

# Dynamic Feature Generation for Relational Learning

Alexandrin Popescul\* and Lyle H. Ungar

Department of Computer and Information Science  
University of Pennsylvania  
Philadelphia, PA 19104  
{popescul, ungar}@cis.upenn.edu

**Abstract.** We provide a methodology which integrates dynamic feature generation from relational databases with statistical feature selection and modeling. Unlike the standard breadth- or depth-first search of a refinement graph, the order in which we generate features and test them for inclusion in the model is dynamically determined based on which features are found to be predictive. This best-first search often reduces the number of computationally expensive feature evaluations. Multiple feature streams are created based on the syntactic structure of the feature expressions; for example, based on the type of aggregate operator used to generate the feature. At each iteration, the next feature to be evaluated is taken from the stream which has been producing better features. Experiments show that dynamic feature generation and selection produces more accurate models per feature generated than its static alternative.

## 1 Introduction

We provide a statistical relational learning method which integrates dynamic feature generation with statistical modeling and feature selection. Dynamic feature generation can lead to the discovery of predictive features with less computation than generating all features in advance. Dynamic feature generation decides the order in which features are evaluated based on run-time feature selection feedback.

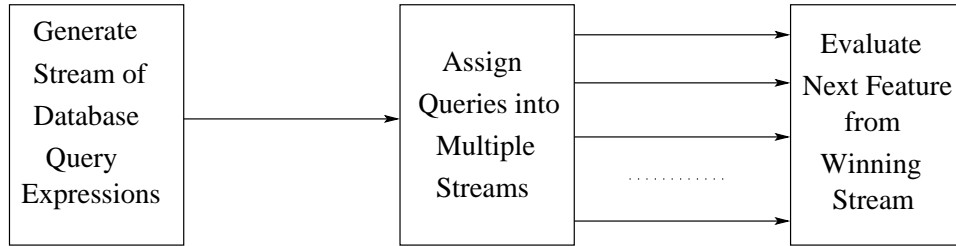
We implement dynamic feature generation within Structural Generalized Linear Regression (SGLR) framework [5, 6]. SGLR integrates generation of feature candidates from relational data with their selection using statistical model selection criteria. Relational feature generation is cast as a search in the space of queries to a relational database. This search covers the space of queries involving one or more relation instances; query evaluations produce numeric values which are candidate features. At each search node, candidate features are considered for inclusion in a discriminative statistical model such as linear or logistic regression. The result is a statistical model where each selected feature is the evaluation of a database query encoding a predictive data pattern.

The SGLR framework allows us to control the order in which features are generated and tested for inclusion in the model. Feature generation in SGLR consists of two steps: query expression generation, which is cheap as it involves only syntactic operations on

---

\* Contact author. Now at Ask Jeeves, Inc. (apopescul@askjeeves.com).

query strings, and query evaluation, which is computationally demanding. Prior to being evaluated, query expressions are assigned into multiple streams. At each iteration, one of the streams is chosen to generate the next potential feature, based on the value of the features that stream has provided relative the other streams (Figure 1). Multi-stream feature selection is motivated by the fact that not all feature candidates are equally useful. It is often possible to heuristically classify features into different streams reflecting the expectation of a modeler of how likely a stream is to produce better features, or how expensive the features are to evaluate.



**Fig. 1.** Multi-Stream Dynamic Feature Generation

In general, the split of features into multiple streams need not be a disjoint partition. For example, some streams can be formed based on the types of aggregate operators in query expressions, as done below, and other streams can be formed based on the type relations joined in a query, for example, based on whether a query contains a cluster-derived relation or a relation based of the same type as the target concept. A query would be enqueued on to one stream based on its aggregate operator, and on to a different stream based on the type of its relation instances. The method, if used with a check to avoid evaluating a query which has been evaluated previously in a different stream, will not incur significant increase in computational cost.

