Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Predictive Mining of Comparable Entities from the Web

Myungha Jang, Jin-woo Park, Seung-won Hwang
Pohang University of Science and Technology (POSTECH), Korea, Republic of
{myunghaj, jwpark85, swhwang}@postech.edu

Abstract

Comparing entities is an important part of decision making. Several approaches have been reported for mining comparable entities from Web sources to improve user experience in comparing entities online. However, these efforts extract only entities explicitly compared in the corpora, and may exclude entities that occur less-frequently but potentially comparable. To build a more complete comparison machine that can infer such missing relations, here we develop a solution to predict transitivity of known comparable relations. Named CLIQUE-GROW, our approach predicts missing links given a comparable entity graph obtained from versus query logs. Our approach achieved the highest F1-score among five link prediction approaches and a commercial comparison engine provided by *Yahoo!*.

1 Introduction

To assist decision making, it is useful to compare entities that share a common utility but have distinguishing peripheral features. For example, when deciding on a new mobile device to purchase, a customer benefits from knowing products with similar specifications, *e.g.*, iPhone, Nexus One and Blackberry.

One possible approach is *comparable entity mining*, which extracts comparable pairs that are explicitly compared on the Web corpus. However, these techniques are limited by their ability to mine only entities explicitly compared in Web sources, excluding entities that are potentially comparable but are not currently explicitly compared in the corpora. However, for a fully-functional comparison suggestion system, such comparisons should not be disregarded. In fact, such missing links for comparable entities are inevitable even with large datasets.

An orthogonal approach is *predictive mining*, which can complement existing mining approach. It expands the known comparable relations using transitivity to infer the unknown relations. We stress that the two approaches are clearly different: for the task of classifying missing links into comparable and non-comparable ones, the former leads to zero precision and recall, whereas the predictive mining can classify them with reasonable accuracy [1]. We first

[1] Our proposed scheme achieved precision 83% and recall 19%.

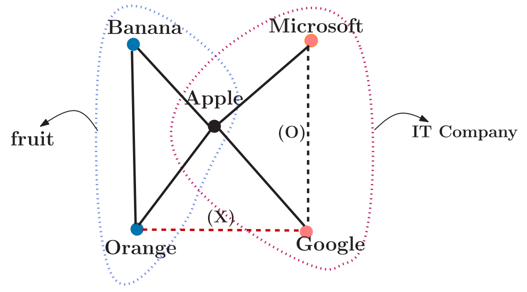

Figure 1: Two different sub-graphs of “fruit” and “IT Company” are connected by a bridge node “Apple”; (o) – comparable edge, (x) – non-comparable edge

consider a *comparable entity graph (CE-graph)* containing these comparable entities and binary relations. It is an undirected graph $G = (V, E)$ where V is a set of named-entities, E is a set of edges where $(v_i, v_j) \in E$ indicates that v_i and v_j are comparable. An initial CE-graph can be constructed with entity pairs that are explicitly compared and mined by using techniques and resources proposed in comparable entity mining (Jindal and Liu 2006; Li et al. 2010; Jain and Pantel 2011). For an unconnected pair of nodes in a CE-graph, we should next determine the comparability of the pair, *i.e.*, we should predict a link between the nodes if the pair is comparable.

To infer such transitivity, two challenges must be overcome. First, ambiguous entities may serve as *bridge nodes* in a graph, which connect two semantically different subgraphs in a CE-graph. For example, *Apple* is the bridge node that connects two subgraphs, (Fruit: Apple, Banana, Orange) and (IT company: Apple, Microsoft, Google) (Fig. 1). Bridge nodes may cause an incorrect prediction deduced from a graph topology, such as (Orange, Google).

Second, the sparseness of an initial CE-graph offers little structural information for link prediction. For example, in a CE-graph obtained from Microsoft Live Search *versus* query logs collected over one month, the number of entity pairs explicitly compared is only 0.03% of all possible pairs of entities in the *versus* query logs (*i.e.*, 5,129 pairs among about 14 million possible pairs of 5,368 entities). Later we empirically show that applying generic link prediction algorithms to such a sparse graph achieves very low recall for prediction.

To predict the missing links considering these challenges, the three criteria listed below are required for a possible solution to properly expand known relations using transitivity.

