SINGLE SOURCE FOR ENTROPY-REGULARIZED OT ALGORITHMS

Dr. Xiaoming Huo
School of Industrial and Systems Engineering
Georgia Institute of Technology
huo@gatech.edu

Yiling Xie
School of Industrial and Systems Engineering
Georgia Institute of Technology
yxie350@gatech.edu

Yiling Luo
School of Industrial and Systems Engineering
Georgia Institute of Technology
yluo373@gatech.edu

Samuel Hart
College of Computing
Georgia Institute of Technology
samuel.hart@gatech.edu

May 6, 2022

ABSTRACT

Optimal transport (OT) research has exploded in recent years due to its applications in fields like domain adaptation, computer vision, natural language processing and more. A large body of the research is in finding computationally inexpensive algorithms for solving for the OT distance between two measures. In our research work, we hope to establish a collection of a subset of these recent algorithms as well as the numerical color transfer experiment, which can be used to compare the algorithms. Our hope is that by providing a collection of implementations of these algorithms, their associated papers, and a simple experiment with which to compare them, that we may make future research in the field easier.

Keywords Optimal Transport · Wasserstein Distance · Optimization · Machine Learning · Computer Vision

1 Introduction

1.1 The OT Problem

The discrete OT problem is parameterized by a discrete source measure r, a discrete target measure c, and a cost or metric matrix $\mathbf{M}$. The source and target measures belong to the simplex $\Sigma_n = \{x \in \mathbb{R}^n_{\geq 0} : x^T \mathbf{1}_n = 1\}$, where $\mathbf{1}_n$ is the n dimensional vector of ones. $M \in \mathbb{R}^{n\times n}_{\geq 0}$, where M_{ij} is the cost of moving a unit value of mass or probability from r_i to c_j. The goal of the problem then is to find the coupling matrix $\mathbf{P} \in \mathbb{R}^{n\times n}_{\geq 0}$ that defines the transportation plan and minimizes

$$\langle \mathbf{M}, \mathbf{P} \rangle = \sum_{i,j=1}^{n} \mathbf{M}_{ij}\mathbf{P}_{ij}.$$

Here, $\mathbf{P}_{ij}$ defines the amount of mass or probability that we move from the entry r_i to the entry c_j. Note that $\mathbf{P}\mathbf{1}_n = r$ and $\mathbf{P}^T\mathbf{1}_n = c$, which simply says that we cannot take more mass or probability from r_i than what exists there, and that we cannot distribute more probability to c_j than what exists at c_j. We can define the transportation polytope, or set of transportation plans from r to c, as the set

$$U(r, c) = \{\mathbf{P} \in \mathbb{R}^{n\times n}_{\geq 0} : \mathbf{P}\mathbf{1}_n = r, \mathbf{P}^T\mathbf{1}_n = c\}.$$

If the matrix $\mathbf{M}$ is a metric matrix or distance matrix satisfying the following properties,

$$\forall i \leq n, \mathbf{M}_{ii} = 0, \text{ and } \forall i, j, k \leq n, \mathbf{M}_{ik} \leq \mathbf{M}_{ij} + \mathbf{M}_{jk},$$

then the quantity $\langle \mathbf{M}, \mathbf{P} \rangle$ is itself a distance between the measures r and c. Simply solving for

$$\min_{\mathbf{P} \in U(r,c)} \langle \mathbf{M}, \mathbf{P} \rangle,$$

is unfortunately computationally expensive. However, [1] reformulated this problem as

$$\min_{\mathbf{P} \in U(r,c)} \langle \mathbf{M}, \mathbf{P} \rangle + \gamma H(\mathbf{P}), \tag{1}$$

which opened the door for a new wealth of research into entropy-regularized algorithms for solving the OT problems. For more information on the OT problem and its different formulations, including its dual problem formulation, see [2].

1.2 Included Algorithms

Note that there are a variety of algorithms for solving the OT problem and that, for now, this repository only contains seven algorithms. They are given below in the table with a reference to their associated papers and their theoretical complexities.

