Within-Network Classification Using Radius-Constrained Neighborhood Patterns[*]

Jialong Han
Renmin University of China
jialonghan@gmail.com

Ji-Rong Wen
Renmin University of China
jirong.wen@gmail.com

Jian Pei
Simon Fraser University
jpei@cs.sfu.ca

ABSTRACT

Within-**N**etwork **C**lassification (WNC) techniques are designed for applications where objects to be classified and those with known labels are interlinked. For WNC tasks like web page classification, the *homophily* principle succeeds by assuming that linked objects, represented as adjacent vertices in a network, are likely to have the same labels. However, in other tasks like chemical structure completion, recent works suggest that the label of a vertex should be related to the local structure it resides in, rather than equated with those of its neighbors. These works also propose structure-aware vertex features or methods to deal with such an issue.

In this paper, we demonstrate that *frequent neighborhood patterns*, originally studied in the pattern mining literature, serve as a strong class of structure-aware features and provide satisfactory effectiveness in WNC. In addition, we identify the problem that the neighborhood pattern miner indiscriminately mines patterns of all radiuses, while heuristics and experiments both indicate that patterns with a large radius take much time only to bring negligible effectiveness gains. We develop a specially designed algorithm capable of working under radius threshold constraints, by which patterns with a large radius are not mined at all. Experiments suggest that our algorithm helps with the trade-off between efficiency and effectiveness in WNC tasks.

[*]We thank the anonymous reviewers for their constructive comments. We are grateful to Chunbin Lin for his discussions and feedbacks, and Christian Desrosiers for his advices on implementing the RL-RW-Deg algorithm. This work was partially supported by the National Key Basic Research Program (973 Program) of China under grant No. 2014CB340403, the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China, an NSERC Discovery grant and a BCIC NRAS Team Project. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

CIKM'14, November 3–7, 2014, Shanghai, China.
Copyright 2014 ACM 978-1-4503-2598-1/14/11 ...$15.00.
http://dx.doi.org/10.1145/2661829.2661979.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

1. INTRODUCTION

Since the ninetieth of the last century, frequent pattern mining has been an active research theme, enabling knowledge discovery and mining in various kinds of data [10]. Besides applications like association rule mining [1] and indexing [35], according to [6], frequent patterns are especially effective features for corresponding classification tasks [21, 7], because they tend to capture the rich underlying semantics of the hosting data.

For graph data, it is obvious that the properties of the data do not hide in any single vertex or edge, but in the way they are combined and organized. In other words, the semantics of graphs lie in their structures. In the literature of graph classification, feature-based approaches, such as [7], all embrace sub-structure patterns of graphs as the most significant cue for classification.

In fact, [20] clarified that graph data is actually modeled differently in two settings: the *graph-transactional setting* and the *single-graph setting*. In the first setting such as chemical structure databases, the data is viewed as a set of relatively small graphs, called *transactions*. Graph classification tasks, such as molecule property prediction, should be categorized into this setting because they essentially classify transactions. In the second setting such as the web graph and social networks, however, the data is viewed as a large network. The corresponding classification task, commonly referred to as *Within-network classification* (denoted by WNC for short), actually involves classifying vertices rather than transactions.

Since transactions have structural patterns as indicators of their property, two analogous questions are interesting in the single-graph setting: 1) do vertices have structural patterns? and 2) if so, can they serve as vertex classification features? Our earlier work [11] answered the first question, where we formulated ***F**requent **N**eighborhood pattern **M**ining* (FNM) to mine patterns for vertices from the local structure they reside in. For example, in an academic network, a neighborhood pattern may suggest that some papers cite another paper with a common author. In a social network, a neighborhood pattern may represent persons having a son and a daughter. In the structure graph of a molecule, a neighborhood pattern may indicate that some carbon atoms appear on a cycle of length 6. It is obvious that neighborhood patterns carry rich semantics about the behavior of vertices in a network. In the first half of this paper we answer the second question based on the work of [11], by relating the label of a vertex to the neighborhood patterns it follows.