- **Graph structure**: To infer transitivity of links in the given graph, graph structure should be considered to reflect how likely the two nodes are to be connected via neighbors.
- **Attributes**: To determine whether two nodes are comparable, attributes (*e.g.*, semantics) of nodes should be considered.
- **Disambiguation**: Graphs inevitably include ambiguous nodes, which should be disambiguated to prevent generation of heterogenous clusters.

In this paper, we present CLIQUEGROW, a new clustering algorithm that satisfies the three criteria.

2 Related Work

We survey two research areas related to our work: (1) comparable entity mining that complements our prediction work and (2) link prediction methods that compete with ours work.

2.1 Comparable Entity Mining

Several approaches exist to extract comparable entities from various web corpus. Jindal and Liu proposed supervised mining of comparable entities from comparative sentences (Jindal and Liu 2006); their method uses a class sequential rule (CSR) to classify sentences into comparative or non-comparative. This method requires a comparative keyword set for training sequential rules; but keyword sets should be manually defined. To overcome this drawback, Li et al. proposed a weakly-supervised bootstrapping method to identify comparative questions and extract comparable entities (Li et al. 2010). Recently, Jain and Pennacchiotti used pattern learning methods to extract comparable entities from both query logs and web documents. Their experiments showed that query logs are superior to web documents as resources from which to extract comparable entities (Jain and Pennacchiotti 2010).

The above entity mining techniques focus on *mining* comparable pairs readily observed in the web corpus, but our work focuses on *predicting* pairs that cannot be observed from it. Our prediction work thus complements existing comparable entity mining; when used together, both approaches achieve the goal of obtaining a comparable entity set.

2.2 Link Prediction

In this section, we describe two main link prediction approaches–(1) using graph structure and (2) using clustering.

Using graph structure This approach uses graph structure to solve link prediction problems. The type of graph structure used includes node neighbors and the ensemble of all possible paths. We do not cover these methods in detail,

Table 1: Characteristics of clustering algorithms sorted by the three criteria

Method	Structure	Attribute	Disambiguation
MC-Cluster	√		√
TP-Cluster		√	√
SA-Cluster	√	√	
Our method	√	√	√

since they have been well-studied and evaluated in a previous survey (Liben-Nowell and Kleinberg 2003).

Using clustering For this approach, we specifically discuss three methods that come closest to meeting the three criteria listed above (Table 1).

MC-Cluster is a generic graph clustering that considers the presence of bridge nodes (Scripps and Tan 2006). In the given graph, MC-Cluster generates clusters in which every pair of nodes in an Must-Link (ML) edge belong to the same cluster, and any nodes in an Cannot-Link (CL) edge cannot in the same cluster. A node is identified as a bridge node when it is connected to two nodes by ML edges and the two nodes are connected by a CL edge. The graph is disambiguated by cloning the bridge node to several nodes such that each belongs to one cluster.

TP-Cluster was devised for a word sense induction problem that clusters semantically-similar words among a list of words that co-occurred with a given word. (Bordag 2006). To disambiguate words, TP-Cluster creates all possible triplets from a list of words that the target word is co-occurred with. Each triplet contains the intersection of the co-occurrence list from each word; the intersection is used as a feature of the triplet. Two triplets are merged if they share similar features, until the algorithm converges. TP-Cluster is a context-based algorithm, so when clustering it considers node attributes, not graph structure.

SA-Cluster is a graph clustering algorithm that considers both structural and attributal properties (Zhou, Cheng, and Yu 2009). SA-Cluster converts attributes to structural properties by inserting attribute nodes that are connected to all nodes that have the corresponding attribute. They exploit a unified random walk distance on the augmented graph. We discovered that SA-Cluster shows a poor performance for our problem due to bridge nodes, and attribute nodes that are commonly-shared by many nodes. As a result, the augmented graph becomes very densely-connected, and may become one big heterogenous cluster instead of several homogenous clusters.

3 Our Approach: CLIQUEGROW

In this section, we introduce CLIQUEGROW, which is a clustering approach that is designed to meet the three criteria. CLIQUEGROW contains two phases: (1) graph enrichment and (2) clustering. In clustering, we aim to find clusters in which all entities within the same cluster are comparable to each other. Clustering is effective despite graph sparseness because all possible links are automatically inferred when an entity is included in the cluster.

