Power-Law Distributions for Paraphrases Extracted from Bilingual Corpora

Spyros Martzoukos   Christof Monz
Informatics Institute, University of Amsterdam
Science Park 904, 1098 XH Amsterdam, The Netherlands
{s.martzoukos, c.monz}@uva.nl

Abstract

We describe a novel method that extracts paraphrases from a bitext, for both the source and target languages. In order to reduce the search space, we decompose the phrase-table into sub-phrase-tables and construct separate clusters for source and target phrases. We convert the clusters into graphs, add smoothing/syntactic-information-carrier vertices, and compute the similarity between phrases with a random walk-based measure, the *commute time*. The resulting phrase-paraphrase probabilities are built upon the conversion of the commute times into artificial co-occurrence counts with a novel technique. The co-occurrence count distribution belongs to the power-law family.

1 Introduction

Paraphrase extraction has emerged as an important problem in NLP. Currently, there exists an abundance of methods for extracting paraphrases from monolingual, comparable and bilingual corpora (Madnani and Dorr, 2010; Androutsopoulos and Malakasiotis, 2010); we focus on the latter and specifically on the phrase-table that is extracted from a bitext during the training stage of Statistical Machine Translation (SMT). Bannard and Callison-Burch (2005) introduced the *pivoting* approach, which relies on a 2-step transition from a phrase, via its translations, to a paraphrase candidate. By incorporating the syntactic structure of phrases (Callison-Burch, 2005), the quality of the paraphrases extracted with pivoting can be improved. Kok and Brockett (2010) (henceforth KB) used a random walk framework to determine the similarity between phrases, which was shown to outperform pivoting with syntactic information, when multiple phrase-tables are used. In SMT, extracted paraphrases with associated pivot-based (Callison-Burch et al., 2006; Onishi et al., 2010) and cluster-based (Kuhn et al., 2010) probabilities have been found to improve the quality of translation. Pivoting has also been employed in the extraction of syntactic paraphrases, which are a mixture of phrases and non-terminals (Zhao et al., 2008; Ganitkevitch et al., 2011).

We develop a method for extracting paraphrases from a bitext for both the source and target languages. Emphasis is placed on the quality of the phrase-paraphrase probabilities as well as on providing a stepping stone for extracting syntactic paraphrases with equally reliable probabilities. In line with previous work, our method depends on the connectivity of the phrase-table, but the resulting construction treats each side separately, which can potentially be benefited from additional monolingual data.

The initial problem in harvesting paraphrases from a phrase-table is the identification of the search space. Previous work has relied on breadth first search from the query phrase with a depth of 2 (pivoting) and 6 (KB). The former can be too restrictive and the latter can lead to excessive noise contamination when taking shallow syntactic information features into account. Instead, we choose to cluster the phrase-table into separate source and target clusters and in order to make this task computationally feasible, we decompose the phrase-table into sub-phrase-tables. We propose a novel heuristic algorithm for the decomposition of the phrase-table (Section 2.1), and use a well-established co-clustering algorithm for clustering each sub-phrase-table (Section 2.2).

The underlying connectivity of the source and target clusters gives rise to a natural graph representation for each cluster (Section 3.1). The vertices of the graphs consist of phrases and features with a dual smoothing/syntactic-information-carrier role. The latter allow (a) re-distribution of the mass for phrases with no appropriate paraphrases and (b) the extraction of syntactic paraphrases. The proximity among vertices of a graph is measured by means of a random walk distance measure, the *commute time* (Aldous and Fill, 2001). This measure is known to perform well in identifying similar words on the graph of WordNet (Rao et al., 2008) and a related measure, the *hitting time* is known to perform well in harvesting paraphrases on a graph constructed from multiple phrase-tables (KB).

Generally in NLP, power-law distributions are typically encountered in the collection of counts during the training stage. The distances of Section 3.1 are converted into artificial co-occurrence counts with a novel technique (Section 3.2). Although they need not be integers, the main challenge is the type of the underlying distributions; it should ideally emulate the resulting count distributions from the phrase extraction stage of a monolingual parallel corpus (Dolan et al., 2004). These counts give rise to the desired probability distributions by means of relative frequencies.

2 Sub-phrase-tables & Clustering

2.1 Extracting Connected Components

For the decomposition of the phrase-table into sub-phrase-tables it is convenient to view the phrase-table as an undirected, unweighted graph P with the vertex set being the source and target phrases and the edge set being the phrase-table entries. For the rest of this section, we do not distinguish between source and target phrases, i.e. both types are treated equally as vertices of P. When referring to the size of a graph, we mean the number of vertices it contains.

A trivial initial decomposition of P is achieved by identifying all its *connected components* (components for brevity), i.e. the mutually disjoint connected subgraphs, $\{P_0, P_1, ..., P_n\}$. It turns out (see Section 4.1) that the largest component, say P_0, is of significant size. We call P_0 *giant* and it needs to be further decomposed. This is done by identifying all vertices such that, upon removal, the component becomes disconnected. Such vertices are called *articulation points* or *cut-vertices*. Cut-vertices of high connectivity degree are removed from the giant component (see Section 4.1). For the remaining vertices of the giant component, new components are identified and we proceed iteratively, while keeping track of the cut-vertices that are removed at each iteration, until the size of the largest component is less than a certain threshold θ (see Section 4.1).