It is worth clarifying that our assumption is essentially different from the *homophily* principle, which states that the label of a vertex tends to follow the majority of the vertex's neighbors. This principle undoubtedly applies to many scenarios, e.g., friends share hobbies, and web pages link to pages with similar topics. However, properties of vertices do not always propagate along edges. For example, a carbon atom in a chemical structure may be surrounded by many hydrogen ones. Instead, we relate the classification of a vertex to the structure of its neighborhood, which is not limited to the information of all neighbors' labels. The "homophily-or-structure" issue has caught some attention in recent WNC studies [12, 8, 25]. But to the best of our knowledge, we are the first to address this issue under the context of frequent-pattern-based classification.

In the second half of this paper, we consider incorporating the Markov assumption [24] of WNC into our neighborhood-pattern-based classifier. According to the assumption, distant structures tend to have negligible effects on the classification of a vertex, thus can be neglected for the interest of efficiency. However, the neighborhood miner in [11] cannot directly control the *radius* of the patterns it finds. For example, it may produce a pattern of radius 3, saying that the center person has a friend whose father works for a company with the label "IT". This pattern may not be effective in classifying the person because it takes distant structures into consideration. We design an algorithm called r-FNM (radius-constrained frequent neighborhood mining), which accommodates a pre-defined radius threshold on the patterns. Compared to the naïve approach of post-filtrating large-radius patterns, r-FNM saves time by not generating them at all, thus making the trade-off more worthwhile.

We summarize our contributions as follows: **1)** We empirically show that neighborhood patterns are effective vertex features in WNC tasks; **2)** We propose an algorithm r-FNM to directly mine neighborhood patterns w.r.t. a given radius upper bound; **3)** We experimentally show that r-FNM helps balancing effectiveness and efficiency in WNC tasks.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 introduces relevant preliminaries. In Section 4, we discuss how to apply neighborhood patterns to classify networked data. We incorporate the radius constraints into the generation of neighborhood patterns in Section 5. We present experimental results in Section 6, followed by conclusions in Section 7.

2. RELATED WORK

The problem of within-network classification has been extensively studied in recent years. The distinguishing challenge of WNC problems is that the conventional independent and identical distribution (i.i.d.) assumption does not hold, because the labels of unclassified objects depend on each other. [24] points out that common within-network classifiers all comprise a local classifier, a relational classifier, and a collective inference procedure. The local classifier performs a rough classification using network-independent information (such as the text content of a web page or the age of a person) to provide an initial estimation. Given a vertex v, the relational classifier adjusts the prediction for v by considering both the network structure and the predictions of the other vertices. The collective inference component repeatedly calls the relational classifier on every vertex according to certain updating strategies, and returns the classification results when certain stopping criterion is met.

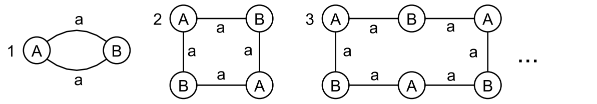


Figure 1: Random walk and degree information cannot distinguish between the neighborhoods of vertex 1, 2, 3, . . .

Our focus lies under the scope of the relational classifier component. We note that, the relational classifier actually encodes our assumption for the specific problem. The simplest wvRN [23] method, which operates under the homophily assumption, classifies a vertex using the weighted votes of its neighbors' labels. Moreover, cdRN [28] and nLB [22] adopt a more general assumption, which relates (but not equals) a vertex's label to the distribution of those of its neighbors. However, none of those classifiers exploits the rich structural information of the network. [8] proposes RL-RW-Deg, which characterizes the structure of a vertex's neighborhood by its degree and the distribution of different label sequences obtained after starting a random walk from the vertex. RL-RW-Deg is reported to outperform various non-structural methods [23, 28, 22] in the chemical structure completion task. However, we categorize RL-RW-Deg as a *semi-structural* method, because it is not enough to recover a vertex's neighborhood structure using only its degree and random walk distribution information. Consider Figure 1. Here each connected component is a cycle of an even length, where vertices have alternating A and B labels. Let $v_1, v_2, \ldots, v_i$ be an arbitrary A vertex in the component of length $2i$. Apparently, $v_1, v_2, \ldots, v_i$ reside in different neighborhood structures. However, they have equal degrees, and all of them only initiates random walks of the label sequence $AaBaAaBa\ldots$. Therefore, RL-RW-Deg does not have a full discriminative power at least in this case. In this paper, we adopt RL-RW-Deg as the baseline and compete with it on the same task. It is worth noting that [12, 25] also proposes structure-dependent approaches. However, their methods cannot exploit edge label information, thus are only applicable to homogeneous networks.