Table 2: Types and Topic-indicative probabilities for the entity "Apple".

Types (Step 1)	Collapsed Types (Step 2)	TI Probability (Step 3)
Company Business Organization	Company	0.68
Computer Electronics	Computer	0.20
Fruit	Fruit	0.11
Artist Person	Artist	0.01

3.1 Graph Enrichment

In this phase, a CE-Graph is enriched with semantic knowledge, namely *types*. Types describe the domains to which an entity belongs, and can be obtained from a taxonomy database (examples: Table 2). However, we cannot directly use such types to determine whether two entities are comparable–One may argue if they have the same type, they are comparable; but we find that this is not the case. Types are defined in varying granularity such that some types cover too broad a concept, so a pair having a common type might not be always comparable (i.e., false positive). Alternatively, some types are too narrow that a pair having different types might be comparable (i.e., false negative). To illustrate, in Freebase, 99% of non-comparable pairs would share the same Freebase type and thus would be falsely predicted to be comparable.

In this graph enrichment phase, we thus refine the type data by sorting and ranking to estimate the probability that two entities are comparable in clustering. The graph enrichment has four steps: Step 1 obtains the type, Steps 2 and 3 sort and rank them, and Step 4 extends the coverage of types in the graph.

Step 1: Adding Types to Entity Pairs We first obtain types from a taxonomy such as Freebase [2], which is an open-sourced web-scale taxonomy for over 41 million entities. On average, each entity is associated with 15 types. We stress that although we use Freebase, our approach is not specific to this source and can be applied to other such resources.

We match an entity v_i to the entries in Freebase whose names are identical to that of entity v_i. We use a lemmatizer to cover the entities in several forms. An entity can have many types because it may be used in several contexts, or have several meanings (Table 2). A type set for each entity v_i in G is represented as a multi-dimensional binary vector t_i, in which t_i^k = 1 if an entity v_i has k-th type t^k, and t_i^k = 0 otherwise. After obtaining the types in the graph, we collapse dependent types (Step 2) and rank the most-likely type for the intent of the entity (Step 3).

Step 2: Collapsing Dependent Types Types that are highly dependent on other types do not offer extra information, so we may collapse these dependent types to increase computational efficiency and efficacy. Highly-dependent types include semantically-similar types (*e.g.*, "company", "business", and "organization") and super-types (*e.g.*, "computer" and "electronics") (Table 2).

We define a dependency score of t^i for t^j as:

$$dep(t^i \mid t^j) = \textit{co-occurence}(t^i, t^j)/\textit{occurence}(t^j) \qquad (1)$$

where occurrence(t^i, t^j) is the number of entities that have both t^i and t^j, and occurrence(t^j) is the number of entities that have t^j. t_i is said to be dependent on t_j if $dep(t_i \mid t_j) >$ type removal threshold σ, which suggests all entities that have t_j also have t_i with a high probability. In this case, the existence of t_i is implied by the existence of t_j, which motivates us to remove such t_j (see 2nd column in Table 2).

Collapsing dependent types reduces computational cost by avoiding unnecessary comparisons and increases efficacy (Figure 3) by allowing the Topic-Indicative Probability (Step 3) to be properly calculated.

Step 3: Topic-Indicative Probability of Types The types attached to each entity must be ranked by how representative they are to the user's search objective. For example, when "Apple" was compared with many entities such as "Microsoft (in company)", "Banana (in fruit)", but not with any entity in "Artist", we can infer that "Apple" is likely to be used as "company" or "fruit" but highly unlikely to be used as "Artist" in the CE-graph.

For each type of entity in the CE-graph, we calculate topic-indicative ($\mathcal{TI}$) probability, which refers to the probability that the corresponding type is the representative intent of the entity in the CE-graph.

In existing work, such probability has been computed for a set of entities that belongs to the same concept, using a naive Bayes model (Song et al. 2011). However, in our problem context, identifying such a set is our problem goal. We thus modify the model to first infer the representative type for a given edge, which contains the smallest set of entities used in the same context:

$$P(t^k \mid (v_i, v_j)) = \frac{P((v_i, v_j)|t^k)P(t^k)}{p((v_i, v_j))} \qquad (2)$$

$$P((v_i, v_j)|t^k)P(t^k) \propto \frac{t_i^k \cdot t_j^k \cdot W(v_i, v_j)}{\sum_{(v_p, v_q)\in E} t_p^k \cdot t_q^k \cdot W(v_p, v_q)}. \qquad (3)$$

where $W(v_i, v_j)$ is the edge weight between v_i and v_j defined as occurrences of (v_i vs v_j) in the query logs.