The need for aggregates in relational learning comes from the fact that the central type of relational representation is a table (set); aggregation summarizes information in a table into a scalar value which can be included in a statistical model. Aggregate-based feature generation often produces more accurate predictive models from relational data than pure logic-based features [4]. While rich types of aggregates can be included into feature generation, not all of them are expected to be equally useful, suggesting using the aggregate type as a heuristic for stream assignment in dynamic feature generation.

First-order expressions are treated as database queries resulting in a table of all satisfying solutions, rather than a single boolean value. The following is an example of an aggregate feature useful in link prediction; here the target concept is binary and the feature (right-hand side of the expression) is a database query about both target documents  $d1$  and  $d2$ :

$$cites'(d1, d2) \sim count [cites(d1, D), cites(d2, D)]$$

is the number of documents that both  $d1$  and  $d2$  cite.

We report below an experiment where features are queued for evaluation into two streams according to the type of aggregate operator in the query expression. We compare two-stream dynamic with one-stream static feature generation. In both cases, query expressions are generated by breadth-first search. The base-line, static, strategy evaluates queries in the same order that the expressions appear in the search queue, while the alternative, dynamic strategy, enqueues queries into two separate streams based on a syntactic criterion, here the type of aggregation, at the time its expression is generated, but chooses the next feature to be evaluated from the stream which has been producing “better” features.

We use data from CiteSeer (a.k.a. ResearchIndex), an online digital library of computer science papers [2] (<http://citeseer.org/>). CiteSeer contains a rich set of data, including text of papers, citation information, author names and affiliations, and conference or journal names. We represent CiteSeer as a relational database. For example, citation information is represented as a binary relation between citing and cited documents. Document authorship, publication venues and word occurrence are also relations:

```
Citation(from:Document, to:Document),
Author(doc:Document, auth:Person),
PublishedIn(doc:Document, vn:Venue),
HasWord(doc:Document, word:Word).
```

The next section describes the experiment testing a two-stream dynamic strategy against the static one-stream alternative.

## 2 Experimental Set-up

Feature generation in the SGLR framework consists of two steps: query expression generation, and query evaluation. The former is cheap as it involves only syntactic operations on query strings; the latter is computationally demanding. The experiment is set up to test two strategies which differ in the order in which queries are evaluated. In both strategies, query expressions are generated by breadth-first search. The base-line, static, strategy evaluates queries in the same order the expressions appear in the search queue, while the alternative, dynamic strategy, enqueues queries into separate streams at the time its expression is generated, but chooses the next feature to be evaluated from the stream with the highest ratio:

$$(featuresAdded + 1)/(featuresTried + 1)$$

where *featuresAdded* is the number of features selected for addition to the model, and *featuresTried* is the total number of features tried by feature selection in this substream. Many other ranking methods could be used; this one has the virtue of being simple and, for the realistic situation in which the density of predictive features tends to decrease as one goes far into a stream, complete.

A query expression is assigned into one of two streams based on the type of aggregate operator it uses:

- Stream 1: queries with aggregates `exists` and `count` over entire table.
- Stream 2: other aggregates. Here, these are the counts of unique elements in individual columns.

We compare test set accuracies with dynamic and static feature generation in two scenarios: i) difference in accuracies against the number of features generated and ii) difference in accuracies against time.<sup>1</sup>

## 2.1 Data Sets

The experiments are performed for two tasks using CiteSeer data: classifying documents into their publication venues, conferences or journals, and predicting the existence of a citation between two documents. The target concept pair in the two tasks are  $\langle \text{Document}, \text{Venue} \rangle$  and  $\langle \text{Document}, \text{Document} \rangle$  respectively. In the case of venue prediction, the value of the response variable is one if the pair’s venue is a true publication venue of the corresponding document and zero otherwise. Similarly, in link prediction, value of the response variable is one if there exists a citation between two documents and zero otherwise. In both tasks, the search space contains queries based on several relations about documents and publication venues, such as citation information, authorship and word content of the documents.

