 Level ).0381 -0.67 - 1.23 surface
emd_5140.map gaussian
#4 240³ step 1 Level 0.133 -0.67 - 1.23 surface
Command: volume gaussian #1 sd 3
```

Figure 3: Two smoothed version of EMDB 5140 with different displaying threshold
Left : A lot of noise appears with this thresholding value
Right : Threshold is chose to be just above where noise begins to appear
Defining threshold is here equal to 0.133 (see red square on the bottom right)

the command line : (quoted words designate where one has to put his own values)

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

Finally, ChimeraX deals better with displaying the transition pathway if inputs have values between 0 and 1 so we type :

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

Our first structure has been preprocess and is ready. Assume the same operations have been done to the second structure

3. In the case where both structure are not on the same grid, one can use **volume resample** to correct this by typing (assuming the first and second structure have IDs 1 and 2)

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

All structures are now preprocessed and have the same size, we can start using *MorphOT*

## 3.2 displaying morphOT movie

To produce a OT-trajectory, just type down :

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

Options **reg** and **frames** will have the most effect on how the movie looks like.
Alternatively, for a faster computed, 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 OT computed frames between which linear interpolation is done.

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

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

## 4 Appendix

### 4.1 Running on GPU

Before further development of our plugin, the use of GPUs by MorphOT in USCF ChimeraX is not automatic. GPU computing is only available for GPUs with CUDA cores (most NVIDIA GPUS) The steps are the following :

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

### 4.2 Details about the algorithm

**Refresher:** To produce trajectories between two EM density maps $V_0$ and $V_1$, we use the following interpolant

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

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, finding efficient ways of computing it is an active research field. In this work, we followed the recent method introduced by Solomon *et al.* [1]. First, $\mathcal{W}_2^2$ is regularized with the entropy $H(\pi)$ of the *transportation plan* $\pi$ (as inspired by [2]), defined as

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

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

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

where $\gamma > 0$ is the *entropy parameter* (as $\gamma$ increases, more spread-out solutions are promoted [1]). 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 [3]: 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 $t \overset{\text{def.}}{=} \gamma/2$, 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

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

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 $\pi$ in closed form which can be efficiently computed using Sinkhorn-Knopp matrix scaling algorithm. [4]. We refer to [1] and its supplemental material for more details.

**Computation of $V_t$:** Now that Equation 5 provides a good approximation of $\mathcal{W}_2^2$, we re-evaluate $V_t$ using this approximation :

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

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

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. [5].

We obtain the following algorithm for any number of input maps $\{\boldsymbol{\mu_i}\}_i$ and given a set of weights $\{\alpha_i\}_i$ , we search for the barycenter $\boldsymbol{\mu}$ !

**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] 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).

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

[3] 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).

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

[5] 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).