Note that at each iteration, when removing cut-vertices from a giant component, the resulting collection of components may include graphs consisting of a single vertex. We refer to such vertices as *residues*. They are excluded from the resulting collection and are considered for separate treatment, as explained later in this section.

The cut-vertices need to be inserted appropriately back to the components: Starting from the last iteration step, the respective cut-vertices are added to all the components of P_0 which they used to 'glue' together; this process is performed iteratively, until there are no more cut-vertices to add. By 'addition' of a cut-vertex to a component, we mean the re-establishment of edges between the former and other vertices of the latter. The result is a collection of components whose total number of unique vertices is less than the number of vertices of the initial giant component P_0.

These remaining vertices are the residues. We then construct the graph R which consists of the residues together with *all* their translations (even those that are included in components of the above collection) and then identify its components $\{R_0, ..., R_m\}$. It turns out, that the largest component, say R_0, is giant and we repeat the decomposition process that was performed on P_0. This results in a new collection of components as well as new residues: The components need to be pruned (see Section 4.1) and the residues give rise to a new graph R' which is constructed in the same way as R. We proceed iteratively until the number of residues stops changing. For each remaining residue u, we identify its translations, and for each translation v we identify the largest component of which v is a member and add u to that component.

The final result is a collection $\mathcal{C} = \mathcal{D} \cup \mathcal{F}$, where $\mathcal{D}$ is the collection of components emerging from the entire iterative decomposition of P_0

and R, and $\mathcal{F} = \{P_1, ..., P_n\}$. Figure 1 shows the decomposition of a connected graph G_0; for simplicity we assume that only one cut-vertex is removed at each iteration and ties are resolved arbitrarily. In Figure 2 the residue graph is constructed and its two components are identified. The iterative insertion of the cut vertices is also depicted. The resulting two components together with those from R form the collection $\mathcal{D}$ for G_0.

The addition of cut-vertices into multiple components, as well as the construction method of the residue-based graph R, can yield the occurrences of a vertex in multiple components in $\mathcal{D}$. We exploit this property in two ways:

(a) In order to mitigate the risk of excessive decomposition (which implies greater risk of good paraphrases being in different components), as well as to reduce the size of $\mathcal{D}$, a conservative merging algorithm of components is employed. Suppose that the elements of $\mathcal{D}$ are ranked according to size in ascending order as $\mathcal{D} = \{D_1, ..., D_k, D_{k+1}, ..., D_{|\mathcal{D}|}\}$, where $|D_i| \leq \delta$, for $i = 1, ..., k$, and some threshold δ (see Section 4.1). Each component D_i with $i \in \{1, ..., k\}$ is examined as follows: For each vertex of D_i the number of its occurrences in $\mathcal{D}$ is inspected; this is done in order to identify an appropriate vertex b to act as a bridge between D_i and other components of which b is a member. Note that translations of a vertex b with smaller number of occurrences in $\mathcal{D}$ are less likely to capture their full spectrum of paraphrases. We thus choose a vertex b from D_i with the smallest number of occurrences in $\mathcal{D}$, resolving ties arbitrarily, and proceed with merging D_i with the largest component, say D_j with $j \in \{1, ..., |\mathcal{D}| - 1\}$, of which b is also a member. The resulting merged component $D_{j'}$ contains all vertices and edges of D_i and D_j and new edges, which are formed according to the rule: if u is a vertex of D_i and v is a vertex of D_j and (u, v) is a phrase-table entry, then (u, v) is an edge in $D_{j'}$. As long as no connected component has identified D_i as the component with which it should be merged, then D_i is deleted from the collection $\mathcal{D}$.

(b) We define an *idf*-inspired measure for each phrase pair (x, x') of the same type (source or target) as

$$idf(x, x') = \frac{1}{\log |\mathcal{D}|} \log \left(\frac{2c(x, x')|\mathcal{D}|}{c(x) + c(x')} \right), \quad (1)$$

where $c(x, x')$ is the number of components in which the phrases x and x' co-occur, and equivalently for $c(\cdot)$. The purpose of this measure is for pruning paraphrase candidates and its use is explained in Section 3.1. Note that $idf(x, x') \in [0, 1]$.

The merging process and the *idf* measure are irrelevant for phrases belonging to the components of $\mathcal{F}$, since the vertex set of each component of $\mathcal{F}$ is mutually disjoint with the vertex set of any other component in $\mathcal{C}$.

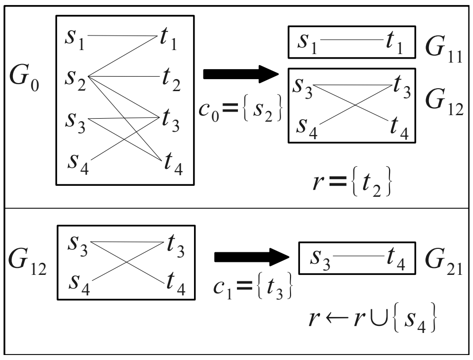


Figure 1: The decomposition of G_0 with vertices s_i and t_j: The cut-vertex of the ith iteration is denoted by c_i, and r collects the residues after each iteration. The task is completed in Figure 2.

