An Analysis of Euclidean vs. Graph-Based Framing for Bilingual Lexicon Induction from Word Embedding Spaces

Kelly Marchisio[1], Youngser Park[4,5], Ali Saad-Eldin[3], Anton Alyakin[2], Kevin Duh[1,5], Carey Priebe[2,5], Philipp Koehn[1]

Depts. of [1]Computer Science, [2]Applied Mathematics and Statistics, and [3]Biomedical Engineering
[4]Center for Imaging Science, [5]Human Language Technology Center of Excellence
Johns Hopkins University

{kmarc,youngser,asaadel1}@jhu.edu, aalyaki1@alumni.jh.edu
kevinduh@cs.jhu.edu, {cep, phi}@jhu.edu

Abstract

Much recent work in bilingual lexicon induction (BLI) views word embeddings as vectors in Euclidean space. As such, BLI is typically solved by finding a linear transformation that maps embeddings to a common space. Alternatively, word embeddings may be understood as nodes in a weighted graph. This framing allows us to examine a node's graph neighborhood without assuming a linear transform, and exploits new techniques from the graph matching optimization literature. These contrasting approaches have not been compared in BLI so far. In this work, we study the behavior of Euclidean versus graph-based approaches to BLI under differing data conditions and show that they complement each other when combined. We release our code at https://github.com/kellymarchisio/euc-v-graph-bli.

1 Introduction

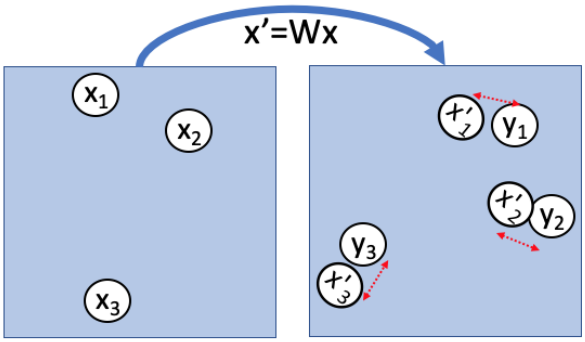
Bilingual lexicons are useful in many natural language processing tasks including constrained decoding in machine translation, cross-lingual information retrieval, and unsupervised machine translation. There is a large literature inducing bilingual lexicons from cross-lingual spaces. "Mapping" methods based on solving the orthogonal Procrustes problem and its generalizations are popular, where languages are mapped to a common space from which a lexicon is extracted. This has been successful when word embedding spaces are roughly isomorphic, but fails as embedding spaces diverge (Søgaard et al., 2018; Vulić et al., 2019).

Rather than word embeddings in Euclidean space, we can work with weighted graphs derived from embeddings. Graphs may be full-connected or sparse to capture the underlying data manifold. For instance, we may create a similarity graph with words as nodes and cosine distance between word vectors as edges. The use of graphs in NLP has a rich history, for tasks as varied as summarization, part-of-speech tagging, syntactic parsing, information extraction, measures of semantic similarity, and evaluation of cross-lingual word embeddings (Mihalcea and Radev, 2011; Nastase et al., 2015; Fujinuma et al., 2019). Graphs can also represent rich relationships like hyponym/hypernym, syntactic roles, synonymy such as in WordNet (Miller, 1995) and Freebase (Bollacker et al., 2008). We focus on fully-connected graphs derived from pairwise cosine similarities between word embeddings in this work.

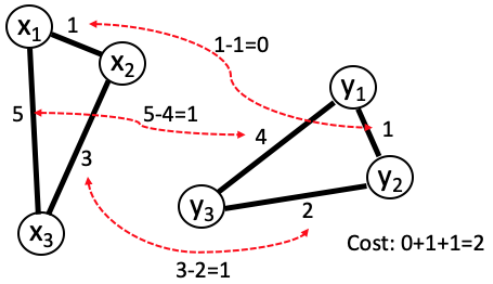
The Euclidean view, exemplified by methods solving the Procrustes problem, works with embedding spaces and assumes the existence of a linear transform that maps the spaces. The graph-based view works with graphs for each language and directly performs matching on edge pairs based on neighborhood information. This view is exemplified by graph matching methods that solve the quadratic assignment problem from the combinatorial optimization literature. Ruder et al. (2018) and Haghighi et al. (2008) incorporate related techniques for bilingual lexicon induction. We use Seeded Graph Matching (SGM; Fishkind et al., 2019) as representative of this class of approach. Figure 1 illustrates the differences between the framings; while they both exploit the idea that words with similar neighbors (in Euclidean or graph space) should be translations of one another, they implement the idea in very different ways.

We explore these two different views of BLI. Our main contributions are (a) a thorough comparison of Euclidean vs. graph-based framings to BLI under varying data conditions, and (b) a method for combining both approaches that achieves better performance than either alone.

We organize our work into three main experimental setup and results sections. First, we compare standard algorithms of performing BLI via solutions to the orthogonal Procrustes problem ("Procrustes", for short) and SGM in Section 4; we



(a) Linear transform W maps x to common space as y: Cost measures distance between vectors x' and y



(b) Quadratic assignment for graph matching: Cost measures difference between edge weights

Figure 1: Comparing **Euclidean (Procrustes)** vs. **Graph (SGM)** views. (a) Euclidean view assumes common embedding space and computes costs based on pairs of vectors. (b) Graph-based view assumes graph structure and computes cost based on pairs of edges. Both exploit within-language neighborhood info but in different ways.

find that their performance varies depending on the number of seeds. SGM appears better when using less seeds. Second, as it is common to improve results by bootstrapping, we compare iterative versions of Procrustes and SGM in Section 5. We find that Iterative Procrustes improves much more rapidly than Iterative SGM. We also introduce stochastic variants of the iterative algorithms to improve robustness and experiment with active learning setups. Finally, we present our combined system which outperforms individual Procrustes and SGM approaches in Section 6.