Algorithm	Complexity
Sinkhorn, [1], [3], [4]	n^2/ε^2
Greenkhorn, [3], [5]	n^2/ε^2
Stochastic Sinkhorn, [5]	n^2/ε^3
Stochastic Average Gradient, [6], [7]	UNSURE
APDAGD, [4]	$n^{5/2}/\varepsilon$
AAM, [8]	$n^{5/2}/\varepsilon$
APDRCD, [9]	$n^{7/2}/\varepsilon$

Table 1: Summary of included algorithms

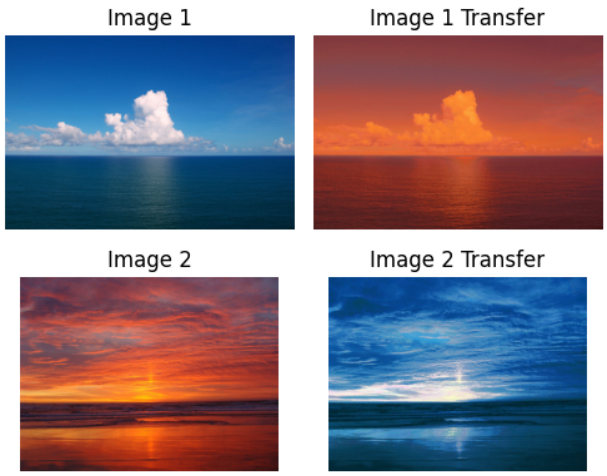
Note that we could not find a complexity for the Stochastic Average Gradient algorithm based on the parameters associated with the OT problem.

1.3 Notes on the Python Optimal Transport Package

This work could not be completed without giving notice to the `Python Optimal Transport` (POT) [10] package for Python. In our work, we make use of their implementations for the Sinkhorn, Greenkhorn, Stochastic Sinkhorn, and Stochastic Average Gradient algorithms. However, the POT package was made for use in both practical and academic machine learning applications and do not include the additional three algorithms we provide. However, the POT package is open-source and as such it is theoretically possible to integrate this work with theirs.

1.4 The Color Transfer Experiment

In addition to providing implementations for the previously mentioned algorithms, we also include an example of the color transfer experiment for which can be used to compare runtimes and the number of operations required for each algorithm. The color transfer experiment works to convert a color palette of one image to the color palette of another image. Below is an example of such an experiment, in which, the AAM algorithm was used.



Note that the images have seemingly swapped color distributions, while the overall structure of the image has remained the same. Our experiment follows the way of computing color transfer via OT as described in [11] and follows the code laid out in the POT example. Color transfer has interesting applications in the imaging and video world for things like color correction of rendered images, panorama stitching, and changing colors of an image to match a different time of day.

2 Algorithms

We now provide some relevant points of information on each of the algorithms included. These points include things like relevance and similarity to other algorithms in the repository, the source of their implementations, what changes were made, etc.

2.1 Sinkhorn

This is the algorithm first proposed to solve the problem (1) and whose original reference is given in [1]. We chose to use the implementation given in the POT toolbox as it was credible and had been developed for performance. We made a few changes to the implementation given in POT to further simplify the code and ensure that it was similarly formatted to our other algorithms. Further information and pseudocode can be found in [2], [4], [5]. To see what changes we made check out the POT code.

2.2 Greenkhorn

The name Greenkhorn comes from the fact that this algorithm was formulated in [3] as a greedy version of the Sinkhorn algorithm. Instead of updating all the variables with each step, as is done in Sinkhorn, this algorithm proposes updating a single coordinate of the variables which is chosen based on the distances of the current estimation of $\mathbf{P}$ from the polytope $U(r, c)$. We again used the implementation given in POT for the same reasons mentioned previously. Further information and pseudocode can be found in [5],[3].

2.3 Stochastic Sinkhorn

Building again off Sinkhorn, this algorithm proposed in [5] operates similarly to that of Greenkhorn, however, it randomly chooses which coordinate to update treating the distance of $\mathbf{P}$ from the polytope $U(r, c)$ as a probability distribution. Once again, this implementation was sourced from the POT package. For more information, see [5].