After defining the types for each edge, we can use the probability of types in neighboring edges to infer the $\mathcal{TI}$ probabilities of each entity, because the likelihood that a type is a representative topic increases with the frequency at which it is compared to its neighbors. $\mathcal{TI}$ probability of type t^k for entity pair (v_i, v_j) is defined as:

$$P(t^k \mid v_i) = \frac{P(v_i \mid t^k)P(t^k)}{P(v_i)} = \frac{P(v_i, t^k)}{P(v_i)}, \qquad (4)$$

In Eq. 4, we infer $P(v_i \mid t^k)$ from edges of v_i:

$$P(v_i, t^k) = \sum_{v_j \in N(v_i), (v_i, v_j)\in E} P(t^k \mid (v_i, v_j))P((v_i, v_j)). \qquad (5)$$

[2]http://www.freebase.com

$P(t^k|v_i)$ is normalized such that the sum of the probabilities for all types given the entity is one. To illustrate, $\mathcal{TI}$ probabilities for edges around "Apple" were calculated (see 3rd column in Table 2).

Step 4: Type Propagation The CE-graph still includes unlabeled entities, *i.e.*, nodes that are not identified with any type. Unlabeled entities are intrinsic due to the dynamic nature of the Web whereby new entities are introduced continually. To propagate types, we adopt a state-of-the-art label propagation algorithm, Gaussian Random Field (GRF) (Zhu, Ghahramani, and Lafferty 2003). A new challenge in our problem context is that more than one types are propagated; we straightforwardly address this by weighted propagation using type probability. More specifically, probabilistic label matrix Y is modified from a binary matrix in the original matrix, to $Y_{ij} = P(t^j \mid v_i)$.

3.2 Clustering

Algorithmic Framework CLIQUEGROW is an agglomerative algorithm that aims to group nodes into clusters of mutually comparable entities, such that obtaining a transitive closure of each cluster would complete the CE-graph. Once clusters are identified, any two nodes belonging to the same cluster are comparable.

CLIQUEGROW starts with seed unit clusters and iteratively merges other base structures, until they converge to natural clusters. We use triangles (closed triplets) as initial seeds because a triangle is the basic unit of *transitive closure* that is observed. A triangle defines a unique topic among the three pairs of comparable entities of a triangle.

Using triangles as seeds, we gradually grow clusters, by connecting to neighboring entities. By the nature of an agglomerative approach, the topic purity is diluted as the cluster grows. We thus quantify the quality of triangles and populate a priority queue $\mathcal{H}$, to expand only high quality triangles. The quality is quantified as the lowest edge weight of a triangle, as a triangle with a high quality score corresponds to the clique in which every pair co-occurs frequently.

In this process, bridge nodes are first automatically disambiguated by being split into several triangles in the seeds, in which each triangle represents a single semantic. Link prediction is done in this process as well, as the cluster grows– When new entities are added, new links from all possible pairs of entities are inferred. We define and utilize a metric *comparability power* ($\mathcal{CP}$), to quantify the comparability of a a new base structure to the unit cluster that was grown from an initial seed.

Let A and B be groups, represented by a set of nodes, that have $\mathcal{TI}$ probability. $\mathcal{CP}$ is computed as:

$$\mathcal{CP}(A,B) = \sum_{i=1}^{m} \sum_{j=1}^{m} P(t_i,t_j) \cdot P(t_i \mid A) \cdot P(t_j \mid B), \tag{6}$$

where m denotes the number of types, $P(t_i \mid A)$ is a $\mathcal{TI}$ probability of t_i in a unit structure A, and $P(t_i,t_j)$ is a probability that t_i and t_j exist from each of comparable entities. In Eq. 6, $P(t_i \mid A)$ is computed as:

$$P(t_i \mid A) = \frac{\sum_{v_k \in A} P(t_i \mid v_k)}{|A|} \tag{7}$$

In Eq. 6, $P(t_i,t_j)$ is computed as:

$$P(t_i,t_j) = \frac{\sum_{v_p^i \wedge v_q^j = 1} W(v_p,v_q)}{\sum_{t_r^i \vee t_s^j = 1} W(v_r,v_s)} \tag{8}$$