Each of the tasks consists of two datasets: one using the original relational representation and the other using an augmented cluster-based representation. Alternative “cluster-relations” are derived from the attributes in the original database schema and included in the feature generation process. We use clustering to derive new first class relational entities reflecting hidden topics of papers, author communities and word groups. New cluster relations included into the feature generation process in addition to the original relations result in the creation of richer cluster-based features, where clusters enter into more complex relationships with existing background relations rather than only provide dimensionality reduction. This approach can result in more accurate models than those built only from the original relational concepts [5].

The following are descriptions of basic relations we use, followed by the description of derived cluster relations we use to augment the search space:

- `PublishedIn(doc:Document, vn:Venue)`. Publication venues are extracted by matching information with the DBLP database<sup>2</sup>. Publication venues are known for 60,646 CiteSeer documents. This is the total number of documents participating in the experiments. All other relations are populated with information about these documents. There are 1,560 unique conferences and journals. Training and test examples are sampled from this background relation in the venue prediction task.
- `Author(doc:Document, auth:Person)`. 53,660 out of the total of 60,646 documents have authorship information available; there are 26,740 unique last names of authors. The number of tuples in this relation is 131,582.

<sup>1</sup> Time is recorded on machines with equivalent characteristics and not used for other processing. MySQL database engine is used.

<sup>2</sup> <http://dblp.uni-trier.de/>

- Citation (*from*:Document, *to*:Document). There are a total of 173,410 citations among our “universe” of 60,646 documents. The relation contains 42,749 unique citing documents, 31,603 unique cited documents, and the total of 49,398 documents. Training and test examples are sampled from this background relation in the link prediction task.
- HasWord (*doc*:Document, *word*:Word). This is by far the largest relation even for relatively small vocabularies. It is populated by binary word occurrence vectors, i.e. there is a tuple for each word in the vocabulary if it is contained in a corresponding document. With word data available for 56,104 documents and vocabulary of size 1,000, the total number of tuples in HasWord is 6,894,712 (the vocabulary contains top count words in the entire collection after Porter stemming and stop word removal).

We use *k*-means to derive cluster relations; any other hard clustering algorithm can be used for this purpose. The results of clustering are represented by binary relations `<ClusteredEntity, Cluster ID>`. Cluster relations are precomputed and added to the relational schema before feature generation phase. The original database schema contains several entities which can be clustered based on a number of alternative criteria. Each many-to-many relation in the original schema presented above can produce two distinct cluster relations. Three out of four relations are many-to-many (with the exception of `PublishedIn`), this results in six new cluster-relations. Since the `PublishedIn` relation is not used to produce new clusters, nothing has to be done to exclude the attributes of entities in training and test sets from participating in clustering. In the case of link prediction, on the other hand, the relation corresponding to the target concept, `Citation`, does produce clusters. Clustering is run without the links sampled for training and test sets. The following is the list of these six cluster relations which we add to the relational database schema:

- `ClusterDocumentsByAuthors(doc:Document, clust:Clust0)`.  
53,660 documents are clustered based on the identity of their 26,740 authors.
- `ClusterAuthorsByDocuments(auth:Person, clust:Clust1)`.  
26,740 authors are clustered based on 53,660 documents they wrote.
- `ClusterDocumentsByCitingDocuments(doc:Document, clust:Clust2)`.  
31,603 documents are clustered based on 42,749 documents citing them (the numbers are slightly lower in link prediction where target concept links do not participate in clustering).
- `ClusterDocumentsByCitedDocuments(doc:Document, clust:Clust3)`.  
42,749 documents are clustered based on 31,603 documents cited from them (the numbers are slightly lower in link prediction where target concept links do not participate in clustering).
- `ClusterDocumentsByWords(doc:Document, clust:Clust4)`.  
56,104 documents are clustered based on the vocabulary of top 1,000 words they contain.
- `ClusterWordsByDocuments(word:Word, clust:Clust5)`.  
The vocabulary of 1,000 words is clustered based on their occurrence in this collection of 56,104 documents.