As Section 1 mentions, many WNC methods employ the Markov assumption, while ours is no exception. Besides [23, 28, 22], ego-net based graph mining methods [2, 12], which represent a vertex as the induced graph generated by its neighbors and itself, also adopt this assumption implicitly.

The collective inference component is critical to WNC tasks [15]. Common collective inference algorithms include iterative classification (ICA) [27], Gibbs sampling [4], and relaxation labeling [5]. In our method, we use ICA as our collective classifier. We refer readers to [29] for a summary of common collective classifiers and their comparison.

When evaluating classification algorithms, it is common to use cross-validation to fully exploit the labeled data. Meanwhile, it is desirable to compare the effectiveness of different WNC algorithms with networks of varied label ratio (i.e., the proportion of labeled vertices). However, the above two options are contradictory, because the fold number of cross-validation determines the ratio of training set size and testing set size, thus fixing the label ratio. To cope with this issue, work such as [24] replaces cross-validation with a simple resampling procedure, thus resulting in overlapped testing sets. [26] points out that applying paired t-test in such a scenario leads to potentially high type I error (i.e., performance improvements will be incorrectly concluded when there is none). They also propose ***N**etwork **C**ross-**V**alidation* (NCV) as a solution to enable cross-validation in WNC without introducing overlapped testing sets. In this paper, we adopt NCV in our experimental methodology. More details will be given in Section 6.1.

Like the single-graph-based WNC problem, graph classification under the graph-transactional setting has also received much attention. Common graph classification methods fall under two categories: pattern-based and kernel-based. Pattern-based methods, such as [7], employ frequent subgraph patterns as classification features, which inspires us to apply neighborhood patterns to vertex classification. Sophisticated feature selection methods [32, 18] are also proposed to select the most discriminative subgraphs for classification. Kernel-based methods, however, propose various kernel functions to measure the similarity between different transactions by the structure indicators they share. Common indicators include random walks [16], shortest paths [3], subtrees [30], cycles [13], and graphlets [31]. [9] provides an interesting hardness result, suggesting that any graph kernel is not *complete*, i.e., having a full discriminating power, unless it is as hard to compute as to test graph isomorphism.

3. PRELIMINARIES

3.1 Problem Definition

Definition 1. Let Σ_V be the set of vertex label names, and Σ_E that for edge labels. An (undirected) **labeled graph** is a 3-tuple $G = \langle V, E, l\rangle$, where V is a set of all vertices, and $E = \{(v_i, v_j)|v_i, v_j \in V, i < j\}$ is a set of edges. $l : V \to \Sigma_V \cup \{null\}$ and $E \to \Sigma_E$ is a labeling function mapping vertices and edges to their labels. Note that a vertex v may have no label, i.e., $l(v) = null$.

In the rest of this paper, when modeling our data graphs, i.e., networks, we use symbol N instead of G. Although only undirected graphs are discussed, our work can be straightforwardly extended to directed graphs.

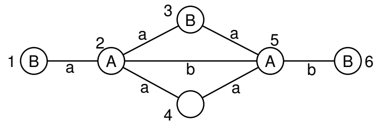
Given a network N with some vertices having no labels, following [24], we let $V^U = \{v|l(v) = null\}$ be the set of vertices to be classified, and $V^K = V \setminus V^U$ the set of vertices with known labels. Our task is to classify V^U into Σ_V based on the given information about V^U and N.

3.2 Mining Neighborhood Patterns

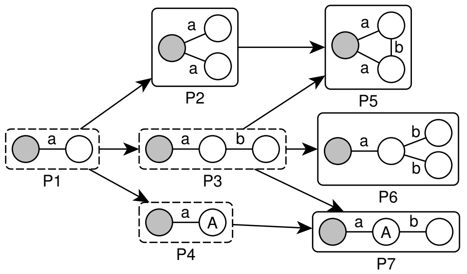
In this section, we revisit some key points of [11], which will be used in this paper.

Definition 2. A pivoted graph is a tuple $\mathcal{G} = \langle G, v_p\rangle$, where G is a labeled graph, and $v_p \in V(G)$ is a special vertex in G, called the *pivot* of $\mathcal{G}$.