With this metric defined, CLIQUEGROW iteratively identifies the highest-quality triangle seed and grows it into a cluster set by collecting qualifying neighboring base structures as follows:

1. Retrieve all triangle structures from the G and insert them into a priority queue $\mathcal{H}$, ordered by the minimum of the edge weights.
2. Pick the top seed (a triangle) in the sorted $\mathcal{H}$, and compute its $\mathcal{TI}$ probability (Eq. 7).
3. Compute $\mathcal{CP}(S,B)$ for each neighboring base structure B from seed S (Eq. 6). Include the corresponding structure in the cluster, if $\mathcal{CP}(S,B) >$ clustering threshold (CT) δ.
4. Update the $\mathcal{TI}$ probability for the expanded cluster at this iteration.
5. Go to Step 3 and repeat until expansion ceases.
6. Remove the clustered triangles from $\mathcal{H}$ go to Step 1 and iterate until $\mathcal{H} = \varnothing$.

Algorithm Implementations Two implementations of CLIQUEGROW are possible, one using an edge as a base structure (CLIQUEGROW+E) and the other using a triangle (CLIQUEGROW+T).

CLIQUEGROW+E For each neighboring edge of the cluster (or seed at first), this algorithm calculates the $\mathcal{CP}$ between the edge and the cluster. Specifically, given a cluster S and an edge e_i, if $\mathcal{CP}(S,e_i) >$ threshold δ_e, a new node on the edge is included in the cluster.

CLIQUEGROW+T This algorithm inspects whether a neighboring triangle t_i is comparable to cluster S, such that $\mathcal{CP}(S,t_i) >$ threshold δ_t. However, triangle structures are rare in sparse graphs, so we add Step 0, prior to Step 1, to add triangles to the graph:

0. Retrieve all open triplets from graph G and calculate $\mathcal{CP}$ between the two edges from the triplet. If $\mathcal{CP}>$ seed transitivity threshold γ, and add a new edge for the missing link with a low weight (*e.g.*, zero) such that these triangles are considered after the original triangles are processed as seeds in the sorted $\mathcal{H}$.

4 Competing Algorithms

We compared our algorithms to six approaches: two from an approach that use a graph structure to predict links (LPREDICT, DLPREDICT), three that use a clustering approach (MC-Cluster, TP-Cluster, SA-Cluster), and one from a commercial service (Yahoo! versus query suggestion).

4.1 Link Prediction Algorithms using Graph Structure

We compare CLIQUEGROW with LPREDICT and DLPREDICT. LPREDICT is a generic link prediction algorithm that uses graph structure. We choose AdamicAdar algorithm,

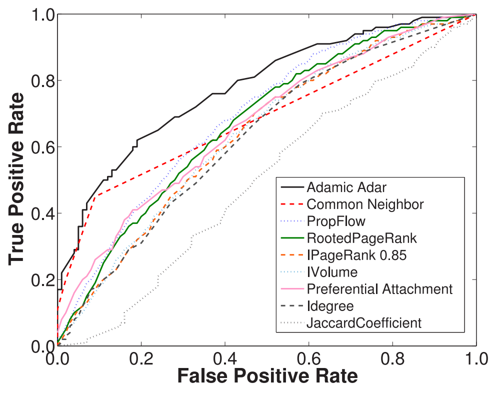

Figure 2: Receiver Operating Characteristics (ROC) of nine generic link prediction algorithms

which our performance test showed had the best prediction accuracy in among nine existing algorithms (AdamicAdar, CommonNeighbor, PropFlow, RootedPageRank, IPageRank, IVolume, Preferential Attachment,IDegree, and JaccardCoefficient (Newman 2001; Liben-Nowell and Kleinberg 2003; Lichtenwalter, Lussier, and Chawla 2010)). In this test, we used 80% of the graph as the training example and 20% as the test set, using an open-source solution for link (Lichtenwalter and Chawla 2011) (Figure 2).