An important aspect of optimizing cluster utility, in general, and of the use of cluster relations in our setting, in particular, is the choice of  $k$ , the number of groups into which the entities are clustered. In our case, for each potential value of  $k$ , we would need to repeat expensive feature generation for all cluster derived features. In the experiments presented here we fix  $k$  to be equal to 100 in all cluster relations except for the last one, ClusterWordsByDocuments, where the number of clusters is 10. The latter is clustered into fewer groups than the rest of the data to reflect roughly an order of magnitude smaller number of objects, words, to be clustered: we selected the vocabulary of size 1,000 to make the size of HasWord relation manageable. The accuracy of resulting cluster-based models reported below can potentially be improved even further if one is willing to incur the additional cost of optimizing the choice of  $k$ .

Table 1 summarizes the sizes of four original relations and the sizes of six derived cluster relations.

**Table 1.** Sizes of the original and cluster-based relations

Relation	Size
PublishedIn	60,646
Author	131,582
Citation	173,410
HasWord	6,894,712
ClusterDocumentsByAuthors	53,660
ClusterAuthorsByDocuments	26,740
ClusterDocumentsByCitingDocuments	31,603
ClusterDocumentsByCitedDocuments	42,749
ClusterDocumentsByWords	56,104
ClusterWordsByDocuments	1,000

Each of the sets,

- Set1: venue prediction, with cluster relations,
- Set2: venue prediction, without cluster relations,
- Set3: link prediction, with cluster relations,
- Set4: link prediction, without cluster relations,

consists of ten partitions which are used to produce 10-fold cross validation point-wise confidence estimates of the accuracy curves at chosen intervals. At each interval, confidence is derived from ten points obtained from testing ten models learned from each cross validation partition against the remaining nine partitions. The number of observations in each partition of the venue prediction dataset and of the link prediction dataset is 1,000 and 500 respectively.

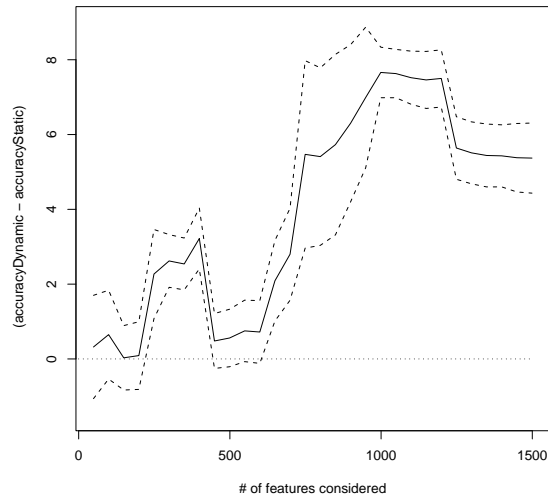
Each of the datasets has 3,500 available features. Table 2 gives the number of features in each stream. We stop the experiment when one of the streams is exhausted.

**Table 2.** Sizes of feature streams

Data	Stream 1	Stream 2
set1	1,552	1,948
set2	1,509	1,991
set3	1,778	1,722
set4	1,633	1,867

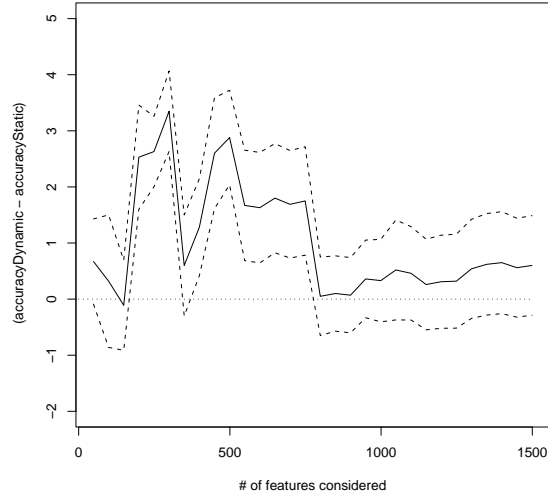
### 3 Results

This section reports the difference in test set accuracy for dynamic and static feature generation. In each of four datasets, the difference in accuracy is plotted against the number of evaluated features considered by feature selection and against the time taken to evaluate the queries. Each plot reports 95% confidence intervals of the accuracy difference. The accuracies are compared until the winner stream becomes empty. After the initial brief alternation between the two streams the process stabilized and continued to request most of the features from the winning stream, which was Stream 1 in all datasets.