A **neighborhood pattern** $\mathcal{P}$ is a connected pivoted graph, where the pivot points out the vertex whose neighborhood the whole pattern describes. For example, Figure 2b shows 7 neighborhood patterns (referred by IDs), where $\mathcal{P}_2$ stands for vertices associated with two a-edges. All non-null vertex labels and labeled edges are called **elements** of a pattern. The size of a pattern $\mathcal{P}$ is defined as $|\mathcal{P}| = |\{v|l(v) \neq null\}| + |E|$, i.e., the number of elements. For example, in Figure 2b, the sizes of both $\mathcal{P}_2$ and $\mathcal{P}_4$ are 2, respectively.



(a) A data graph (network). Numbers are IDs of vertices.



(b) Some neighborhood patterns of Figure 2a. Pivots are marked gray. Not all patterns are listed.

Figure 2: A network with some neighborhood patterns. We use uppercase letters for vertex labels and lowercase for edges. Null vertex labels are omitted.

Definition 3. A pivoted graph $\mathcal{G}_1$ is **pivoted subgraph isomorphic** to $\mathcal{G}_2$, denoted by $\mathcal{G}_1 \subseteq_p \mathcal{G}_2$, if there exists an injective mapping $f : V(\mathcal{G}_1) \to V(\mathcal{G}_2)$ such that: 1) $\forall v \in V(\mathcal{G}_1)$, $l(v) = null$ or $l(v) = l(f(v))$ (preserving vertex labels); 2) $\forall (v_1, v_2) \in E(\mathcal{G}_1)$, $l(v_1, v_2) = l(f(v_1), f(v_2))$ (preserving edge labels); 3) $f(v_p(\mathcal{G}_1)) = v_p(\mathcal{G}_2)$, where $v_p(\mathcal{G})$ is the pivot of $\mathcal{G}$ (mapping pivot to pivot).

Using pivoted subgraph isomorphism, the match between a neighborhood pattern and a vertex in the network can be defined. $\mathcal{P}$ **matches** vertex v of a network N, if $\mathcal{P}$ is pivoted subgraph isomorphic to the pivoted graph derived by making v the pivot in N, i.e., $\mathcal{P} \subseteq_p \langle N, v\rangle$. For instance, in Figure 2b, $\mathcal{P}_2$ matches v_2, v_3, v_4, and v_5 in Figure 2a, while $\mathcal{P}_5$ only matches v_3 and v_4. We can also use $\subseteq_p$ to define a partial order on all neighborhood patterns to describe the sub/super pattern relationship. A neighborhood pattern $\mathcal{P}$ is called a sub-pattern of $\mathcal{P}'$, if $\mathcal{P} \subseteq_p \mathcal{P}'$. For example, in Figure 2b, $\mathcal{P}_1$ and $\mathcal{P}_2$ are both sub-patterns of $\mathcal{P}_5$.

Definition 4. Given a vertex set $V_0 \subseteq V(N)$, the set of vertices in V_0 that $\mathcal{P}$ matches is $M_{V_0}(\mathcal{P}) = \{v \in V_0|\mathcal{P} \subseteq_p \langle N, v\rangle\}$. The support of $\mathcal{P}$ in V_0 is defined as the size of $M_{V_0}(\mathcal{P})$. $\mathcal{P}$ is a **frequent neighborhood pattern** of V_0, if its support is above a given threshold τ.

For instance, in Figure 2a, $M_V(\mathcal{P}_2) = \{v_2, v_3, v_4, v_5\}$. Therefore, the support of $\mathcal{P}_2$ in Figure 2a is 4. With the support measure defined, the FNM problem is simply finding all frequent neighborhood patterns, w.r.t. a given vertex set V_0 and a threshold τ.

Like conventional apriori-based algorithms, [11] generates and verifies patterns in the search space in a level-wise manner, where patterns are at levels corresponding to their sizes. After obtaining all frequent patterns at level k, we *join* each pair of patterns which share a size-$(k-1)$ sub-pattern to generate candidates to be verified at the next level. For instance, in Figure 2b, the join of $\mathcal{P}_2$ and $\mathcal{P}_3$ produces $\mathcal{P}_5$, and we get $\mathcal{P}_7$ if we join $\mathcal{P}_3$ and $\mathcal{P}_4$. Note that self-join is allowed. A join may produce multiple candidates. For example, the self-join of $\mathcal{P}_3$ produces $\mathcal{P}_5$ and $\mathcal{P}_6$.