Since a generic link prediction algorithm does not consider the presence of bridge nodes, we design DLPREDICT, which combines graph disambiguation and a link prediction. The purpose of this approach is to show that a link prediction using only graph structure is not as efficient as CLIQUEGROW in our problem because the CE-graph is extremely sparse.

DLPREDICT contains three phases: graph enrichment; graph disambiguation; and link prediction.

Phase 1: Graph Enrichment A graph is first enriched (Section 3.1) to be disambiguated.

Phase 2: Graph Disambiguation Once edges are curated with $\mathcal{TI}$ probabilities in the enriched graph, we identify bridge nodes. For this purpose, we first perform *entity disambiguation* on edges to assign a *surface type* for each, out of many types associated with each edge. Then a bridge node is identified as a node connected to heterogeneous surface types. Existing entity disambiguation methods leverage *context* of an entity to find its surface type (Cucerzan 2007; Kulkarni et al. 2009), *e.g.*, using Wikipedia disambiguation page (Cucerzan 2007). Our work can be viewed as using an open-source taxonomy (*e.g.*, Freebase), as such context. Our preliminary study suggests that using such taxonomy achieves comparable accuracy to existing approaches that require crawling of Wikipedia or a Web corpus, but we leave an extensive analysis as future work.

The surface type $t^{\widehat{k}}$ of edge (v_i, v_j) is defined with $\widehat{k}$:

$$\widehat{k} = \underset{k}{\operatorname{argmax}}\, min(P(t_k \mid v_i), P(t_k \mid v_j)) \tag{9}$$

Once the surface types are identified, a bridge node connected to neighbors with m heterogeneous types can be split into m clone nodes representing each type. The CE-graph is thus disambiguated in the form a few homogenous subgraphs.

Phase 3: Link Prediction We apply a generic link prediction algorithm to the disambiguated CE-graph.

4.2 Clustering Algorithms

How each clustering method works was described in Section 2, so we only explain details relevant to implementation.

MC-Cluster We first sort edges by descending weight, then set the top 10% edge weights as ML edges and the bottom 10% edges as CL edges.

TP-Cluster was originally not graph-based but context-based, which requires a list of co-occurrence words obtained from external Web documents for each entity. However, we do not have such contextual data in our problem, so we modified TP-Cluster to be graph-based. Instead of a co-occurrence list, we used the list of neighboring nodes as a comparison list (Bordag 2006).

SA-Cluster We set $k = 200$ and density value $= 0.06$.

4.3 Commercial Service

We crawled the list of *versus* query suggestion for each entity from Yahoo!, and compared the result with ours. Yahoo! automatically suggests a list of comparable entities that are extracted from various comparable queries when entity A is typed in with *'A versus ...'*. Since Yahoo! suggests maximum 10 comparable entities, the complete set of comparable entities is unknown when the number of comparable entities is ≥ 10. Hence, we identified entities with < 10 comparable entities; these comprise $\sim$20% of all entities, or $\sim$500 entities.

5 Algorithm Evaluation

5.1 Experiment Setup

Graph Construction We used *versus query* logs to construct an initial CE-graph. Although more-sophisticated techniques or proprietary resources can be used to obtain a denser CE-graph, our focus is to show how we reinforce the given CE-graph. Thus we use readily available resource that does not require any complicated tool. These logs are Web queries in several forms such as "A [versus/vs/v.s] B". These versus query logs represent explicit user intention to make comparisons. We used Microsoft Live Search 2006 query logs[3], composed of 14.9×10^6 search queries collected over one month. The number of *versus* queries in the logs was 0.7×10^6; from these we constructed an initial graph that contained 9,574 entities and 15,287 edges.

[3]This log was awarded as a part of Microsoft Research Asia research award.

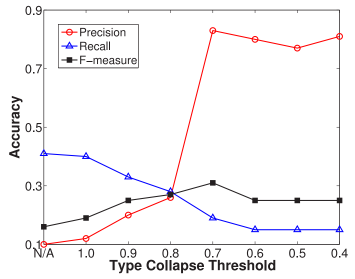

Figure 3: Clustering accuracy varies on TCT

Gold Standard We manually labeled a gold standard set on comparable entities in the *versus* query logs. We labeled 7,233 pairs of entities in 447 clusters of mutually comparable entities.