2.4 Stochastic Average Gradient (SAG)

The Stochastic Average Gradient algorithm belongs to the family of gradient descent algorithms and was first proposed in optimization theory by [6] and was later applied to the OT problem in [7]. We again used the implementation provided in the POT toolbox, however, with much more modification. The reason for this is that the SAG algorithm

given in the POT toolbox runs for a set number of iterations passed into the function. This differs from the previous algorithms in that there is no condition to check if the transportation matrix $\mathbf{P}$ has converged close enough to the transportation polytope $U(r, c)$. To improve runtimes and prevent the algorithm from running the full number of max iterations, we add a condition every 1000 iterations to check to see if it the algorithm has converged sufficiently. The reason this condition is only checked every 1000 iterations, is because the SAG algorithm only computes on the dual variables of the OT problem, and to compute our transportation matrix $\mathbf{P}$ we must compute the other dual variable. This would be computationally expensive and so we settle to only perform the computation of $\mathbf{P}$ every 1000 iterations to lower run times and the number of arithmetic operations. Further information and pseudocode given in [6], [7].

2.5 Adaptive Primal-Dual Accelerated Gradient Descent (APDAGD)

The APDAGD algorithm given in [4] is proposed for a general optimization of which the regularized OT problem is shown to be a special case of. Our implementation is based on the implementation developed for [8] which is given on GitHub. Note that this implementation varies from the pseudocode in [4] in order to provide better numerical stability. Our implementation builds off this implementation, by better specifying the parameters of the algorithm and its local values all in one place. Check the associated reference and GitHub for more theoretical considerations.

2.6 Accelerated Alternating Minimization (AAM)

The AAM algorithm is also proposed for a general optimization problem of which the regularized OT problem is a special case as is shown in [8]. Our implementation comes directly from the implementation they developed and is given on the same Github file previously linked. It includes the same numerical stability additions, and we again combine all the necessary parameters and local values of the algorithms into one function.

2.7 Adaptive Primal-Dual Randomized Coordinate Descent (APDRCD)

We provide our own implementation of the APDRCD algorithm that is given in [9]. Our implementation varies slightly from the pseudocode with added steps to reduce the number of arithmetic operations by avoiding unnecessary computation of unchanging values. This has to do with the fact that it is a coordinate-wise descent algorithm and thus only certain values of a variable change with each step. Unfortunately, we were not able to give a faster implementation for this algorithm and welcome improvements to our implementation.

3 Color Transfer Experiment & Results

The entropy-regularized algorithms are ideal for the color transfer experiment since the regularization term helps to prevent colorization artefacts in the resulting images [11]. In order to perform the color transfer experiment, we first need two images with which to perform the experiment on. For our experiment, we chose the following images:


To make the transfer experiment feasible, we take a random sample of 250 of the pixels of each image, taking the same sample each time to keep results consistent. Then we use these samples to produce the cost matrix $\mathbf{M}$ where $\mathbf{M}_ij$ is the euclidean distance between the rgb values of ith pixel in the sample from one image to jth pixel in the sample from the other image. We initialize our source and target measures, r, c, as the uniform distribution over 250 points. It is uniform since each pixel had equal probability of being sampled. Next, we use one of OT algorithms to transform each sample to the other. The transform can then be extended to the whole space by finding the nearest in-sample pixel for each out-of-sample pixel and adding back the distance between their rgb values in the original image. To quantify this let x be an out of sample pixel and T be the transformation given from the OT algorithm. Then the tranformation of x is

given as follows

$$T(x) = T(X_{i(x)}) + x - X_{i(x)}, \quad \text{where } i(x) = \underset{1 \le i \le 250}{\operatorname{argmin}} \|x - X_i\|,$$

and the X_i's are the in-sample pixels. This helps to improve the visual quality of the color transfer and maintain textural details of the original image.

Before moving onto the results of the experiment, we have to make note to some necessary experiment setup for the algorithms. Firstly, results for the APDRCD algorithm could not be gathered as it's runtime was simply too long for an OT problem of this size. It's complexity is much worse with respect to n compared to the other algorithms and so it was left out for this experiment. Second, AAM had poor numerical stability for any accuracy $\varepsilon \le 0.01$, as it's Lipschitz constant grew or shrunk exponentially and eventually led to numerical errors that prevented convergence. It is possible that it could be adjusted for the experiment or have better performance in another experiment. Finally, the Sinkhorn, Greenkhorn, Stochastic Sinkhorn, and SAG algorithms required larger regularization terms γ for (1) than APDAGD or AAM and so we passed in a regularization term of $\gamma = 1e - 1$ for those four algorithms.