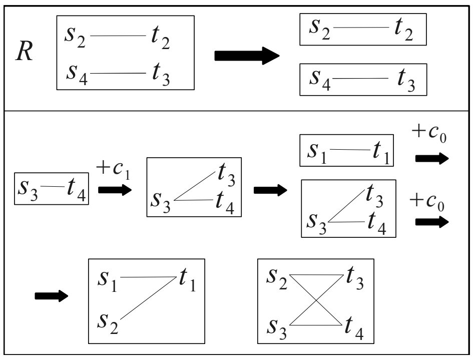


Figure 2: Top: Residue graph with its components (no further decomposition is required). Bottom: Adding cut-vertices back to their components.

2.2 Clustering Connected Components

The aim of this subsection is to generate separate clusters for the source and target phrases of each sub-phrase-table (component) $C \in \mathcal{C}$. For this purpose the Information-Theoretic Co-Clustering (ITC) algorithm (Dhillon et al., 2003) is employed, which is a general principled clustering algorithm that generates *hard* clusters (i.e. every element belongs to exactly one cluster) of two interdependent quantities and is known to perform well on high-dimensional and sparse data. In our case, the interdependent quantities are the source and target phrases and the sparse data is the phrase-table.

ITC is a search algorithm similar to K-means, in the sense that a cost function, is minimized at each iteration step and the number of clusters for both quantities are meta-parameters. The number of clusters is set to the most conservative initialization for both source and target phrases, namely to as many clusters as there are phrases. At each iteration, new clusters are constructed based on the identification of the argmin of the cost function for each phrase, which gradually reduces the number of clusters.

We observe that conservative choices for the meta-parameters often result in good paraphrases being in different clusters. To overcome this problem, the hard clusters are converted into soft (i.e. an element may belong to several clusters): One step before the stopping criterion is met, we modify the algorithm so that instead of assigning a phrase to the cluster with the smallest cost we select the bottom-X clusters ranked by cost. Additionally, only a certain number of phrases is chosen for soft clustering. Both selections are done conservatively with criteria based on the properties of the cost functions.

The formation of clusters leads to a natural refinement of the *idf* measure defined in eqn. (1): The quantity $c(x, x')$ is redefined as the number of components in which the phrases x and x' co-occur in at least one cluster.

3 Monolingual Graphs & Counts

We proceed with converting the clusters into directed, weighted graphs and then extract paraphrases for both the source and target side. For brevity we explain the process restricted to the source clusters of a sub-phrase-table, but the same method applies for the target side and for all sub-phrase-tables in the collection $\mathcal{C}$.

3.1 Monolingual graphs

Each source cluster is converted into a graph G as follows: The vertex set consists of the phrases of the cluster and an edge between s and s' exists, if (a) s and s' have at least one translation from the same target cluster, and (b) $idf(s, s')$ is greater than some threshold σ (see Section 4.1). If two phrases that satisfy condition (b) and have translations in more than one common target cluster, a distinct such edge is established. All edges are bi-directional with distinct weights for both directions.

Figure 3 depicts an example of such a construction; a link between a phrase s_i and a target cluster implies the existence of at least one translation for s_i in that cluster. We are not interested in the target phrases and they are thus not shown. For simplicity we assume that condition (b) is always satisfied and the extracted graph contains the maximum possible edges. Observe that phrases s_3 and s_4 have two edges connecting them, (due to target clusters T_c and T_d) and that the target cluster T_a is irrelevant to the construction of the graph, since s_1 is the only phrase with translations in it. This conversion of a source cluster into a graph G

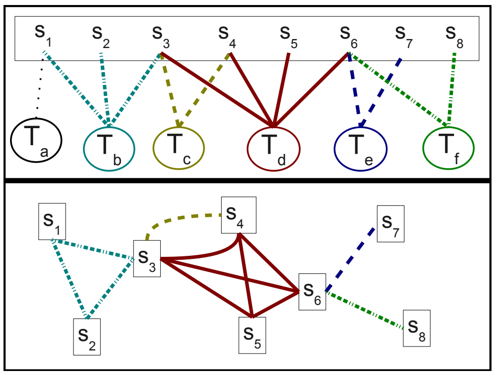


Figure 3: Top: A source cluster containing phrases s_1,..., s_8 and the associated target clusters T_a,..., T_f. Bottom: The extracted graph from the source cluster. All edges are bi-directional.

results in the formation of subgraphs in G, where each subgraph is generated by a target cluster. In general, if condition (b) is not always satisfied, then G need not be connected and each connected component is treated as a distinct graph.

Analogous to KB, we introduce *feature* vertices to G: For each phrase vertex s, its part-of-speech (POS) tag sequence and stem sequence are identified and inserted into G as new vertices with bi-directional weighted edges connected to s. If phrase vertices s and s' have the same POS tag sequence, then they are connected to the same POS tag feature vertex. Similarly for stem feature vertices. See Figure 4 for an example. Note that we do not allow edges between POS tag and stem fea-

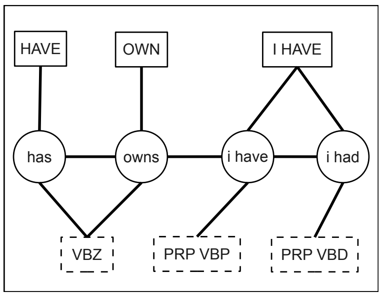


Figure 4: Adding feature vertices to the extracted graph (`has`) $\rightleftharpoons$ (`owns`) $\rightleftharpoons$ (`i have`) $\rightleftharpoons$ (`i had`). Phrase, POS tag feature and stem feature vertices are drawn in circles, dotted rectangles and solid rectangles respectively. All edges are bidirectional.

ture vertices. The purpose of the feature vertices, unlike KB, is primarily for smoothing and secondarily for identifying paraphrases with the same syntactic information and this will become clear in the description of the computation of weights.