To validate our gold standard, we performed a user study in which 10 human assessors were given 50 questions that ask to determine whether or not each entity pair is comparable. Twenty-five comparable and twenty-five non-comparable pairs were randomly selected from the gold standard. To qualitatively measure the agreement between the gold standard and assessors' answers, we use Cohen's kappa (κ) (Cohen 1960). We obtained $\kappa = 0.9$, which indicates that human assessors reached a very good agreement on our gold standard (Altman 1990).

Evaluation Metric We used two evaluation metrics depending on the purpose of the each experiment. To compare the effectiveness among DLPREDICT, CLIQUEGROW, Commercial service, we use precision, recall and F1-score. Precision is the number of correct links predicted divided by the number of links predicted by the algorithm. Recall is the number of correct links predicted divided by the number of links to be predicted in the gold standard.

To compare the effectiveness of CLIQUEGROW and other clustering algorithms, we adopted Extended-BCubed metrics (Amigó et al. 2009) that is designed to evaluate the quality of overlapping clusters. The correctness of the relation between two entities e and e' is defined as:

$$\text{Correctness}(e,e') = \begin{cases} 1 & \text{iff } G(e) = G(e') \leftrightarrow C(e) = C(e') \\ 0 & \text{otherwise} \end{cases}$$

where G(e) denotes the gold-standard category and C(e) denotes the cluster to be evaluated. With this correctness function, BCubed precision and recall are:

$$\text{BC Precision} = \text{Avg}_e[\text{Avg}_{e'.C(e)=C(e')}[\text{Correctness}(e,e')]],$$

$$\text{BC Recall} = \text{Avg}_e[\text{Avg}_{e'.G(e)=G(e')}[\text{Correctness}(e,e')]]$$

Note that in our evaluation, we measure the recall only for the predicted pairs that did not appear in the original query graph (*i.e.*, log).

Parameter setting We empirically set three types of parameters: (1) the type collapsing threshold (TCT) σ, (2) the clustering threshold (CT) δ_e and δ_t, and (3) seed transitivity threshold (STT) γ.

Table 3: Accuracy comparison of LPREDICT, DLPREDICT, and CliqueGrow

Method	Precision	Recall	F1-score
LPREDICT	0.6233	0.0246	0.0474
DLPREDICT	0.7721	0.0108	0.0213
CLIQUEGROW+E	**0.8310**	0.1941	**0.3146**
CLIQUEGROW+T	0.5234	**0.2160**	0.3059

(1) Choice of TCT (σ) We observed the change of accuracy as σ varied. The number of types collapsed increased as σ decreased. When $0.7 \geq \sigma \geq 0.4$, the precision was high; (average 80%). When $\sigma = 0.8$, the precision dramatically dropped to 26%. This shows that to achieve the best accuracy of the algorithm, the choice of an adequate type removal parameter is important. Based on the experiment (Fig. 3), we set $\sigma = 0.7$.

(2) Choice of CT (δ) and STT (γ) We observed similar trade-offs between precision and recall for the remaining parameters, and similarly set the optimal parameters that gave the highest F-scores, *i.e.*, $\delta_e = 1.4$, $\delta_t = 0.5$, and $\gamma = 3.0$.

5.2 Evaluation Results

We performed two experiments to compare the effectiveness between (1) LPREDICT, DLPREDICT and CLIQUEGROW (2) CLIQUEGROW and other clustering algorithms, and (3) CLIQUEGROW and the commercial service provided by Yahoo!.

We did not compare ours with other comparable entity mining work such as (Jain and Pennacchiotti 2010), because we measured the number of correctly *predicted* edges that did not appear in the original log, and these pairs cannot be found from a mining-based approach (*i.e.*, recall of Jain's work in this scenario will be zero).

In each experiment, we measured the prediction accuracy by comparing results to the gold standard.

(1) LPREDICT, DLPREDICT vs CLIQUEGROW Among the three methods, CLIQUEGROW showed notably higher precision and recall than the other methods. Between LPREDICT and DLPREDICT, DLPREDICT improved precision by 15% but decreased recall by 1%, compared to LPREDICT (Table 3). Although graph disambiguation in DLPREDICT led to precise prediction, it produced sparser graph by separating a graph component into several sub-graphs than the original graph, which resulted in a lower recall than in LPREDICT.