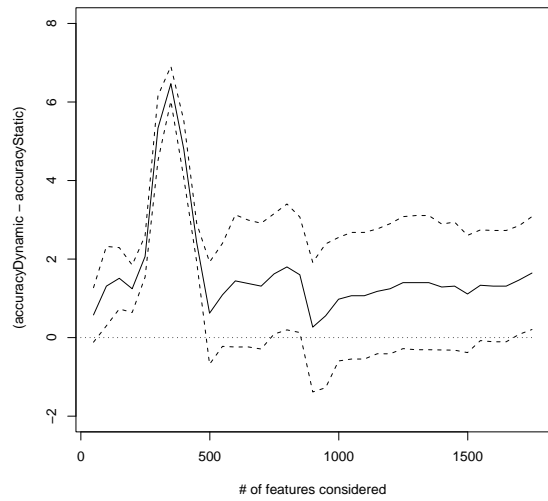


**Fig. 2.** Set 1 (venue prediction, with cluster relations). Test accuracy difference against the number of features considered. Errors: 95% confidence interval (10-fold cross validation, in each of 10 runs  $N_{train} = 1,000$  and  $N_{test} = 9,000$ )

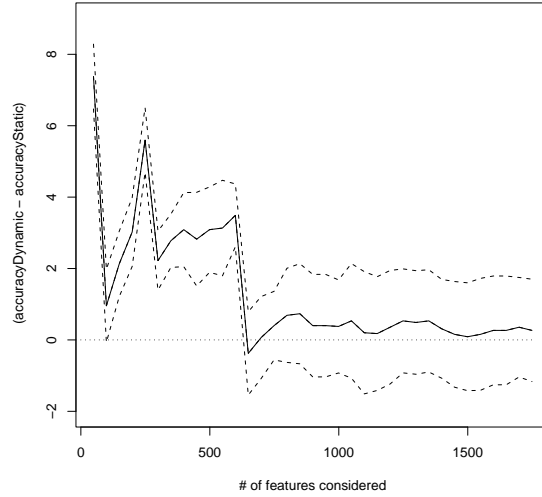
Figures 2, 3, 4 and 5 present the dynamic search test accuracy minus the static search test accuracy against the number of features generated for sets 1 through 4 respectively.



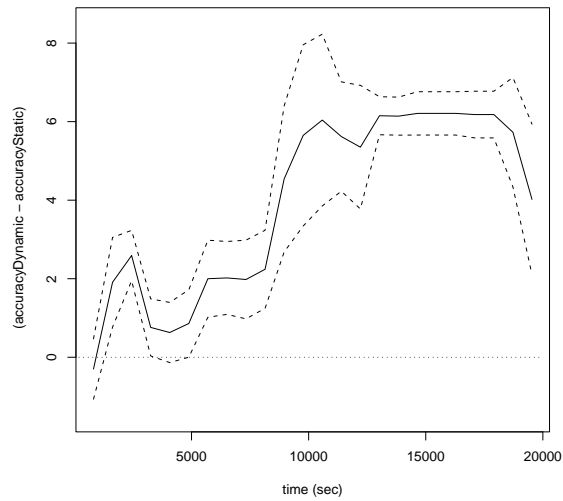
**Fig. 3.** Set 2 (venue prediction, without cluster relations). Test accuracy difference against the number of features considered. Errors: 95% confidence interval (10-fold cross validation, in each of 10 runs  $N_{train} = 1,000$  and  $N_{test} = 9,000$ )