The set of all phrase vertices that are adjacent to s is written as $\Gamma(s)$, and referred to as the *neighborhood* of s. Let $n(s,t)$ denote the co-occurrence count of a phrase-table entry (s,t) (Koehn, 2009). We define the strength of s in the subgraph generated by cluster T as

$$n(s;T) = \sum_{t \in T} n(s,t), \tag{2}$$

which is simply a partial occurrence count for s. We proceed with computing weights for all edges of G:

Phrase$\rightleftharpoons$phrase weights: Inspired by the notion of *preferential attachment* (Yule, 1925), which is known to produce power-law weight distributions for evolving weighted networks (Barrat et al., 2004), we set the weight of a directed edge from s to s' to be proportional to the strengths of s' in all subgraphs in which both s and s' are members. Thus, in the random walk framework, s is more likely to visit a stronger (more reliable) neighbor. If $T_{s,s'} = \{T \,|s \text{ and } s' \text{ coexist in subgraph generated by } T\}$, then the weight $w(s \to s')$ of the directed edge from s to s' is given by

$$w(s \to s') = \sum_{T \in T_{s,s'}} n(s';T), \tag{3}$$

if $s' \in \Gamma(s)$ and 0 otherwise.

Phrase$\rightleftharpoons$feature weights: As mentioned above, feature vertices have the dual role of carrying syntactic information and smoothing. From eqn. (3) it can be deduced that, if for a phrase s, the amount of its outgoing weights is close to the amount of its incoming weights, then this is an indication that at least a significant part of its neighborhood is reliable; the larger the strengths, the more certain the indication. Otherwise, either s or a significant part of its neighborhood is unreliable. The amount of weight from s to its feature vertices should depend on this observation and we thus let

$$\text{net}(s) = \left| \sum_{s' \in \Gamma(s)} (w(s \to s') - w(s' \to s)) \right| + \epsilon, \tag{4}$$

where ϵ prevents net(s) from becoming 0 (see Section 4.1). The net weight of a phrase vertex s is distributed over its feature vertices as

$$w(s \to f_X) = < w(s \to s') > + \text{net}(s), \tag{5}$$

where the first summand is the average weight from s to its neighboring phrase vertices and $X = \text{POS}, \text{STEM}$. If s has multiple POS tag sequences, we distribute the weight of eqn. (5) relatively to the co-occurrences of s with the respective POS tag feature vertices. The quantity $< w(s \to s') >$ accounts for the basic smoothing and is augmented by a value net(s) that measures the reliability of s's neighborhood; the more unreliable the neighborhood, the larger the net weight and thus larger the overall weights to the feature vertices.

The choice for the opposite direction is trivial:

$$w(f_X \to s) = \frac{1}{|\{s' : (f_X, s') \text{ is an edge }\}|}, \tag{6}$$

where $X = \text{POS}, \text{STEM}$. Note the effect of eqns. (4)–(6) in the case where the neighborhood of s has unreliable strengths: In a random walk the feature vertices of s will be preferred and the resulting similarities between s and other phrase vertices will be small, as desired. Nonetheless, if the syntactic information is the same with any other phrase vertex in G, then the paraphrases will be captured.

The transition probability from *any* vertex u to *any* other vertex v in G, i.e., the probability of hopping from u to v in one step, is given by

$$p(u \to v) = \frac{w(u \to v)}{\sum_{v'} w(u \to v')}, \quad (7)$$

where we sum over all vertices adjacent to u in G. We can thus compute the similarity between *any* two vertices u and v in G by their commute time, i.e., the expected number of steps in a round trip, in a random walk from u to v and then back to u, which is denoted by $\kappa(u, v)$ (see Section 4.1 for the method of computation of κ). Since $\kappa(u, v)$ is a distance measure, the smaller its value, the more similar u and v are.

3.2 Counts

We convert the distance $\kappa(u, v)$ of a vertex pair u, v in a graph G into a co-occurrence count $n_G(u, v)$ with a novel technique: In order to assess the quality of the pair u, v with respect to G we compare $\kappa(u, v)$ with $\kappa(u, x)$ and $\kappa(v, x)$ for all other vertices x in G. We thus consider the average distance of u with the other vertices of G other than v, and similarly for v. This quantity is denoted by $\kappa(u; v)$ and $\kappa(v; u)$ respectively, and by definition it is given by

$$\kappa(i; j) = \sum_{\substack{x \in G \\ x \neq j}} \kappa(i, x) p_G(x|i) \quad (8)$$