Like most frequent pattern mining problems, e.g., frequent itemset mining [1] and frequent subgraph mining [14, 19, 34], FNM also satisfies the *anti-monotonicity* property, i.e., for any patterns $\mathcal{P} \subseteq_p \mathcal{P}'$, the support of $\mathcal{P}'$ does not exceed that of $\mathcal{P}$. For example, in Figure 2b, the support of $\mathcal{P}_2$ is 4, while for $\mathcal{P}_5$ it is 2. Given a threshold τ, say, 3, once $\mathcal{P}_5$ is found infrequent, we removes it to avoid checking its super-patterns. The pruning is both sound and efficient.

Although the anti-monotonicity property of FNM ensures a complete result set for apriori-based algorithms, we emphasize non-trivial *building blocks* as a particularity of FNM, which, if improperly treated, may harm the completeness of the results. We define *building blocks* as follows and give the following result:

Definition 5. A pattern is called a **building block**, if it *cannot* be obtained by joining smaller patterns at the previous level.

LEMMA 1. *A pattern $\mathcal{P}$ is a building block, if and only if there* do not *exist* <u>*two*</u> *elements in it, by removing* <u>*either*</u> *we obtain a valid, but smaller pattern $\mathcal{P}' \subseteq_p \mathcal{P}$.*

For example, in Figure 2b, building blocks are marked by dotted lines. $\mathcal{P}_3$ is a building block, because to obtain a connected sub-pattern of size 1, we have no other choice than removing the rightmost dangling edge b (the vertex on the right tail is simultaneously removed because this does not affect the size). $\mathcal{P}_4$ is also a building block because the only choice to shrink it is to remove the vertex label A.

When designing an apriori-based algorithm, it is vital to know the shape of building blocks in advance and guide the algorithm to treat them specially, because 1) they cannot be obtained by the join operation, 2) they themselves are part of the mining result, and 3) the absence of any of them may cause higher-level patterns depending on them to be missed. For conventional pattern mining problems, it is safe to trivially initiate with all level-1 patterns, because building blocks only appear at level-1. However, this is apparently not enough in the FNM case, since as shown in the above example, $\mathcal{P}_3$ and $\mathcal{P}_4$ are found to be non-trivial building blocks at level-2, and there are potentially more at the upper levels. For the FNM problem, where do building blocks appear, and what do they generally look like?

Interestingly, [11] formally proved that the building blocks of FNM are only limited to *path patterns* as defined below:

Definition 6. A neighborhood pattern is a **path pattern** if 1) it is a path of labeled edges, where the pivot is on one end of the path; 2) it contains at most one vertex label, which (if exists) must appear on the other end of the path.

As the search space of path patterns is tree-like, they are easy to be generated in advance. Therefore, the FNM algorithm follows the apriori framework, with an important difference. During initiation, the algorithm generates all frequent path patterns by "extending" rather than joining, then an apriori-based search is performed. Note that, the join operation at each level misses all building blocks belonging to the current level. Therefore, before moving on to the next level, the building blocks of appropriate size must be added to allow for larger patterns that depend on them. In the rest of this paper, for application-specific requirements, sometimes we use an additional parameter k to end the search at level-k to produce all patterns whose size are up to k.

4. NEIGHBORHOOD PATTERNS AS VERTEX FEATURES

In the literature of graph classification, there is a significant branch of research (e.g., [7]) which explores the effectiveness of frequent subgraph patterns as classification features. Given a collection of patterns S generated by a subgraph mining algorithm, they represent each graph g involved in the task as a vector $\mathbf{x}_g = (x_1, x_2, ..., x_j, ..., x_{|S|})$, where $x_j = 1$ if and only if g contains the j-th pattern as a subgraph. After all graphs are vectorized, common classification algorithms like SVM are performed to learn from graphs with known labels and classify the others.

Inspired by this, we are interested in an analogous question: can frequent neighborhood patterns [11] help classifying vertices in the setting of within-network classification? To the best of our knowledge, this problem is never studied in the literature. In this section, we introduce this approach by aligning it to the framework described in Section 2, and preview some of our experimental results which suggest that this approach is not only feasible but also competitive. We also identify and analyze a drawback of this approach, which motivates the second and third contributions of this paper.