(2) CLIQUEGROW vs Clustering algorithms We compared CLIQUEGROW with MC-Cluster, TP-Cluster, SA-Cluster. The two CLIQUEGROW methods gave better results than the the other algorithms (Table 4). Although MC-Cluster showed the highest precision, it also showed the lowest recall. SA-Cluster failed to give a high precision, because its augmented graph resulted in many components, which were highly-connected by inserted type nodes that are commonly shared by many entities. *CliqueGrow* + *E* gave a higher precision but lower recall than did *CliqueGrow* + *T*.

Table 4: Comparison of six clustering methods by Extended B-Cubed metrics

Method	BC Precision	BC Recall	BC F1-score
MC-Cluster	0.8132	0.2351	0.3647
TP-Cluster	0.3183	0.2662	0.2899
SA-Cluster	0.0209	0.7782	0.0407
CLIQUEGROW+E	0.6006	0.4496	**0.5142**
CLIQUEGROW+T	0.4994	0.4498	0.4733

Table 5: Comparison of CLIQUEGROW and Yahoo! query suggestion for entities in the long-tail queries

Method	Precision	Recall	F1-score
Yahoo!	0.3700	0.0637	0.1087
CLIQUEGROW+E	**0.9394**	0.1005	0.1816
CLIQUEGROW+T	0.7647	**0.1265**	**0.2171**

(3) CLIQUEGROW **vs Commercial Service** Both CLIQUEGROW variants gave higher precision, recall, and F1-score than query suggestions provided by Yahoo! (Table 5). The result shows that our approach especially has an advantage when finding comparable entities in the long-tail queries.

6 Conclusion

To predict missing links among a comparable entity graph obtained from the query logs, we developed CLIQUEGROW. CLIQUEGROW is a clustering algorithm that clusters a set of comparable entities from the given graph, inferring the missing links. CLIQUEGROW gave a higher F-measure than did other link prediction approaches and Yahoo! search engine. Our results are superior due to the predictive power employed, namely the ability to infer missing links between edges.

Acknowledgement

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea and Microsoft Research, under IT/SW Creative research program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2011-C1810- 1102-0008).

References

Altman, D. G. 1990. *Practical Statistics for Medical Research (Statistics texts)*. Chapman & Hall.

Amigó, E.; Gonzalo, J.; Artiles, J.; and Verdejo, F. 2009. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Inf. Retr.*

Bordag, S. 2006. Word sense induction: Triplet-based clustering and automatic evaluation. In *ACL*.

Cohen, J. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*.

Cucerzan, S. 2007. Large-scale named entity disambiguation based on wikipedia data. In *EMNLP and CNLL*.

Jain, A., and Pantel, P. 2011. How do they compare? automatic identification of comparable entities on the web. In *IRI*.

Jain, A., and Pennacchiotti, M. 2010. Open entity extraction from web search query logs. In *COLING*.

Jindal, N., and Liu, B. 2006. Identifying comparative sentences in text documents. In *SIGIR*.

Kulkarni, S.; Singh, A.; Ramakrishnan, G.; and Chakrabarti, S. 2009. Collective annotation of wikipedia entities in web text. In *KDD*.

Li, S.; Lin, C.-Y.; Song, Y.-I.; and Li, Z. 2010. Comparable entity mining from comparative questions. In *ACL*.

Liben-Nowell, D., and Kleinberg, J. 2003. The link prediction problem for social networks. In *CIKM*.

Lichtenwalter, R. N., and Chawla, N. V. 2011. Lpmade: Link prediction made easy. *J. Mach. Learn. Res.*

Lichtenwalter, R. N.; Lussier, J. T.; and Chawla, N. V. 2010. New perspectives and methods in link prediction. In *KDD*.

Newman, M. E. J. 2001. Clustering and preferential attachment in growing networks. *Phys. Rev. E.*

Scripps, J., and Tan, P.-N. 2006. Clustering in the presence of bridge-nodes. In *SDM*.

Song, Y.; Wang, H.; Wang, Z.; Li, H.; and Chen, W. 2011. Short text conceptualization using a probabilistic knowledgebase. In *IJ-CAI*.

Zhou, Y.; Cheng, H.; and Yu, J. X. 2009. Graph clustering based on structural/attribute similarities. *Proc. VLDB Endow.*

Zhu, X.; Ghahramani, Z.; and Lafferty, J. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*.