2 Background

BLI begins with two word embedding matrices: $\mathbf{X} \in \mathbb{R}^{\mathbf{n}\times\mathbf{d}}$ represents the d-dimensional word embeddings for n vocabulary items in language X, and $\mathbf{Y} \in \mathbb{R}^{\mathbf{m}\times\mathbf{d}}$ represents the m embeddings separately trained on monolingual data in language Y. We assume seeds $\{(x_1, y_1), (x_2, y_2), ...(x_s, y_s)\}$ are given, which are supervised labels indicating translation correspondence between vocabulary items in the languages. We sort the corresponding submatrices of $\mathbf{X}$ and $\mathbf{Y}$ so each row of $\overline{\mathbf{X}} \in \mathbb{R}^{s\times d}$ and $\overline{\mathbf{Y}} \in \mathbb{R}^{s\times d}$ corresponds to the seeds. Usually, s is strictly smaller than both n and m and the goal is to find translation correspondences in the remaining words.

Procrustes and linear transforms: The popular Procrustes-based methods for BLI (e.g. Artetxe et al., 2016a, 2019; Conneau et al., 2018; Patra et al., 2019) match seeds by calculating a linear transformation $\mathbf{W}$ by a variant of the below:

$$\min_{\mathbf{W}\in\mathbb{R}^{\mathbf{d}\times\mathbf{d}}} ||\overline{\mathbf{X}}\mathbf{W} - \overline{\mathbf{Y}}||_{\mathbf{F}}^{\mathbf{2}} \quad (1)$$

If $\mathbf{W}$ is required to be orthogonal, then distances between points are unchanged by the transform and a closed form solution can be computed by singular value decomposition (Schönemann, 1966).

Once languages are mapped to the same space by $\mathbf{W}$, nearest neighbor search finds additional translation pairs. If $\mathbf{W}$ is known, one can find translations by optimizing over permutations Π:

$$\min_{\mathbf{P}\in\mathbf{\Pi}} ||\mathbf{X}\mathbf{W} - \mathbf{P}\mathbf{Y}||_{\mathbf{F}}^{\mathbf{2}} \quad (2)$$

$\mathbf{P} \in \{\mathbf{0}, \mathbf{1}\}^{\mathbf{n}\times\mathbf{n}}$ is permutation matrix that shuffles the rows of $\mathbf{Y}$. If we enforce the 1-to-1 correspondence, this is linear assignment problem that is solvable in polynomial time, e.g. with the Hungarian algorithm (Kuhn, 1955) or Wasserstein methods (Grave et al., 2019a). In the NLP literature, a large number of methods are based on the same underlying idea of linear transform followed by correspondence search/matching (see Related Work).

To extract lexicons, one performs nearest neighbor search on the transformed embeddings. To mitigate the hubness problem (where some words are close to too many others) (Radovanovic et al., 2010; Suzuki et al., 2013), Conneau et al. (2018) modifies the similarity using cross-domain similarity local scaling (CSLS) to penalize hubs. For x, y in embedding space V:

$$\text{CSLS}(x, y) = 2\cos(x, y) - \text{avg}(x, k) - \text{avg}(y, k)$$

$$\text{avg}(v, k) = \frac{1}{k} \sum_{v_n \in N_k(v,V)} cos(v_n, v)$$

$N_k(v, V)$ returns the k-nearest-neighbors to $v \in V$ by cosine similarity (typically $k = 10$).

Graph matching: In fields such as pattern recognition, network science, and computer vision, there exist a large body of related work termed "graph matching." Rather than assuming the existence of a linear transform between the embedding spaces, these methods start with or construct two graphs and try to match vertices such that neighborhood structure is preserved. Intuitively, the motivation of preserving neighborhood structure is the same as Procrustes methods, but the absence of linear transform $\mathbf{W}$ is an important distinction that potentially makes graph matching more flexible. Indeed, some recent BLI work argue against linear transforms (Mohiuddin et al., 2020) and discuss the failure modes due to lack of isometry (Søgaard et al., 2018; Nakashole and Flauger, 2018; Ormazabal et al., 2019; Glavaš et al., 2019; Vulić et al., 2019; Patra et al., 2019; Marchisio et al., 2020).

For BLI, we may build the graphs as $\mathbf{G_x} = \mathbf{XX^T}$ and $\mathbf{G_y} = \mathbf{YY^T}$. For standard graph matching objectives, we restrict the vocabularies of $\mathbf{X}$ and $\mathbf{Y}$ to equal size, thus $\mathbf{G_x}, \mathbf{G_y} \in \mathbb{R}^{\mathbf{n \times n}}$. We find the optimal relabeling of nodes such that:

$$\min_{\mathbf{P} \in \mathbf{\Pi}} ||\mathbf{G_x} - \mathbf{P G_y P^T}||_{\mathbf{F}}^{\mathbf{2}} \quad (3)$$

This is an instance of the quadratic assignment problem and is much harder than Eq. 2. It is NP-Hard (Sahni and Gonzalez, 1976) but various approximation methods exist. Vogelstein et al. (2015) use the Frank-Wolfe method (Frank et al., 1956) to find an approximate doubly-stochastic solution, then project onto the space of permutation matrices.

When seeds are available, SGM can be applied to solve the amended objective in Equation 4, where s is the number of seeds and Π_{n-s} is the set of permutation matrices for the $n - s$ non-seed words. See Appendix for details.

$$\min_{\mathbf{P} \in \mathbf{\Pi_{n-s}}} \|\mathbf{G_x} - (\mathbf{I_s} \oplus \mathbf{P})\mathbf{G_y}(\mathbf{I_s} \oplus \mathbf{P})^{\mathbf{T}}\|_{\mathbf{F}}^{\mathbf{2}} \quad (4)$$

2.1 Differences between Procrustes and SGM

Two differences in behavior of Procrustes vs. SGM are worth discussing for their relevance to BLI.

Procrustes is many-to-one; SGM is one-to-one. After solving the orthogonal Procrustes problem, translation pairs are selected by finding the $y \in \mathbf{Y}$ that is closest to the mapped source word in $x_w \in \mathbf{XW}$. It is possible that the nearest neighbor to both $x_{w_1} \in \mathbf{XW}$ and $x_{w_2} \in \mathbf{XW}$ may be $y_1 \in \mathbf{Y}$, so $\{(x_{w_1}, y_1), (x_{w_2}, y_1)\}$ may be induced as final translation hypotheses. Conversely, SGM solutions are strictly one-to-one; If x_{w_1} is paired with y_1, then x_{w_2} cannot be. As such, SGM may avoid hubs naturally without CSLS. A way around the one-to-one restriction is to use SoftSGM. For instance, if x_{w_1} is paired with y_1 on 40% of internal runs of SoftSGM and x_{w_2} is paired with y_1 on 40% of runs (and y_1 is the most frequent pairing for both x_{w_1} and x_{w_2}), we may induce $\{(x_{w_1}, y_1), (x_{w_2}, y_1)\}$ as final hypotheses.