4.1 Approach and Preliminary Results

According to the framework described in [24], a within-network classifier commonly consists of three key components: a local model, a relational model, and a collective inference procedure. To use neighborhood patterns to assist WNC tasks, we have to first specify the role of neighborhood patterns in such a framework, and appropriately instantiate the three components.

A local model exploits network-independent information on a vertex to provide the whole task with a good initial guess. In our method, we simply keep the null labels of unclassified vertices as the initial guess. We make this choice for three reasons: 1) a null label does not mean a cold start since the edge structure of the network carries information and is always available, 2) the study of how local classifiers affect the overall performance is rather orthogonal to that of the other two components, and 3) the principle of constructing local classifiers has been well studied in conventional machine learning literature.

In the relational model component, the neighborhood patterns actually take effect. Given V^K as training set, first, we run the FNM algorithm on V^K with appropriate parameters and obtain a list of patterns $\{\mathcal{P}_j\}, j = 1...m$. Each $\mathcal{P}_j$ describes the surrounding structures where a number of vertices reside[1]. Second, we represent each vertex v in V^K as

[1]Note that as "features", $\mathcal{P}_j$ must not contain any information about the label of its pivot vertex. This could be done by a post-filtration on the mining results. However, it is provably correct and efficient to achieve this by simply blocking those size-1 patterns consisting of only a pivot with a label from entering level-2.

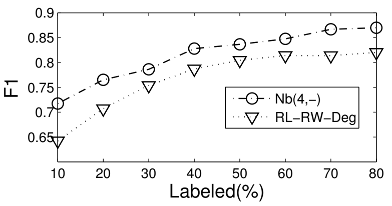


Figure 3: Performance of RL-RW-Deg and FNM (k=4) on the Protein dataset

an m-dimensional vector $\mathbf{x}_v = (x_1, x_2, ..., x_j, ..., x_m)$, where

$$x_j = \begin{cases} 1 & \text{if } \mathcal{P}_j \subseteq_p \langle N, v\rangle \\ 0 & \text{otherwise} \end{cases}$$

Clearly, the patterns $\mathcal{P}_j$ serve as vertex features for training the relational model. In the following parts of this paper, we will not distinguish "patterns" from "features" as they are essentially equivalent in our task. Third, we use the true label of v to form the training label, i.e., $y_v = l(v)$. Finally, we collect all $\langle \mathbf{x}, y\rangle$ pairs as training data to train the relational model $\mathcal{M}$.

Algorithm 1 Iterative Classification

```
Input: Network $N$, Neighborhood Features $\{\mathcal{P}_j\}$, relational model $\mathcal{M}$
Output: Predicted Labels $\mathbf{y}^U$.
 1: while true do
 2:     $\mathbf{x}^U \leftarrow \mathbf{1}(\mathcal{P}_j \subseteq_p \langle N, v_i^U\rangle)$ //$\mathbf{1}(p_{ij})$ is a 0-1 matrix, where the $(i,j)$-th element is 1 if and only if $p_{ij}$ is true.
 3:     $\mathbf{y}^U \leftarrow Classify(\mathbf{x}^U, \mathcal{M})$
 4:     if $\mathbf{y}^U$ converges or $n_{iter}$ is reached then
 5:         Break
 6:     else
 7:         Update $N$ with $\mathbf{y}^U$
 8:     end if
 9: end while
10: return $\mathbf{y}^U$
```

When doing inference, we adopt ICA [27] as the collective inference component because it is simple and effective. The pseudo code of our implementation is in Algorithm 1. Given an initial guess and a relational model, the algorithm performs multiple iterations over V^U. In each iteration, the features of every vertex v are reconstructed because the labels of vertices that v depends on might have changed. Then, the relational model is applied to provide a newer version of prediction. The algorithm stops when the result converges or a given number of iterations n_{iter} have been reached.

In practice, we find frequent neighborhood features quite effective. In Figure 3, we use the chemical structure completion task [8] to compete our approach with the baseline on testing networks of different label ratio $\frac{|V^K|}{|V|}$. We carry out 10-fold network cross-validation (further explained in Section 6.1) and all results are averaged over the 10 runs. It turns out that our method outperforms RL-RW-Deg by at least 11.7% in terms of weighted F1 score. Our features are powerful because they remember the exact structures rather than a statistics or an aggregation of the structure, which leads to a definite classification of the residing vertex. We will present more detailed experimental methodology, parameter setting, and results in Section 6.