We ran used each algorithm except APDRCD for the color transfer experiment following the notes of the set up above. The experiment was run 9 times with each algorithm on the following list of accuracies $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. Below, we show the results of each algorithm with accuracy $\varepsilon = 0.02$.

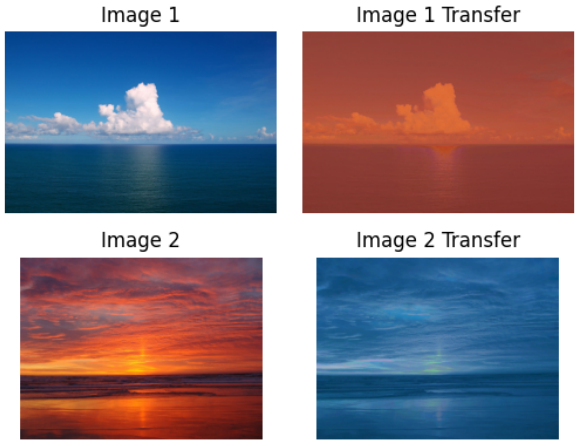


Figure 1: Sinkhorn Color Transfer, $\varepsilon = 0.02$

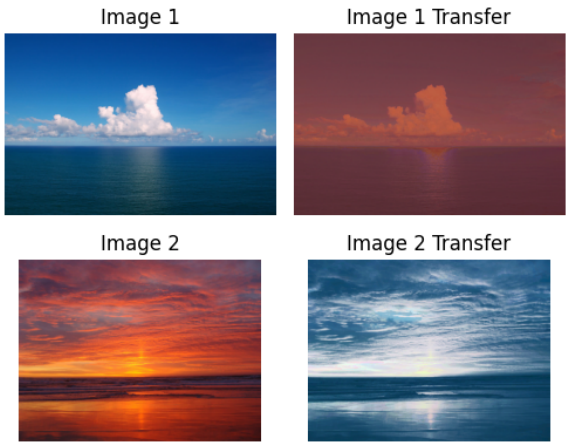


Figure 2: Greenkhorn Color Transfer, $\varepsilon = 0.02$

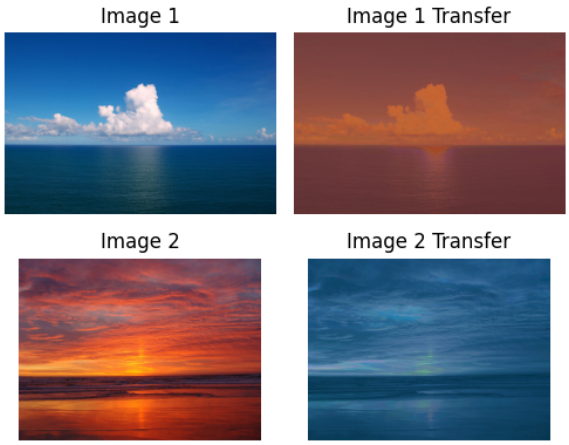


Figure 3: Stochastic Sinkhorn Color Transfer, $\varepsilon = 0.02$

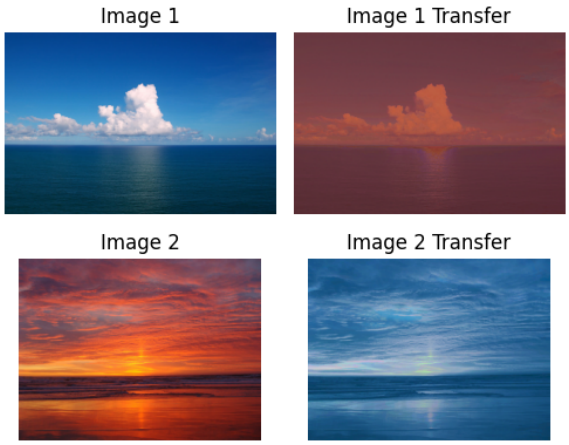


Figure 4: SAG Color Transfer, $\varepsilon = 0.02$

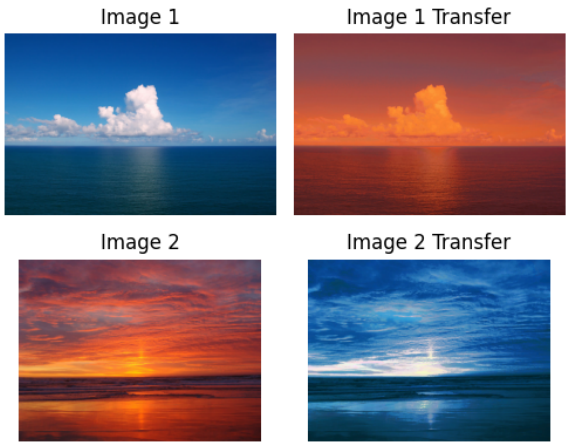


Figure 5: APDAGD Color Transfer, $\varepsilon = 0.02$

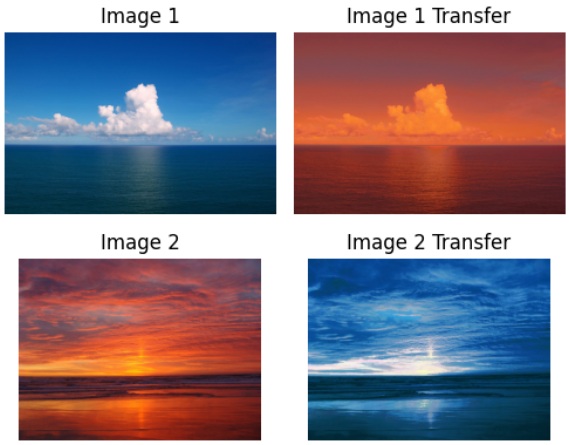
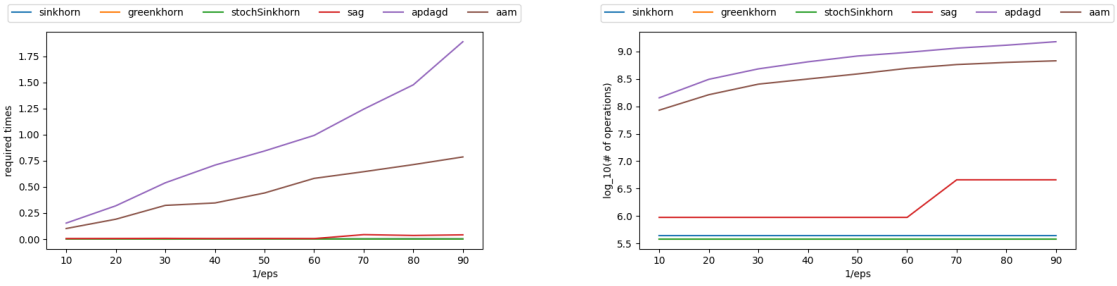


Figure 6: AAM Color Transfer, $\varepsilon = 0.02$

Note that the accelerated algorithms seemed to provide the highest quality results in terms of details. The other algorithms likely lack in detail due to the fact that they used a larger regularization term $\gamma = 0.1$ than those used by the accelerated methods, which set $\gamma = \varepsilon/(3 * \log(250))$. Each algorithm has differing parameter constraints that makes it difficult to find a good range of accuracy for comparison. In the case of this experiment, it was necessary to run the non-accelerated algorithms with $\gamma = 0.1$ for numerical stability, however, results could be better if we allowed for a range of accuracy parameters smaller than 0.01. Unfortunately, this led to numerical issues with the AAM algorithm and so we chose this range of accuracy to properly compare the run times and number of arithmetic operations of the algorithms. Those results are summarized below.