Procrustes is soft-seeded; SGM is hard-seeded. Procrustes is "soft-seeded"; giving seed (x_1, y_1) does not guarantee that x_1 and y_1 will be paired in the solution, because y_1 may not be the nearest neighbor to the mapped x_{w_1}. Conversely, SGM is "hard-seeded": pairings given as seeds will always appear in the solution. This is ideal when one is confident about the quality of the seeds, but means that SGM is not robust to errors in the seed set.

3 Experimental Setup

Because there are three methods and results sections, we detail the experimental setup first. We evaluate on English→German (En-De) and Russian→English (Ru-En).

Monolingual Word Embeddings We use 300-dimensional monolingual word embeddings trained on Wikipedia using fastText (Bojanowski et al., 2017).[1] We normalize to unit length, mean-center, and renormalize, following Artetxe et al. (2018a) ("iterative normalization", Zhang et al. (2019)).

Data & Software Bilingual dictionaries from MUSE[2] are many-to-many lexicons of the 5000 most-frequent words from the source language, paired with one or more target-side translations. We filter each lexicon to be one-to-one for simplicity of analysis. For source words with multiple target words, we keep the first occurrence. This is equivalent to randomly sampling a target sense for polysemous source words because target words are in arbitrary order. En-De originally contains 14667 pairs, and 4903 remain after filtering. Ru-En has 7452 pairs, reduced to 4084. We use 100-4000 pairs as seeds, chosen in frequency order. The rest are the test set. Seed/test splits are in Table 1. We use the public implementation of SGM with random initialization from Graspologic[3] (Chung

[1] https://fasttext.cc/docs/en/pretrained-vectors.html
[2] https://github.com/facebookresearch/MUSE
[3] https://github.com/microsoft/graspologic

et al., 2019). We leave all other hyperparameters as their defaults (maximum Franke-Wolfe iterations: 30 with epsilon stopping criterion = 0.03, shuffle_input=True).

Seeds	100	200	500	1000	2000	4000
En-De Test	4803	4703	4403	3903	2903	903
Ru-En Test	3984	3884	3584	3084	2084	84

Table 1: Seed/test set splits for En-De, Ru-En.

4 Non-Iterative Experiments

4.1 Methods

Procrustes We compare Procrustes versus SGM methods when each is run once. We solve the orthogonal Procrustes problem of Equation 1 over known seeds and apply the linear transform $\mathbf{W}$ to the entire source embedding matrix $\mathbf{X}$. For each mapped source word in $\mathbf{XW}$, we select y from target embedding matrix $\mathbf{Y}$ with the minimum CSLS score as the translation.

SGM We construct graphs $\mathbf{G_x} = \mathbf{XX^T}$ and $\mathbf{G_y} = \mathbf{YY^T}$, which are matrices of cosine similarity. We solve Equation 4 using the SGM algorithm from Fishkind et al. (2019). We implement Fishkind et al. (2019)'s SoftSGM algorithm by running SGM ten times, each time using a different random initialization for the permutation matrix $\mathbf{P}$. This gives a probability distribution over potential matches.

Standard metrics for BLI are precision@1 and precision@5 (p@1, p@5). Evaluating p@1 is straightforward. For Procrustes p@5, we select the five nearest neighbors per source word. Because SGM only makes one guess per source word, we calculate p@5 using SoftSGM (Fishkind et al., 2019), which returns a probability distribution over possible matches given multiple runs of SGM. We select the top five hypotheses per source word from the probability distribution.[4] We calculate recall@5 and F1@5 analogously.

4.2 Results

Table 2 shows non-iterative results. SGM outperforms Procrustes in nearly all scenarios, and the effect with less seeds is particularly marked: Procrustes scores just 4.1% with 100 seeds and 16.6% with 500 seeds for Ru-En, while SGM scores 50.1% and 52.2%, respectively. With a moderate number of seeds, Procrustes and SGM perform similarly.[5]

	En-De		Ru-En	
Seeds	Procrustes	SGM	Procrustes	SGM
100	3.6	**45.8**	4.1	**50.1**
200	16.1	**47.3**	16.6	**52.2**
500	44.9	**51.9**	45.3	**56.0**
1000	**57.2**	54.9	56.6	**58.1**
2000	**63.1**	61.5	62.7	**67.1**
4000	70.8	**74.2**	67.9	**89.3**

Table 2: P@1 of Procrustes vs. SGM.

We evaluate p@5, recall@5, and F1@5 in Table 3. SGM has considerably higher precision and F1 than Procrustes across all experiments (by 50+ percentage points in extreme cases) but Procrustes generally has greater recall when seed size is 500 or greater. With 100 or 200 seeds, SGM outperforms Procrustes across-the-board. We note the difference in the number of translation hypotheses induced for each method in "Total Hyps."

5 Iterative Experiments

5.1 Methods

It is popular to use the Procrustes solution iteratively. One applies the transformation calculated via Procrustes, extracts a dictionary of translation candidates, then uses those as seeds for the next round of Procrustes. We develop an analogous iterative algorithm for SGM. Figure 2 illustrates the two related approaches.

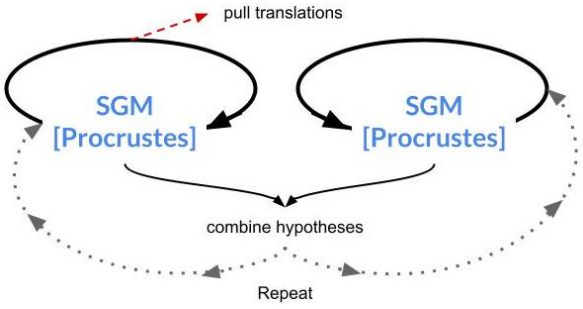


Figure 2: IterSGM [or IterProc]. Run SGM [or Procrustes] in forward & reverse directions. Combine hypotheses and pass as seeds to SGM [Procrustes]. Pull final translations on last iteration from forward run.