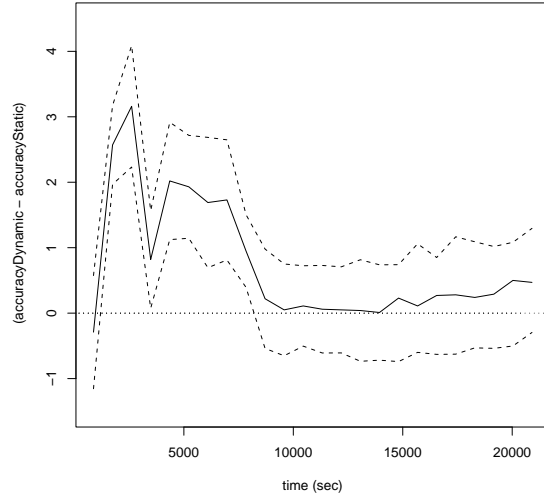
**Fig. 4.** Set 3 (link prediction, with cluster relations). Test accuracy difference against the number of features considered. Errors: 95% confidence interval (10-fold cross validation, in each of 10 runs  $N_{train} = 500$  and  $N_{test} = 4,500$ )



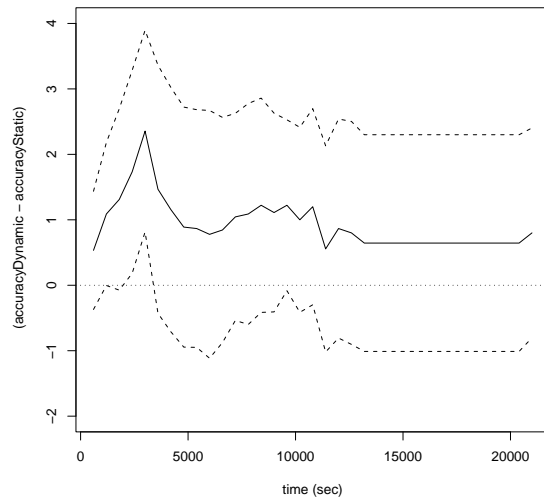
**Fig. 5.** Set 4 (link prediction, without cluster relations). Test accuracy difference against the number of features considered. Errors: 95% confidence interval (10-fold cross validation, in each of 10 runs  $N_{\text{train}} = 500$  and  $N_{\text{test}} = 4,500$ )



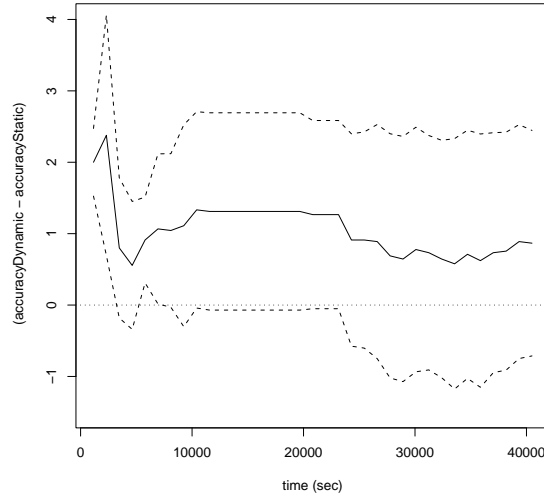
**Fig. 6.** Set 1 (venue prediction, with cluster relations). Test accuracy difference against time taken to generate features. Errors: 95% confidence interval (10-fold cross validation, in each of 10 runs  $N_{\text{train}} = 1,000$  and  $N_{\text{test}} = 9,000$ )



**Fig. 7.** Set 2 (venue prediction, without cluster relations). Test accuracy difference against time taken to generate features. Errors: 95% confidence interval (10-fold cross validation, in each of 10 runs  $N_{train} = 1,000$  and  $N_{test} = 9,000$ )