where $p_G(x|i) \equiv p(x|G, i)$ is a yet unknown probability distribution with respect to G. The quantity $(\kappa(u; v) + \kappa(v; u))/2$ can then be viewed as the average distance of the pair u, v to the rest of the graph G. The co-occurrence count of u and v in G is thus defined by

$$n_G(u, v) = \frac{\kappa(u; v) + \kappa(v; u)}{2\kappa(u, v)}. \quad (9)$$

In order to calculate the probabilities $p_G(\cdot|\cdot)$ we employ the following heuristic: Starting with a uniform distribution $p_G^{(0)}(\cdot|\cdot)$ at timestep $t = 0$, we iterate

$$\kappa^{(t)}(i; j) = \sum_{\substack{x \in G \\ x \neq j}} \kappa(i, x) p_G^{(t)}(x|i) \quad (10)$$

$$n_G^{(t)}(u, v) = \frac{\kappa^{(t)}(u; v) + \kappa^{(t)}(v; u)}{2\kappa(u, v)} \quad (11)$$

$$p_G^{(t+1)}(v|u) = \frac{n_G^{(t)}(u, v)}{\sum_{x \in G} n_G^{(t)}(u, v)} \quad (12)$$

for all pairs of vertices u, v in G until convergence. Experimentally, we find that convergence is always achieved. After the execution of this iterative process we divide each count by the smallest count in order to achieve a lower bound of 1.

A pair u, v may appear in multiple graphs in the same sub-phrase-table C. The total co-occurrence count of u and v in C and the associated conditional probabilities are thus given by

$$n_C(u, v) = \sum_{G \in C} n_G(u, v) \quad (13)$$

$$p_C(v|u) = \frac{n_C(u, v)}{\sum_{x \in C} n_C(u, x)}. \quad (14)$$

A pair u, v may appear in multiple sub-phrase-tables and for the calculation of the final count $n(u, v)$ we need to average over the associated counts from all sub-phrase-tables. Moreover, we have to take into account the type of the vertices: For the simplest case where both u and v represent phrase vertices, their expected count is, by definition, given by

$$n(s, s') = \sum_C n_C(s, s') p(C|s, s'). \quad (15)$$

On the other hand, if at least one of u or v is a feature vertex, then we have to consider the phrase vertex that generates this feature: Suppose that u is the phrase vertex s='acquire' and v the POS tag vertex f='NN' and they co-occur in two sub-phrase-tables C and C' with positive counts $n_C(s, f)$ and $n_{C'}(s, f)$ respectively; the feature vertex f is generated by the phrase vertices 'ownership' in C and by 'possession' in C'. In that case, an interpolation of the counts $n_C(s, f)$ and $n_{C'}(s, f)$ as in eqn. (15) would be incorrect and a direct sum $n_C(s, f) + n_{C'}(s, f)$ would provide the true count. As a result we have

$$n(s, f) = \sum_{s'} \sum_C n_C(s, f(s')) p(C|s, f(s')), \quad (16)$$

where the first summation is over all phrase vertices s' such that $f(s') = f$. With a similar argument we can write

$$n(f, f') = \sum_{s,s'} \sum_C n_C(f(s), f(s')) \times \\ \times p(C|f(s), f(s')). \quad (17)$$

For the interpolants, from standard probability we find

$$p(C|u,v) = \frac{p_C(v|u)p(C|u)}{\sum_{C'} p_{C'}(v|u)p(C'|u)}, \quad (18)$$

where the probabilities $p(C|u)$ can be computed by considering the likelihood function

$$\ell(u) = \prod_{i=1}^{N} p(x_i|u) = \prod_{i=1}^{N} \sum_{C} p_C(x_i|u)p(C|u)$$

and by maximizing the average log-likelihood $\frac{1}{N}\log \ell(u)$, where N is the total number of vertices with which u co-occurs with positive counts in all sub-phrase-tables.

Finally, the desired probability distributions are given by the relative frequencies

$$p(v|u) = \frac{n(u,v)}{\sum_x n(u,x)}, \quad (19)$$

for all pairs of vertices u, v.

4 Experiments

4.1 Setup

The data for building the phrase-table P is drawn from DE-EN bitexts crawled from `www.project-syndicate.org`, which is a standard resource provider for the WMT campaigns (News Commentary bitexts, see, e.g. (Callison-Burch et al., 2007)). The filtered bitext consists of 125K sentences; word alignment was performed running GIZA++ in both directions and generating the symmetric alignments using the 'grow-diag-final-and' heuristics. The resulting P has 7.7M entries, 30% of which are '1-1', i.e. entries (s,t) that satisfy $p(s|t) = p(t|s) = 1$. These entries are irrelevant for paraphrase harvesting for both the baseline and our method, and are thus excluded from the process.

The initial giant component P_0 contains 1.7M vertices (Figure 5), of which 30% become residues and are used to construct R. At each iteration of the decomposition of a giant component, we remove the top $0.5\% \cdot size$ cut-vertices ranked by degree of connectivity, where $size$ is the number of vertices of the giant component and set $\theta = 2500$ as the stopping criterion. The latter choice is appropriate for the subsequent step of co-clustering the components, for both time complexity and performance of the ITC algorithm.