We run SGM or Procrustes and extract potential translation pairs in source→target and

[4] There may be less than five hypotheses available per source word if there is not great diversity in output hypotheses.

[5] SoftSGM performs similarly to SGM, so is not reported.

	Seeds	Precision		Recall		F1		Total Hyps.	
		Procrustes	SoftSGM	Procrustes	SoftSGM	Procrustes	SoftSGM	Proc.	SoftSGM
En-De	100	2.2	**30.2**	11.2	**53.5**	3.7	**38.6**	24015	8516
	200	6.8	**34.8**	33.9	**53.3**	11.3	**42.1**	23515	7203
	500	13.9	**43.1**	**69.6**	55.7	23.2	**48.6**	22015	5694
	1000	15.9	**48.2**	**79.6**	57.1	26.5	**52.3**	19515	4625
	2000	16.8	**58.3**	**83.8**	62.6	28.0	**60.4**	14515	3117
	4000	17.2	**74.2**	**86.2**	74.2	28.7	**74.2**	4515	903
Ru-En	100	2.5	**33.3**	12.6	**59.8**	4.2	**42.8**	19920	7150
	200	7.6	**38.2**	38.0	**59.4**	12.7	**46.5**	19420	6046
	500	14.1	**45.8**	**70.3**	59.6	23.5	**51.8**	17920	4666
	1000	16.0	**53.8**	**80.0**	59.8	26.7	**56.6**	15420	3430
	2000	16.8	**67.1**	**83.9**	67.1	28.0	**67.1**	10420	2084
	4000	17.1	**89.3**	85.7	**89.3**	28.5	**89.3**	420	84

Table 3: P@5, Recall@5, and F1@5 of Procrustes vs. SGM. "Total Hyps." = total number of hypotheses.

target→source language directions, resulting in two sets of translation hypotheses (one hypothesis per source word from each translation direction). For Procrustes, this extraction is done with CSLS. We intersect the hypotheses from the two directions, and feed the resulting set back to Procrustes [or SGM] as seeds. We abbreviate the iterative procedures as IterProc and IterSGM.

The general procedure is:

1. Run Procrustes [or SGM], forward direction.
2. Run Procrustes [or SGM], reverse direction.
3. Intersect the hypotheses from both directions.
4. Feed the hypotheses into step 1. Repeat.

Before step 1 for IterSGM, we form the graphs as described in Section 4. For IterProc, we combine the hypotheses in step 3 with the gold seeds, which is unncessary for SGM because seeds are always returned in the hypotheses.

How one select seeds in Step 4 for subsequent rounds is important. We try three variations:

Add-All Itersect hypotheses from forward and reverse directions. All become seeds for the next round, for N total rounds. Advantage: all correct pairs are passed to the next iteration. Disadvantage: all incorrect hypotheses are, too.

Stochastic-Add Add up to H new hypotheses each iteration; For iteration two, H random hypotheses from the intersection are chosen and added to the gold seeds for the forward direction. A separate random selection is taken for the reverse direction. The next round, $2H$ random hypotheses are chosen. This continues until all hypotheses are used.[6] This setting was designed to encourage robustness and improve accuracy by minimizing the number of erroneous seeds passed to subsequent rounds, to allow for recovery from mistakes. As distinct subsets are passed to forward and reverse directions, we encourage solutions of the runs to also be different, increasing output diversity. When intersecting the hypotheses, we aim to select pairs which are most likely to be correct—having two different solutions agree increases confidence that the induced pairs are correct, and selecting only a small subset allows recovery from mistakes. This is particularly important for SGM, where incorrect seeds are repeated in the output. In passing a subset to the next round, some incorrect pairs are dropped, and the model gets another chance to induce translations with a (presumably) stronger model. The stochasticity builds in robustness.

Active-Learning Seeds may also be added in an active learning fashion ("human-in-the-loop"). To simulate a human judging hypothesis quality, we use the union of hypotheses from forward and reverse directions and pass only correct hypotheses as seeds for the next iteration.

For Add-All and active learning experiments, we run for ten iterations ($N = 10$). For Stochastic-Add, $H = 100$. Tuning H is for future work.

[6] If not enough seeds in the intersection, all are used.

	Seeds	Add-All		Stochastic-Add		Active-Learning	
		IterProc	IterSGM	IterProc	IterSGM	IterProc	IterSGM
En-De	100	**61.3**	47.2	**62.1** *(+0.8)*	50.2 *(+3.0)*	**66.1** *(+4.8)*	56.6 *(+9.4)*
	200	**61.5**	48.2	**62.0** *(+0.5)*	50.8 *(+2.6)*	**66.3** *(+4.8)*	56.7 *(+8.5)*
	500	**62.6**	52.1	**62.8** *(+0.2)*	52.9 *(+0.8)*	**66.6** *(+4.0)*	58.3 *(+6.2)*
	1000	**63.0**	54.7	**63.5** *(+0.5)*	54.8 *(+0.1)*	**67.3** *(+4.3)*	59.5 *(+4.8)*
	2000	**65.2**	61.4	**65.2** *(+0.0)*	61.7 *(+0.3)*	**69.1** *(+3.9)*	65.6 *(+4.2)*
	4000	71.3	**74.2**	71.7 *(+0.4)*	**74.4** *(+0.2)*	74.6 *(+3.3)*	**75.4** *(+1.2)*
Ru-En	100	**62.4**	51.6	**62.7** *(+0.3)*	56.3 *(+4.7)*	**71.0** *(+8.6)*	62.5 *(+10.9)*
	200	**62.4**	53.7	**63.1** *(+0.7)*	56.4 *(+2.7)*	**71.1** *(+8.7)*	61.9 *(+8.2)*
	500	**63.7**	56.1	**63.7** *(+0.0)*	58.0 *(+1.9)*	**71.3** *(+7.6)*	63.1 *(+7.0)*
	1000	**64.0**	58.1	**64.0** *(+0.0)*	60.3 *(+2.2)*	**71.2** *(+7.2)*	66.4 *(+8.3)*
	2000	66.1	**67.1**	65.7 *(-0.4)*	**68.2** *(+1.1)*	**72.3** *(+6.2)*	71.0 *(+3.9)*
	4000	69.0	**89.3**	69.0 *(+0.0)*	**89.3** *(+0.0)*	72.6 *(+3.6)*	**89.3** *(+0.0)*

Table 4: P@1 of IterProc vs. IterSGM. Add-All runs for 10 iterations, seeding subsequent iterations with the intersection of hypotheses from forward and reverse directions. For Stochastic-Add, seeds are fed in up to 100 at a time until all are used. In parentheses is the improvement over Add-All.