	RL-RW-Deg	$r = 1$	$r \leq 2$	$r \leq 3$	$r \leq 4$
#Feature	-	906.2	4804.1	7370.7	7978.6
F1	0.804	0.824	0.834	0.836	0.836
Time (sec)	79.6	3.1	18.3	28.8	31.4

Table 1: Large-radius features bring negligible improvements. All improvements over the baseline are statistically significant under the $p < 0.01$ paired t-test.

4.2 Analysis of Feature Contribution

In Figure 3, our relational model is trained using all neighborhood features of size up to 4. Generally speaking, features of large sizes increase the expressivity of the relational model, enabling it to capture more semantics of the data. However, a large k allows many weak features to pass, causing the training & inferencing to be rather slow. Therefore, it is critical to explore techniques to prune weak features while keeping the remaining ones as expressive as possible.

For a pattern $\mathcal{P}$, there is no doubt that its size $|\mathcal{P}|$ characterizes some properties of itself. However, due to the particularity of neighborhood patterns, another characteristic indicator, called *radius*, also deserves to be emphasized. The radius of $\mathcal{P}$, denoted by $r(\mathcal{P})$, is defined as

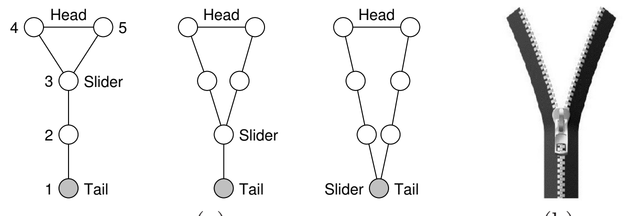
$$r(\mathcal{P}) = \max_{v \in V(\mathcal{P})} d(v, v_p)$$

where $d(u, v)$ is the distance between vertex u and v in $\mathcal{P}$.

A neighborhood pattern's radius has rich connections with several concepts in the network mining and classification literature. For example, in within-network classification [24] and graphical model [17], the first-order Markov assumption (or property) relates the label of a vertex to those of its direct neighbors. In [2], the *ego-net* (the induced graph generated by a vertex and all its neighbors) of a vertex v is used to fully represent the behavior of v. In essence, they all recognize a nature of real networks that a vertex's property tends not to be affected by distant vertices or edges.

Similarly, in our approach, a feature with a large radius tends to incorporate the influence from sub-structures faraway into the classification of a vertex, while small-radius features only consider information nearby. To investigate the contribution of features of different radius, we grouped all features in Figure 3 by their radius. We retrained the model with features of radius $\underline{\text{up to}}$ 1, 2, 3, and 4, respectively, applied it in the classification task, and report the feature number and performance (label rate = 50%) in the first two rows of Table 1. Two points are clearly observed from the table. First, under all feature combinations, our method outperforms RL-RW-Deg. Second, the classification performance in terms of weighted F1 almost stops increasing when we attempt to add features of $r = 3$ and 4 (weighted precision and recall also follow the same trend). This reveals the fact that, compared to small-radius features, features with a large radius tend to be relatively weak.

Since large-radius features are weak, did we spend much time on mining them? In the FNM algorithm, the time spent on patterns of different radius cannot be directly recorded, because at each level, patterns of all radius are mixed and mined simultaneously. However, the time spent and the distributions of patterns w.r.t. radius at each level are avail-



(a) (b)

Figure 4: (a) Zipper patterns with $r = 3$; (b) A real zipper

able. Therefore, we estimate the time cost w.r.t radius as follows. We ignore the time spent on building blocks because it makes up only a small proportion of the whole procedure. We also assume that at each level, time is uniformly amortized among patterns. For each level, we calculate the time cost w.r.t. radius using the time cost and the radius distribution at this level. Then, the radius-specified time cost at all levels are accumulated to obtain the time cost w.r.t radius. We transform the results to the time cost of different feature combinations and present them in the third row of Table 1. For comparison, we also attach the time RL-RW-Deg needs for a run. It is obvious from the estimation that we could have spent little time only on small-radius features to gain most of the potential performance.