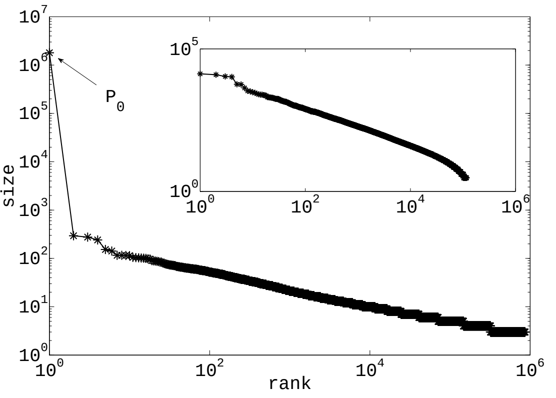


Figure 5: Log-log plot of ranked components according to their size (number of source and target phrases) for: (a) Components extracted from P. '1-1' components are not shown. (b) Components extracted from the decomposition of P_0.

In the components emerging from the decomposition of R_0, we observe an excessive number of cut-vertices. Note that vertices that consist these components can be of two types: i) former residues, i.e., residues that emerged from the decomposition of P_0, and ii) other vertices of P_0. Cut-vertices can be of either type. For each component, we remove cut-vertices that are not translations of the former residues of that component. Following this pruning strategy, the degeneracy of excessive cut-vertices does not reappear in the subsequent iterations of decomposing components generated by new residues, but the emergence of two giant components was observed: One consisting mostly of source type vertices and one of target type vertices. Without going into further details, the algorithm can extend to multiple giant components straightforwardly. For the merging process of the collection $\mathcal{D}$ we set $\delta = 5000$, to avoid the emergence of a giant component. The sizes of the resulting sub-phrase-tables are shown in Figure 6. For the ITC algorithm we use the smoothing technique discussed in (Dhillon and Guan, 2003) with $\alpha = 10^6$.

For the monolingual graphs, we set $\sigma = 0.65$ and discard graphs with more than 20 phrase vertices, as they contain mostly noise. Thus, the sizes of the graphs allow us to use analytical methods to compute the commute times: For a graph G, we form the *transition matrix* P, whose entries $P(u,v)$ are given by eqn. (7), and the *fundamen-*

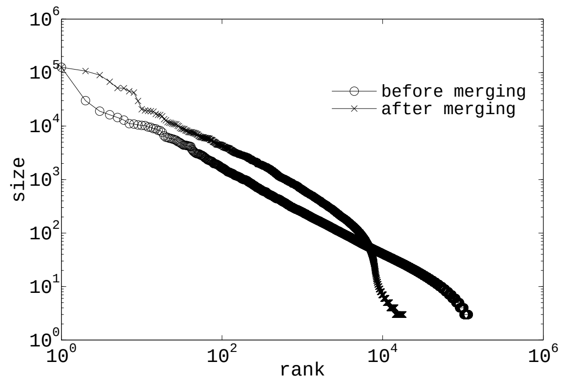


Figure 6: Log-log plot of ranked sub-phrase-tables according to their size (number of source and target phrases).

tal matrix (Grinstead and Snell, 2006; Boley et al., 2011) $Z = (I - P + \mathbf{1}\pi^T)^{-1}$, where I is the identity matrix, $\mathbf{1}$ denotes the vector of all ones and π is the vector of *stationary probabilities* (Aldous and Fill, 2001) which is such that $\pi^T P = \pi^T$ and $\pi^T \mathbf{1} = 1$ and can be computed as in (Hunter, 2000). The commute time between any vertices u and v in G is then given by (Grinstead and Snell, 2006)

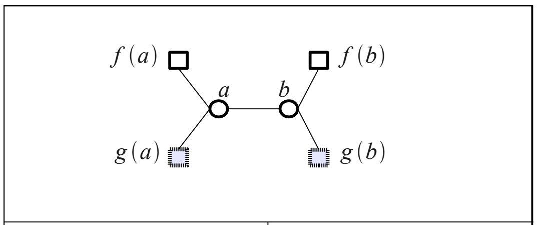
$$\kappa(u,v) = (Z(v,v) - Z(u,v))/\pi(v) \; + \\ + \; (Z(u,u) - Z(v,u))/\pi(u). \qquad (20)$$

For the parameter of eqn. (4), an appropriate choice is $\epsilon = |\Gamma(s)| + 1$; for reliable neighborhoods, this quantity is insignificant. POS tags and lemmata are generated with TreeTagger[1].

Figure 7 depicts the most basic type of graph that can be extracted from a cluster; it includes two source phrase vertices a, b, of different syntactic information. Suppose that both a and b are highly reliable with strengths $n(a;T) = n(b;T) = 40$, for some target cluster T. The resulting conditional probabilities adequately represent the proximity of the involved vertices. On the other hand, the range of the co-occurrence counts is not compatible with that of the strengths. This is because i) there are no phrase vertices with small strength in the graph, and ii) eqn. (9) is essentially a comparison between a pair of vertices and the rest of the graph. To overcome this problem *inflation* vertices i_a and i_b of strength 1 with accompanying feature vertices are introduced to the graph. Figure 8 depicts the new graph, where the lengths of the edges represent the magnitude of commute times. Observe that the quality of the probabilities is preserved but the counts are inflated, as required.