5.2 Results

Results for IterProc and IterSGM are in Table 4. In parentheses is the raw improvement over Add-All. Unlike the single runs of Procrustes and SGM from Table 2, IterProc outperforms IterSGM in all scenarios with 1000 or less seeds, and for En-De with 2000 seeds. Stochastic-Add outperforms Add-All in nearly all experiments. Because SGM is more sensitive to input seeds than Procrustes, it particularly benefits from the stochastic setup which minimizes its exposure to incorrect input seeds and allows recovery from mistakes. Both IterProc and IterSGM benefit from active learning, showing the improved performance that may be achieved from human-in-the-loop.

Figure 3 has p@1 for IterProc vs. IterSGM during training (En-De, 100 seeds). Each data point has the number of hypotheses in the intersection of forward and reverse runs, and the precision of the intersection [Precision (Hyps.)]. IterProc dramatically underperforms SGM initially but quickly recovers. IterSGM stays roughly consistent throughout iterations. Precision of IterProc rapidly improves, but stays roughly the same for IterSGM. The number of hypotheses in the intersection is smaller for IterProc, suggesting that forward and reverse directions disagree more, but the hypotheses that they do agree upon are more precise.

The results in this section and the previous suggest that Procrustes and SGM have complementary strengths. While a single run of Procrustes struggles to align word embedding spaces with little supervision, it recovers when run iteratively. Conversely, one run of SGM dramatically outperforms one run of Procrustes with low number of seeds but does not improve much with iterations.

6 System Combination Experiments

6.1 Method

We create a combined system to see whether both methods together can outperform either alone, shown in Figure 4. For simplicity of implementation, we use Add-All IterProc and single runs of SGM. Here, Procrustes and SGM feed off one another to iteratively improve the solution. The combined system is cyclic—one may choose where to begin and end, with differing effect. There are two main components and a hypothesis extraction step:

A. **SGM** Run in forward and reverse directions. Intersect hypotheses and pass to next step.
B. **IterProc** Run for I_{proc} iterations.
C. **Hypothesis Extraction** Pull translation pairs from a forward run of SGM (-PullSGM) or IterProc (-PullProc). This results in one hypothesis for each source word.

We set $I_{\text{proc}} = 5$ and $N = 10$. We start from IterProc and pull results either from IterProc (Start: IterProc -PullProc) or SGM (Start: IterProc -PullSGM) on the final loop of the cycle. We repeat the experiments starting from SGM (Start: SGM).

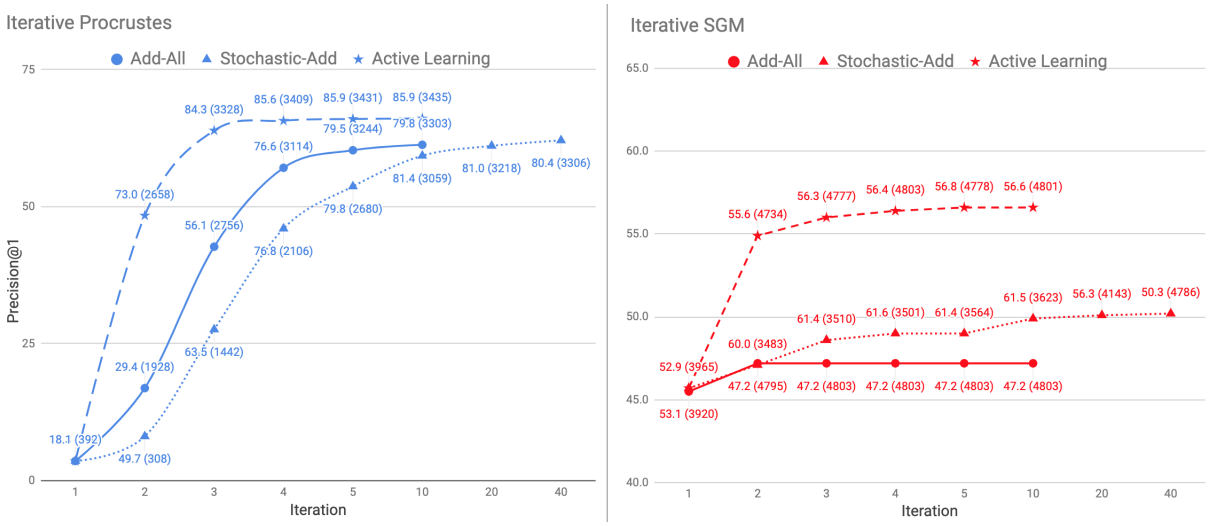


Figure 3: P@1 of iterative methods, by iteration (En-De, 100 seeds). Y-axis has p@1 for the forward run of Procrustes [SGM]. Seeds for subsequent iterations are drawn from the intersection of forward and reverse runs. Size and precision of this intersection is labeled above each point as "Precision (Num Hyps)". Add-All adds all hypotheses in the intersection as seeds to the next iteration. Stochastic-Add adds random samples of up to 100 new hypotheses per iteration. Active Learning adds all **correct** hypotheses.

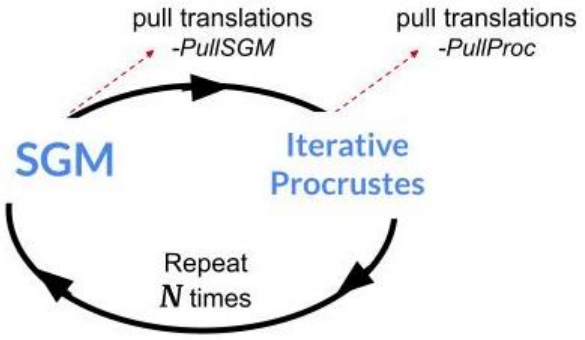


Figure 4: Combined cyclic system. SGM and Add-All IterProc are interspersed. Begin and end anywhere on the cycle. (1) Run SGM [or IterProc] in forward/reverse direction. (2) Intersect hypotheses and pass to forward/reverse IterProc [SGM] as seeds. (3) Pull final translations after Nth cycle from forward SGM [IterProc].

