R1 We appreciate your very constructive comments. Good point. Some new experiments comparing COPT distance to SOTA GW distance: With 6 classes of synthetic graphs, 18 queries on 60 graphs, with nearest-neighbors-based classifier, COPT achieves acc. 83.5±3.4, and GW 62.2±6.6. This is repeated 20 times. Will expand and include these. (The existing vectorize-and-retrieve experiments were designed to be highly practical on GPUs, leveraging the benefits of COPT during the vectorization preprocessing.) • Will update paper to emphasize distance aspect as you suggest. Algorithm#1 indeed already computes the COPT distance as the objective during optimization. • Indeed interesting to vary optimizer. To sketch a 20-node graph down to 5, currently each iteration takes 3.7±.05 ms, after updating to LBFGS, each iteration takes 128±3 ms. Likely because LBFGS is a second order method, and may evaluate the forward pass multiple times per iteration. Will try others. • There are in fact interesting relations to graph edit distance: in COPT, the matrix P allows *continuous* (e.g. partial) flow between vertices, while edit distance involves matching the vertices *one-to-one* (plus insertions & deletions). Edit distance gives the *same cost* to delete an arbitrary edge, while COPT distance is higher if the edges deleted were important to the graph spectrum (i.e. edges connecting distinct clusters). A continuous relaxation of edit distance would resemble [LFF+15] more than COPT for this reason. • Added the fact the measure of variance is standard deviation, number of times 5.2 is repeated, will update broader impact & add details to README. Thanks for these. [LFF+15: Lyzinski et al, Graph matching: Relax at your own risk. 2015.]

R2 We thank you for your points. Indeed Batson-Spielman-Srivastava and related works from TCS are fascinating. The primary reason for choosing the included literature was an alignment in goals. COPT was designed to reduce the number of *vertices*, producing *dense* Laplacians; BSS and related works reduce the number of *edges*, producing *sparse* Laplacians. One benefit of dense data is that they are well-suited for efficient processing on GPUs, such as in many ML applications. • BSS focuses on proving theoretical guarantees on constructing graph sparsifiers, it uses a *discrete* optimization method where one edge is added to the sparse graph per step (though weights are chosen continuously); COPT uses a *continuous* optimization method where every variable can change at every time step. Though discrete optimization can more readily produce provable guarantees, continuous optimization is often preferred in practice, e.g. continuous relaxations of discrete problems such as LP relaxation. • Another effect of the proof-driven construction is that the constructed sparsifier may not be useful for applications such as graph classification. E.g. BSS sparsifies complete graphs to sparse expander graphs, which have intricate structures not present in original graph. Attempts to classify the sparsifier could see this structure instead of properties of the original graph. COPT empirically reduces complete graphs to complete or nearly-complete graphs. • We will add these. • Similarly, the approach (e.g. in Jambulapati-Sidford) of defining *multiple* randomized sketches and then taking the median to obtain a good Laplacian approximation may not be practical for applications. • Good call on motivating §3.2 with downstream applications, *metric* approximations are good for applications where shortest paths between nodes are important, e.g. classifying road networks, classifying graphs arising from physical objects where edge lengths carry geometric information. *Spectral* approximations are useful for when the number of paths matter more, e.g. graph partitioning, where one aims to minimize the number of edges cut (one way is to iteratively coarsen graphs and find cuts on smaller graphs). Similarly COPT is applicable to graph clustering. • Will state nonconvexity earlier in the paper; comment on the theoretical fast matrix multiplication methods vs practice (indeed the theoretical fastest method is not practical); add references as suggested; comment on runtimes; and fix typos. • The only change to work with weighted graphs is to use the *weighted* Laplacian instead of ordinary Laplacian. The proofs only use the fact that L_X is a symmetric PSD matrix. We brought in the additional concept of weightings only when needed, but the potential for applications to weighted graphs is a good point.

R3 In Eq. (4), we define a loss function for a transport map between $\mathbb{R}^X$ and $\mathbb{R}^Y$, as a generalization of the loss in GOT from permutations to arbitrary matrices P. A priori this is just a formula, but in Lemma 3.1, we prove it is equal to a distance in $\mathbb{R}^X \times \mathbb{R}^Y$, after embedding $\mathbb{R}^X$ and $\mathbb{R}^Y$ into $\mathbb{R}^{X \times Y}$, which we use to calculate it. In the special case when P is a permutation, these embeddings have the same image, reducing the formula to optimal transport in just $\mathbb{R}^X$ or $\mathbb{R}^Y$ as in GOT. • §3.2 compares COPT with GW, and discusses how COPT preserves paths-counting in relation to the spectrum. • f is a free variable in the domain of integration, i.e. the space of functions on $\mathbb{R}$. Will specify this in the paper. Thanks.

R4 Thanks much for your comments. It is indeed useful to compare with the WL kernel. Some new experiments: On 60 synthetic graphs across 6 categories with 18 query graphs, using exact nearest-neighbor search with COPT as a distance: acc. 83.7±3.9; SVM with WL kernel: 80.5±9.0. This is repeated 20 times. • Choosing between the two depends on the task and the dataset: For a large dataset of size N, we expect a WL kernel-based SVM to be more practical than using linear scan with COPT distance. As an SVM can be trained during preprocessing, allowing *constant* time query classification, whereas exact nearest-neighbor search requires time *linear* in N. • For small datasets, COPT distance can be advantageous: with only three samples per class, using nearest-neighbor classification with COPT distance gives accuracy 90.7±5.2, and the WL kernel-based SVM gives 77.8±7.1. • For tasks such as predicting *continuous* attributes, an SVM with WL kernel can't readily be applied, but a nearest-neighbors classifier with COPT distance still can. • Will expand on these and add to the paper.