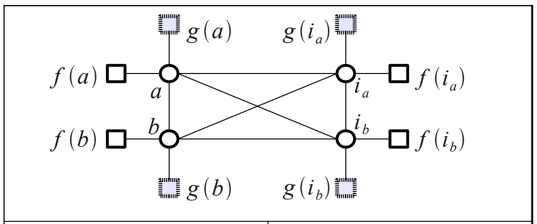
[1]http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

In general, if a source phrase vertex s has at least one translation t such that $n(s,t) \geq 3$, then a triplet $(i_s, f(i_s), g(i_s))$ is added to the graph as in Figure 8. The inflation vertex i_s establishes edges with all other phrase and inflation vertices in the graph and weights are computed as in Section 3.1. The pipeline remains the same up to eqn. (13), where all counts that include inflation vertices are ignored.



$n(a,b)$	$=2.0$	$p(b\|a)$	$=.20$
$n(a,f(a))$	$=2.6$	$p(f(a)\|a)$	$=.27$
$n(a,g(a))$	$=2.6$	$p(g(a)\|a)$	$=.27$
$n(a,f(b))$	$=1.3$	$p(f(b)\|a)$	$=.13$
$n(a,g(b))$	$=1.3$	$p(g(b)\|a)$	$=.13$

Figure 7: Top: A graph with source phrase vertices a and b, both of strength 40, with accompanying distinct POS sequence vertices $f(\cdot)$ and stem sequence vertices $g(\cdot)$. Bottom: The resulting co-occurrence counts and conditional probabilities for a.



$n(a,b)$	$=11.3$	$p(b\|a)$	$=.22$
$n(a,f(a))$	$=13.5$	$p(f(a)\|a)$	$=.26$
$n(a,g(a))$	$=13.5$	$p(g(a)\|a)$	$=.26$
$n(a,f(b))$	$=6.7$	$p(f(b)\|a)$	$=.13$
$n(a,g(b))$	$=6.7$	$p(g(b)\|a)$	$=.13$

Figure 8: The inflated version of Figure 7.

4.2 Results

Our method generates conditional probabilities for any pair chosen from {phrase, POS sequence, stem sequence}, but for this evaluation we restrict ourselves to phrase pairs. For a phrase s, the quality of a paraphrase s' is assessed by

$$P(s'|s) \propto p(s'|s) + p(f_1(s')|s) + p(f_2(s')|s), \quad (21)$$

where $f_1(s')$ and $f_2(s')$ denote the POS tag sequence and stem sequence of s' respectively. All three summands of eqn. (21) are computed from eqn. (19). The baseline is given by pivoting (Bannard and Callison-Burch, 2005),

$$P(s'|s) = \sum_t p(t|s)p(s'|t), \quad (22)$$

where $p(t|s)$ and $p(s'|t)$ are the phrase-based relative frequencies of the translation model.

We select 150 phrases (an equal number for unigrams, bigrams and trigrams), for which we expect to see paraphrases, and keep the top-10 paraphrases for each phrase, ranked by the above measures. We follow (Kok and Brockett, 2010; Metzler et al., 2011) in the evaluation of the extracted paraphrases: Each phrase-paraphrase pair is manually annotated with the following options: 0) Different meaning; 1) (i) Same meaning, but potential replacement of the phrase with the paraphrase in a sentence ruins the grammatical structure of the sentence. (ii) Tokens of the paraphrase are morphological inflections of the phrase's tokens. 2) Same meaning. Although useful for SMT purposes, 'super/substrings of' are annotated with 0 to achieve an objective evaluation.

Both methods are evaluated in terms of the Mean Expected Precision (MEP) at k; the Expected Precision for each selected phrase s at rank k is computed by $E_s[p@k] = \frac{1}{k}\sum_{i=1}^{k} p_i$, where p_i is the proportion of positive annotations for item i. The desired metric is thus given by MEP@k $= \frac{1}{150}\sum_s E_s[p@k]$. The contribution to p_i can be restricted to perfect paraphrases only, which leads to a strict strategy for harvesting paraphrases. Table 1 summarizes the results of our evaluation and

Method	Lenient MEP			Strict MEP		
	@1	@5	@10	@1	@5	@10
Baseline	.58	.47	.41	.43	.33	.28
Graphs	.72	.61	.52	.53	.40	.33

Table 1: Mean Expected Precision (MEP) at k under lenient and strict evaluation criteria.

we deduce that our method can lead to improvements over the baseline.

An important accomplishment of our method is that the distribution of counts $n(u, v)$, (as given by eqns. (15)–(17)) for all vertices u and v, belongs to the power-law family (Figure 9). This is evidence that the monolingual graphs can simulate the phrase extraction process of a monolingual parallel corpus. Intuitively, we may think of the German side of the DE–EN parallel corpus as the 'English' approximation to a 'EN'–EN parallel corpus, and the monolingual graphs as the word alignment process.