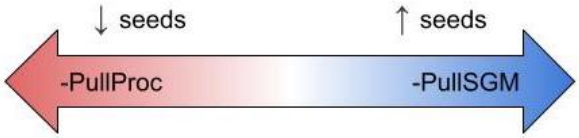


Figure 5: Whether pulling from IterProc or SGM is preferred in the combined cyclic system depends on the number of seeds, with IterProc preferred with a low number of seeds, and SGM preferred with more.

6.2 Results

Results for the combined cyclic system are in Table 5. The "Previous Best" column has the best performance from previous experiments (excluding active learning). For all seed levels, the cyclic system can equal or outperform the previous best performance from earlier experiments with single and iterative Procrustes or SGM.

Looking down the "-PullProc" columns, we discover that it hardly matters whether we begin the cycle with IterProc or SGM. The same is true for "-PullSGM". Whether -PullProc or -PullSGM is preferred appears to be on a continuum, depicted in Figure 5. For a low seed count, -PullProc is preferred, with the effect more pronounced as seed size diminishes. Conversely, -PullSGM is increasingly preferred as seed set size increases.

7 Discussion & Future Directions

Though much work in BLI takes a Euclidean view and elicits solutions via methods such as solutions to the generalized Procrustes problem, BLI may also be viewed as a graph-based problem. Parts of the graph-based view have appeared in existing work, but no one has yet to compare the different framings in the context of BLI. We perform this analysis for two high-resource language pairs with well-trained embeddings from Wikipedia, in a restricted data context. Under our experimental settings, we find that:

1. Procrustes-based methods and SGM behave differently under differing contexts (namely,

	Seeds	Prev. Best	Combination Methods -PullProc *Start: IterProc*	-PullProc *Start: SGM*	-PullSGM *Start: IterProc*	-PullSGM *Start: SGM*
En-De	100	62.1	**62.2**	62.1	59.7	59.5
	200	62.0	**62.8**	62.6	60.4	60.4
	500	62.8	63.5	**63.8**	62.1	62.0
	1000	63.5	63.9	**64.2**	63.0	63.7
	2000	65.2	66.7	66.7	**69.7**	69.0
	4000	74.4	73.2	73.2	**79.7**	79.2
Ru-En	100	62.7	63.9	**64.0**	61.7	62.0
	200	63.1	**64.5**	64.3	62.6	63.1
	500	63.7	**65.3**	**65.3**	64.0	64.3
	1000	64.0	**66.8**	66.4	**66.8**	66.4
	2000	68.2	69.4	69.5	72.9	**73.1**
	4000	**89.3**	77.4	77.4	**89.3**	**89.3**

Table 5: P@1 for combined cyclic method (Figure 4. One may begin from either IterProc ("Start: Procrustes") or SGM ("Start: SGM"), and may pull final hypotheses from either Procrustes ("-PullProc") or SGM ("-PullSGM"). "Prev. Best" is best result from previous experiments (excluding active learning). Bold is best overall.

the amount of available seeds), so either may be favorable given the specific data context. SGM appears favorable with less seeds.

2. SGM can be run iteratively, but does not improve as rapidly as Iterative Procrustes. Both benefit from stochasticity, and active learning can provide strong improvement.
3. Procrustes and SGM can be effectively combined to outperform either alone.

There are limitations to our work which should be addressed by future analyses. We use clean, well-trained embeddings from the same domain. Previous work has shown Procrustes to struggle with poorly-trained and low-resource word embedding spaces, and for well-trained embeddings in mismatched domains (Marchisio et al., 2020). In these cases, SGM might benefit from a different distance metric. A detailed analysis should also be performed when data is many-to-many, as translation is naturally a many-to-many task. It may also be enlightening to revisit word vectors based on co-occurrence statistics. The size of training and test sets should be increased, as the presence of more synonyms/antonyms and other "distractor" words may elicit different behavior. There are computational considerations as we scale-up, particularly for SGM.

8 Related Work

Matching words using vector representations began with vectors based on co-occurrence statistics. Rapp (1995) and Fung (1995) induce bilingual lexica based on the principle that words that frequently co-occur in one language have translations that co-occur frequently in another. Diab and Finch (2000) extend this by measuring similarity between words based on co-occurrence vectors and matching words across language by preserving these similarities. Mikolov et al. (2013) are the first to perform BLI over word embeddings, estimating the transformation matrix using stochastic gradient descent. Most recent work solves a variation of the generalized Procrustes problem (e.g., Conneau et al., 2018; Artetxe et al., 2016b, 2017; Patra et al., 2019; Artetxe et al., 2018b; Doval et al., 2018; Joulin et al., 2018; Jawanpuria et al., 2019; Alvarez-Melis and Jaakkola, 2018). Zhang et al. (2020) learn a mapping that overfits to training pairs thus enforcing "hard-seeding", while Ruder et al. (2018) enforce a one-to-one constraint on the output for BLI.

Some BLI work uses graph based methods implicitly or explicitly. Artetxe et al. (2018a) form an initial solution with similarity matrices and refine with iterative Procrustes. Grave et al. (2019b) optimize "Procrustes in Wasserstein Distance", employing a quadratic assignment formulation and the Frank-Wolfe method. Ren et al. (2020) form CSLS

similarity matrices, iteratively extract cliques, and map with Procrustes. Gutierrez-Vasques and Mijangos (2017) create a weighted graph of translation candidates then create word vectors with Node2Vec (Grover and Leskovec, 2016). Wushouer et al. (2013) use graphs for a source, target, and pivot language to iteratively extract translation pairs based on heuristics. Our active learning approach is inspired by Yuan et al. (2020).

9 Conclusion

We perform the first detailed analysis of the consequences of framing BLI either as a Euclidean problem solved by the common Procrustes solution with nearest-neighbor search, or as a graph-based matching problem solved with SGM. We show that each performs differently under different data contexts, with SGM preferred with low amounts of seeds. We compare iterative versions of SGM and Procrustes, and find that stochasticity benefits both. Finally, we create a combined system that outperforms individual Procrustes and SGM approaches.

Acknowledgements

We thank our anonymous reviewers for their comments. This work was supported in part by the US Defense Advanced Research Projects Agency under the D3M program administered through contract FA8750-17-2-0112.