We found that the non-accelerated algorithms operated significantly faster at the these levels of accuracy, however, it was at the cost of visual quality of the result. Sinkhorn was capable of producing higher quality results with smaller regularization values, while the other non-accelerated algorithms became unstable much quicker with smaller values. We conclude that although there is a small cost in run times and the number of operations for the accelerated algorithms, they provide more visually accurate results. Furthermore, we found that there is no set of parameters that provides good results for every algorithm and that one should tune the parameters specific to the algorithm of choice.

4 Project Repository

The language of choice for this project was Python for its user friendliness, readability, and packages. Regarding packages, this project makes use of `NumPy` and `matplotlib` which will be necessary to run the code. We chose to create this project as a collection of separate Python files to make it easier to find what one is looking for. We chose not to implement it as a Jupyter Notebook due to the different dependencies that may arise between different cells, as well as for better readability. As of right now, the project consists of just five python files: `algos.py`, `util.py`, `test.py`, `CTOT.py`, and `main.py`.

Arguably the most important of these files is the `algos.py` file, which contains the aforementioned algorithms and their documentation. They are setup to return the primal variable associated with the OT problem, as well as the run time in seconds and the number of arithmetic operations to reach convergence. Since the OT problem is specified to the

algorithm as parameters, it is easy to use these implementations for other experiments or tests as one can just define the OT problem in the context of their experiment and pass it into each algorithm to compare their results.

The `util.py` file contains some utility functions that were commonly used in the algorithms. One such example of one of these is the `round` function which is used in APDAGD, AAM, and APDRCD. It's implementation comes from the pseudocode given in [3] as well as the GitHub files for [8]. The `test.py` contains a basic OT problem for the algorithms to solve and prints out their runtimes in seconds and their number of arithmetic operations. These easily editable to handle different types of OT problems and incorporate other algorithms into the test.

The `CTOT.py` file contains the functions necessary for the color transfer experiment. The functions follow from the POT example on the color transfer problem. This includes transforming the images into matrices, computing the OT transport matrix and transferring the color distributions, and plotting their results.

The `main.py` file is set up to run the color transfer experiment using the first six algorithms and plots their runtimes as well as their number of arithmetic operations to reach convergence. Note that if one were to add another experiment to the repository, then they could similarly set up this file to perform the same function on a different experiment.

In addition to these Python files, the repository also contains a directory to store the results and data associated with the color transfer experiments or other experiments that may be added later. The project repository can be found here.

5 Conclusion

In summary, we provide a single repository for some of the popular and recent developments in algorithms solving the entropy-regularized OT problem given by (1) and originally formulated in [1]. In addition to providing the algorithms, we also provide a numerical experiment that can be used to compare the algorithms with different levels of accuracy. This paper serves as a guide to the project with sources of information and further clarifications behind choices in the repository. We hope that in the future, this project can be further expanded upon with more algorithms for both the entropy-regularized OT problem, as well as other problems such as Wasserstein Barycenters. It would also be beneficial to add other numerical experiments found commonly in literature, such as those on synthetic image data and the MNIST data set as seen in [9],[8].

References

[1] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, pages 2292–2300, 2013.

[2] Gabriel Peyré and Marco Cuturi. Computational optimal transport. 2018.

[3] Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1961–1971, 2017.

[4] Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational optimal transport: Complexity by accelerated gradient descent is better than by sinkhorn's algorithm. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1367–1376. PMLR, 10–15 Jul 2018.

[5] Brahim Khalil Abid and Robert Gower. Stochastic algorithms for entropy-regularized optimal transport problems. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1505–1512. PMLR, 09–11 Apr 2018.

[6] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient, 2013.

[7] Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[8] Sergey Guminov, Pavel Dvurechensky, Nazarii Tupitsa, and Alexander Gasnikov. On a combination of alternating minimization and nesterov's momentum. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3886–3898. PMLR, 18–24 Jul 2021.

[9] Wenshuo Guo, Nhat Ho, and Michael Jordan. Fast algorithms for computational optimal transport and wasserstein barycenter. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2088–2097. PMLR, 26–28 Aug 2020.

[10] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[11] Sira Ferradans, Nicolas Papadakis, Gabriel Peyré, and Jean-François Aujol. Regularized discrete optimal transport. *CoRR*, abs/1307.5551, 2013.