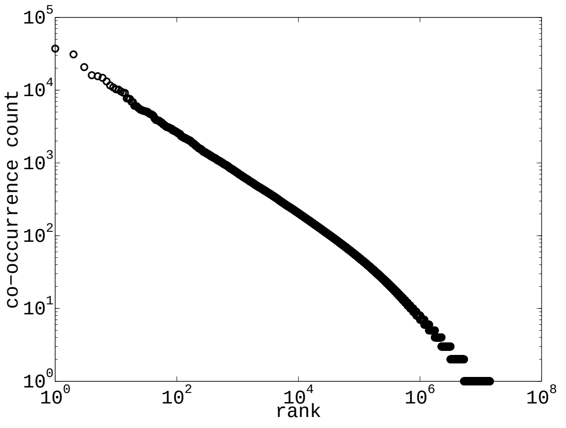


Figure 9: Log-log plot of ranked pairs of English vertices according to their counts

5 Conclusions & Future Work

We have described a new method that harvests paraphrases from a bitext, generates artificial co-occurrence counts for any pair chosen from {phrase, POS sequence, stem sequence}, and potentially identifies patterns for the syntactic information of the phrases. The quality of the paraphrases' ranked lists outperforms that of a standard baseline. The quality of the resulting conditional probabilities is promising and will be evaluated implicitly via an application to SMT.

This research was funded by the European Commission through the CoSyne project FP7-ICT- 4-248531.

References

David Aldous and James A. Fill. 2001. *Reversible Markov Chains and Random Walks on Graphs*. http://www.stat.berkeley.edu/∼aldous/RWG/book.html

Ion Androutsopoulos and Prodromos Malakasiotis. 2010. *A Survey of Paraphrasing and Textual Entailment Methods*. Journal of Artificial Intelligence Research, 38:135–187.

Colin Bannard and Chris Callison-Burch. 2005. *Paraphrasing with Bilingual Parallel Corpora*. Proc. ACL, pp. 597–604.

Alain Barrat, Marc Barthlemy, and Alessandro Vespignani. 2004. *Modeling the Evolution of Weighted Networks*. Phys. Rev. Lett., 92.

Daniel Boley, Gyan Ranjan, and Zhi-Li Zhang. 2011. *Commute Times for a Directed Graph using an Asymmetric Laplacian*. Linear Algebra and its Applications, Issue 2, pp. 224–242.

Chris Callison-Burch. 2008. *Syntactic Constraints on Paraphrases Extracted from Parallel Corpora*. Proc. EMNLP, pp. 196–205.

Chris Callison-Burch, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. 2007 *(Meta-) Evaluation of Machine Translation*. Proc. Workshop on Statistical Machine Translation, pp. 136–158.

Chris Callison-Burch, Philipp Koehn, and Miles Osborne. 2006 *Improved statistical machine translation using paraphrases*. Proc. HLT/NAACL, pp. 17–24.

Inderjit S. Dhillon and Yuqiang Guan. 2003. *Information Theoretic Clustering of Sparse Co-Occurrence Data*. Proc. IEEE Int'l Conf. Data Mining, pp. 517–520.

Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. 2003. *Information-Theoretic Coclustering*. Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, pp. 89–98.

William Dolan, Chris Quirk, and Chris Brockett. 2004. *Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources*. Proc. COLING, pp. 350-356.

Juri Ganitkevitch, Chris Callison-Burch, Courtney Napoles, and Benjamin Van Durme 2011. *Learning Sentential Paraphrases from Bilingual Parallel Corpora for Text-to-Text Generation*. Proc. EMNLP, pp. 1168–1179.

Charles Grinstead and Laurie Snell. 2006. *Introduction to Probability*. Second ed., American Mathematical Society.

Jeffrey J. Hunter. 2000. *A Survey of Generalized Inverses and their Use in Stochastic Modelling*. Res. Lett. Inf. Math. Sci., Vol. 1, pp. 25–36.

Philipp Koehn. 2009. *Statistical Machine Translation*. Cambridge University Press, Cambridge, UK.

Stanley Kok and Chris Brockett. 2010. *Hitting the Right Paraphrases in Good Time*. Proc. NAACL, pp.145–153.

Roland Kuhn, Boxing Chen, George Foster, and Evan Stratford. 2010. *Phrase Clustering for Smoothing TM Probabilities: or, how to Extract Paraphrases from Phrase Tables*. Proc. COLING, pp.608–616.

Nitin Madnani and Bonnie Dorr. 2010. *Generating Phrasal and Sentential Paraphrases: A Survey of Data-Driven Methods*. Computational Linguistics, 36(3):341–388.

Donald Metzler, Eduard Hovy, and Chunliang Zhang. 2011. *An Empirical Evaluation of Data-Driven Paraphrase Generation Techniques*. Proc. ACL:Short Papers, pp. 546–551.

Takashi Onishi, Masao Utiyama, and Eiichiro Sumita. 2010. *Paraphrase Lattice for Statistical Machine Translation*. Proc. ACL:Short Papers, pp. 1–5.

Delip Rao, David Yarowsky, and Chris Callison-Burch. 2008. *Affinity Measures based on the Graph Laplacian*. Proc. Textgraphs Workshop on Graph-based Algorithms for NLP at COLING, pp. 41–48.

George U. Yule. 1925. *A Mathematical Theory of Evolution, based on the Conclusions of Dr. J. C. Willis, F.R.S.* Philos. Trans. R. Soc. London, B 213, pp. 21–87.

Shiqi Zhao, Haifeng Wang, Ting Liu, and Sheng Li. 2008. *Pivot Approach for Extracting Paraphrase Patterns from Bilingual Corpora*. Proc. ACL, pp. 780–788.