References

David Alvarez-Melis and Tommi Jaakkola. 2018. Gromov-Wasserstein alignment of word embedding spaces. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1881–1890, Brussels, Belgium. Association for Computational Linguistics.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2016a. Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2289–2294.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2016b. Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2289–2294, Austin, Texas. Association for Computational Linguistics.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2017. Learning bilingual word embeddings with (almost) no bilingual data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 451–462, Vancouver, Canada. Association for Computational Linguistics.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2018a. A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 789–798.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2018b. A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 789–798, Melbourne, Australia. Association for Computational Linguistics.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2019. An effective approach to unsupervised machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 194–203, Florence, Italy. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 1247–1250, New York, NY, USA. Association for Computing Machinery.

Jaewon Chung, Benjamin D Pedigo, Eric W Bridgeford, Bijan K Varjavand, Hayden S Helm, and Joshua T Vogelstein. 2019. Graspy: Graph statistics in python. *Journal of Machine Learning Research*, 20(158):1–7.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2018. Word translation without parallel data. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Mona Diab and Steven Finch. 2000. A statistical word-level translation model for comparable corpora. In *Proceedings of the Conference on Content-based multimedia information access (RIAO)*.

Yerai Doval, Jose Camacho-Collados, Luis Espinosa Anke, and Steven Schockaert. 2018. Improving cross-lingual word embeddings by meeting in the middle. In *Proceedings of the 2018 Conference on*

Empirical Methods in Natural Language Processing, pages 294–304.

Donniell E Fishkind, Sancar Adali, Heather G Patsolic, Lingyao Meng, Digvijay Singh, Vince Lyzinski, and Carey E Priebe. 2019. Seeded graph matching. *Pattern recognition*, 87:203–215.

Marguerite Frank, Philip Wolfe, et al. 1956. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110.

Yoshinari Fujinuma, Jordan Boyd-Graber, and Michael J. Paul. 2019. A resource-free evaluation metric for cross-lingual word embeddings based on graph modularity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4952–4962, Florence, Italy. Association for Computational Linguistics.

Pascale Fung. 1995. Compiling bilingual lexicon entries from a non-parallel English-Chinese corpus. In *Third Workshop on Very Large Corpora*.

Goran Glavaš, Robert Litschko, Sebastian Ruder, and Ivan Vulić. 2019. How to (properly) evaluate cross-lingual word embeddings: On strong baselines, comparative analyses, and some misconceptions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 710–721, Florence, Italy. Association for Computational Linguistics.

Edouard Grave, Armand Joulin, and Quentin Berthet. 2019a. Unsupervised alignment of embeddings with wasserstein procrustes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1880–1890. PMLR.

Edouard Grave, Sainbayar Sukhbaatar, Piotr Bojanowski, and Armand Joulin. 2019b. Training hybrid language models by marginalizing over segmentations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1477–1482, Florence, Italy. Association for Computational Linguistics.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

Ximena Gutierrez-Vasques and Victor Mijangos. 2017. Low-resource bilingual lexicon extraction using graph based word embeddings. *arXiv preprint arXiv:1710.02569*.

Aria Haghighi, Percy Liang, Taylor Berg-Kirkpatrick, and Dan Klein. 2008. Learning bilingual lexicons from monolingual corpora. In *Proceedings of ACL-08: HLT*, pages 771–779, Columbus, Ohio. Association for Computational Linguistics.

Pratik Jawanpuria, Arjun Balgovind, Anoop Kunchukuttan, and Bamdev Mishra. 2019. Learning multilingual word embeddings in latent metric space: a geometric approach. *Transactions of the Association for Computational Linguistics*, 7:107–120.

Armand Joulin, Piotr Bojanowski, Tomáš Mikolov, Hervé Jégou, and Édouard Grave. 2018. Loss in translation: Learning bilingual word mapping with a retrieval criterion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2979–2984.

Harold W Kuhn. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Kelly Marchisio, Kevin Duh, and Philipp Koehn. 2020. When does unsupervised machine translation work? In *Proceedings of the Fifth Conference on Machine Translation*, pages 571–583, Online. Association for Computational Linguistics.

Rada Mihalcea and Dragomir Radev. 2011. *Graph-based natural language processing and information retrieval*. Cambridge university press.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Tasnim Mohiuddin, M Saiful Bari, and Shafiq Joty. 2020. LNMap: Departures from isomorphic assumption in bilingual lexicon induction through non-linear mapping in latent space. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2712–2723, Online. Association for Computational Linguistics.

Ndapa Nakashole and Raphael Flauger. 2018. Characterizing departures from linearity in word translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 221–227, Melbourne, Australia. Association for Computational Linguistics.

Vivi Nastase, Rada Mihalcea, and Dragomir R Radev. 2015. A survey of graphs in natural language processing. *Natural Language Engineering*, 21(5):665–698.

Aitor Ormazabal, Mikel Artetxe, Gorka Labaka, Aitor Soroa, and Eneko Agirre. 2019. Analyzing the limitations of cross-lingual word embedding mappings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4990–4995, Florence, Italy. Association for Computational Linguistics.

Barun Patra, Joel Ruben Antony Moniz, Sarthak Garg, Matthew R. Gormley, and Graham Neubig. 2019. Bilingual lexicon induction with semi-supervision in non-isometric embedding spaces. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 184–193, Florence, Italy. Association for Computational Linguistics.

Milos Radovanovic, Alexandros Nanopoulos, and Mirjana Ivanovic. 2010. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research*, 11(sept):2487–2531.

Reinhard Rapp. 1995. Identifying word translations in non-parallel texts. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, ACL '95, page 320–322, USA. Association for Computational Linguistics.

Shuo Ren, Shujie Liu, Ming Zhou, and Shuai Ma. 2020. A graph-based coarse-to-fine method for unsupervised bilingual lexicon induction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3476–3485, Online. Association for Computational Linguistics.

Sebastian Ruder, Ryan Cotterell, Yova Kementchedjhieva, and Anders Søgaard. 2018. A discriminative latent-variable model for bilingual lexicon induction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 458–468, Brussels, Belgium. Association for Computational Linguistics.

Sartaj Sahni and Teofilo Gonzalez. 1976. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565.

Peter H Schönemann. 1966. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10.