**Fig. 8.** Set 3 (link prediction, with cluster relations). Test accuracy difference against time taken to generate features. Errors: 95% confidence interval (10-fold cross validation, in each of 10 runs  $N_{train} = 500$  and  $N_{test} = 4,500$ )



**Fig. 9.** Set 4 (link prediction, without cluster relations). Test accuracy difference against time taken to generate features. Errors: 95% confidence interval (10-fold cross validation, in each of 10 runs  $N_{train} = 500$  and  $N_{test} = 4,500$ )

Figures 6, 7, 8 and 9 present accuracy difference against the time in seconds taken to evaluate features considered for selection for sets 1 through 4 respectively. These plots show early gains from dynamic feature generation. Gains tend to vanish as all predictive features are exploited. The dynamic feature generation performs significantly better than its static alternative along considerable intervals of the incremental learning process. The dynamic feature generation never performs significantly worse than the static search, based on the pairwise Gaussian 95% confidence intervals of the accuracy difference.

## 4 Discussion

We presented a method for dynamic feature generation with statistical modeling and feature selection and showed that dynamic feature generation can lead to the discovery of predictive features with less computation than generating all features in advance.

Dynamic feature generation contrasts with “propositionalization” [1, 7], a static relational feature generation approach in which features are first constructed from relational representation and then presented to a propositional algorithm. The generation of features with propositionalization is thus fully decoupled from the model used to make predictions, and prematurely incurs full computational cost of feature generation. Better models can be built if one allows native statistical feature selection criteria provide run-time feedback determining the order in which features are generated. Coupling

feature generation to model construction can significantly reduce computational costs. Some inductive logic programming systems also perform dynamic feature generation, in this case when modeling with logic. For example, Progol [3] uses A\*-like algorithm to direct its search.

In the experiments reported here, dynamic feature generation performs significantly better than its static alternative along considerable intervals of the incremental learning process. The dynamic feature generation never performs significantly worse than the static search, based on the pairwise Gaussian 95% confidence intervals of the accuracy difference.

One of the two feature streams was a clear winner, i.e. the heuristic used to split features was successful in reflecting the expectation that one stream is more likely to produce good features. In situations when the choice of a good heuristic is difficult, dynamic feature generation can still be used; in the worst case, when features in different streams are “equally good”, the method will asymptotically lead to the same performance as the static feature generation by taking features from different streams with equal likelihood.

The two stream approach can be generalized to a multi-stream approach. Also, the split of features into multiple streams does not need to be a disjoint partition. For example, some streams can be formed based on the types of aggregate operators in query expressions, as we did here, and other streams can be formed based on the type of relations joined in a query, for example, split based on whether a query contains a cluster-derived relation, or a relation of the same type as the target concept. A given query is enqueued into one stream based on its aggregate operator, and into a different stream based on the type of its relation instances. The method, if used with a check to avoid evaluating a query which has been evaluated previously in a different stream, will not incur significant increase in computational cost.

Another approach is to split features into multiple streams according to the sizes of their relation instances, which would serve as an estimate of evaluation time. This can lead to improvements for the following reasons: i) out of two nearly collinear features a cheaper one will likely be evaluated first. This will lead to approximately the same accuracy improvement as the second more expensive feature, and ii) there is no obvious correlation between the cost to evaluate a query and its expected predictive power, therefore it can be expected that cheap queries result in good features as likely as more expensive ones.

## References

1. Stefan Kramer, Nada Lavrac, and Peter Flach. Propositionalization approaches to relational data mining. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*, pages 262–291. Springer-Verlag, 2001.
2. Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.
3. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.
4. Claudia Perlich and Foster Provost. Aggregation-based feature invention and relational concept classes. In *KDD-2003*, 2003.

5. Alexandrin Popescul and Lyle Ungar. Cluster-based concept invention for statistical relational learning. In *KDD-2004*, 2004.
6. Alexandrin Popescul, Lyle H. Ungar, Steve Lawrence, and David M. Pennock. Statistical relational learning for document mining. In *ICDM-2003*, 2003.
7. A. Srinivasan and R. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.