Anders Søgaard, Sebastian Ruder, and Ivan Vulić. 2018. On the limitations of unsupervised bilingual dictionary induction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 778–788, Melbourne, Australia. Association for Computational Linguistics.

Ikumi Suzuki, Kazuo Hara, Masashi Shimbo, Marco Saerens, and Kenji Fukumizu. 2013. Centering similarity measures to reduce hubs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 613–623, Seattle, Washington, USA. Association for Computational Linguistics.

Joshua T Vogelstein, John M Conroy, Vince Lyzinski, Louis J Podrazik, Steven G Kratzer, Eric T Harley, Donniell E Fishkind, R Jacob Vogelstein, and Carey E Priebe. 2015. Fast approximate quadratic programming for graph matching. *PLOS one*, 10(4):e0121002.

Ivan Vulić, Goran Glavaš, Roi Reichart, and Anna Korhonen. 2019. Do we really need fully unsupervised cross-lingual embeddings? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4407–4418, Hong Kong, China. Association for Computational Linguistics.

Mairidan Wushouer, Toru Ishida, Katsutoshi Hirayama, and Donghui Lin. 2013. Inducing bilingual lexicon using pivot language. *Annual Conference of the Information Processing Society of Japan (IPSJ)*, 5:6.

Michelle Yuan, Mozhi Zhang, Benjamin Van Durme, Leah Findlater, and Jordan Boyd-Graber. 2020. Interactive refinement of cross-lingual word embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5984–5996, Online. Association for Computational Linguistics.

Mozhi Zhang, Yoshinari Fujinuma, Michael J. Paul, and Jordan Boyd-Graber. 2020. Why overfitting isn't always bad: Retrofitting cross-lingual word embeddings to dictionaries. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2214–2220, Online. Association for Computational Linguistics.

Mozhi Zhang, Keyulu Xu, Ken-ichi Kawarabayashi, Stefanie Jegelka, and Jordan Boyd-Graber. 2019. Are girls neko or shōjo? cross-lingual alignment of non-isomorphic embeddings with iterative normalization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3180–3189, Florence, Italy. Association for Computational Linguistics.

A Appendix for: An Analysis of Euclidean vs. Graph-Based Framing for Bilingual Lexicon Induction from Word Embedding Spaces

A.1 Mathematical Notation

$\oplus$ is the direct sum of matrices:

$$I_s \oplus P = \begin{bmatrix} I_s & 0 \\ 0 & P \end{bmatrix}$$

A.2 Simple Example of Seeded Graph Matching

Recall the constrained optimization objective for seeded graph matching:

$$\underset{P \in \Pi_{n-s}}{\arg\min} \|G_x - (I_s \oplus P) G_y (I_s \oplus P)^T\|_F^2$$

Let $x_1, x_2, x_3, x_4 \in X$ and $y_1, y_2, y_3, y_4 \in Y$. To more clearly see the effect, each of the vectors is orthogonal to all others but not unit length. We create the graphs as below, where each $G_{x_{ij}} = \langle x_i, x_j \rangle$ (equivalently for $G_{y_{ij}} \in G_y$). We take (x_1, y_1) as a seed.

$$G_x = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}, G_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

To minimize $G_x - G_y$, we swap y_2 to y_3, y_3 to y_4, and y_4 to y_2 using P as below:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Let $G'_y = (I_s \oplus P) G_y (I_s \oplus P)^T$:

$$G'_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

We note that this choice for G'_y (and therefore P) minimizes Equation 5. The solutions we extract as translation pairs from G_x and G_y are therefore $(x_1, y_1), (x_2, y_4), (x_3, y_2), (x_4, y_3)$.

A.3 Seeded Graph Matching

This section describes Seeded Graph Matching (Fishkind et al., 2019). Let $G_x, G_y \in \mathbb{R}^{n \times n}$ be graphs representing the relationships between words in word embedding spaces $X, Y \in \mathbb{R}^{n \times d}$, respectively. We use cosine similarity as the measure of distance when weighting the edges, and therefore the resulting graphs are undirected and symmetric. To form G_x, we may normalize the embeddings in X so that $G_x = XX^T$. We create G_y similarly.

Assume seeds $\{(x_1, y_1), (x_2, y_2), ...(x_s, y_s)\}$ are given. We formulate this constrained optimization problem as below:

$$\underset{P \in \Pi_{n-s}}{\arg\min} \|G_x - (I_s \oplus P) G_y (I_s \oplus P)^T\|_F^2 \quad (5)$$

We understand $(I_s \oplus P) G_y (I_s \oplus P)^T$ as the attempt to "move" the rows/columns of the graph G_y such that its rows/columns are in the same order as G_x, which is equivalent to relabeling the edges in G_y. Rows/columns in G_x and G_y after reordering that have the same index are then extracted as translations of one another.

We rearrange Equation 5 to be more tractable for optimization. Letting $G'_y = (I_s \oplus P) G_y (I_s \oplus P)^T$, we perform the below:[7]

$$\underset{P \in \Pi_{n-s}}{\arg\min} \langle G_x - G'_y, G_x - G'_y \rangle_F$$
$$= \underset{P \in \Pi_{n-s}}{\arg\min} ||G_x||_F^2 + ||G_y||_F^2 - 2 \cdot tr(G_x^T G'_y)$$
$$= \underset{P \in \Pi_{n-s}}{\arg\min} \; -2 \cdot tr(G_x^T G'_y)$$
$$= \underset{P \in \Pi_{n-s}}{\arg\max} \; tr(G_x^T (I_s \oplus P) G_y (I_s \oplus P)^T)$$

Because the original objective is non-convex, the constraint on P is relaxed to being a doubly-stochastic matrix[8] $P \in D_{n-s}$, which is the convex hull of the set of permutation matrices (Birkhoff-Von Neumann Theorem). The resulting optimization objective is thus:

$$= \underset{P \in D_{n-s}}{\arg\max} \; tr(G_x^T (I_s \oplus P) G_y (I_s \oplus P)^T) \quad (6)$$

[7] Recall the definition of norm: $||x|| = \sqrt{\langle x, x \rangle}$, properties of the inner product, the trace definition of Frobenius inner product whereby $\langle A, B \rangle_F = tr(B^T A)$, and the fact that a permutation matrix's transpose is its inverse.

[8] All rows/columns sum to 1.