5. HANDLING RADIUS CONSTRAINTS

The observation described in Section 4.2 indicates that, when applying neighborhood features to WNC, most of the classification performance is earned by features with a small radius. By contrast, features with relatively large radiuses cause large running time in the feature-mining stage and classification stage, but only bring a small effectiveness improvement. We can prune all large-radius features from the mined features to reduce the classification time. However, the time spent on mining those large-radius features cannot be taken back. When the size of training data is large and complicated, e.g., the network N is highly dense or large, the wasted computation is even bigger. This raises an interesting question: can we directly mine neighborhood features w.r.t. a given radius threshold r_{max}, without generating those whose radius are above r_{max}?

5.1 Building Blocks Again

Recall that the FNM algorithm without radius constraint is a two-stage algorithm. In the first stage, the algorithm searches for all frequent path patterns. In the second stage, a level-wise apriori-based search is performed, using all path patterns from the first stage as building blocks.

With the additional radius constraint at hand, let us consider modifying the original algorithm to produce a correct and efficient algorithm. One may thinks that we only need to generate paths whose length is no larger than r_{max}. However, this approach again misses some building blocks, thus resulting in an incomplete result set.

Take $r_{max} = 3$ as an example, i.e., the length of any path building block should not exceed 3. We consider the first neighborhood pattern in Figure 4a, where each vertex is associated with an ID for reference. Please note that the word "Head", "Slider", and "Tail" are not vertex or edge labels, and are only markers for future reference. Each edge is associated with a certain edge label, which is omitted to keep the figure clear. Intuitively, the size of this pattern is 5 and the radius is 3. Therefore, according to Lemma 1, if the pattern is not a building block, we can find <u>two</u> edges in the pattern, by removing <u>either</u> of them we obtain a valid pattern of size 4 and radius no larger than 3.

We consider removing the five edges individually. Removing edge $(1, 2)$ or $(2, 3)$ is not valid because it produces unconnected patterns. $(3, 4)$ or $(3, 5)$ are unremovable either, since the radius of the resulted pattern is 4. Edge $(4, 5)$ is the only choice. Therefore, we cannot find two removable edges, and the original pattern *is* actually a building block.

For $r_{max} = 3$, there are three types of non-path building blocks. In addition to the one discussed above, the other two are also shown in Figure 4a. If we arrange them in an appropriate order as in Figure 4a, from left to right they look like an opening zipper (Figure 4b), with the slider (the only vertex of degree 3, if it is not overlapped with the pivot) approaching the tail. Therefore, we call them *zipper patterns*.

Definition 7. A neighborhood pattern is a **zipper pattern** of radius r_{max} if 1) it has no vertex labels; 2) there exists exactly one special edge (we name it the "head"), after removing which the pattern is a rooted tree of depth r_{max}, with the pivot as the root; 3) the two ends of the "head" are the only two leaves of the rooted tree, and they both have depth r_{max}.

Interestingly, we can prove that for any r_{max}, all building blocks are either path patterns or zipper patterns.

THEOREM 1. *Given a FNM instance with a radius threshold r_{max}, the building blocks consist of 1) all path patterns whose length is no greater than r_{max}, and 2) r_{max} types of zipper patterns, whose size ranges from $r_{max}+2$ to $2r_{max}+1$.*

PROOF. It is easy to verify that path and zipper patterns are building blocks. Therefore we only prove that any building block is a path or zipper pattern. We prove by making step-wise restrictions and contradictions. Given a building block $\mathcal{P}$, we narrow down by proposing predicates on the structure of $\mathcal{P}$: if $\mathcal{P}$ violates the predicates there will be two removable elements, suggesting it is not a building block. For each case discussed below, we attach a figure in Figure 5 to illustrate the case. Finally readers will see that the predicates converges to the three conditions in Definition 7.

For all cases, $\mathcal{P}$ must have no more than 1 vertex label. Otherwise, denoting two of them as l_1 and l_2, it turns out that l_1 and l_2 are two removable elements.

For all cases, noticing that $r(\mathcal{P}) \leq r_{max}$, $\mathcal{P}$ must have a rooted spanning tree $\mathcal{P}_T$, whose root is at the pivot and depth is no greater than r_{max}. Throughout Figure 5, edges on $\mathcal{P}_T$ are marked bold. Let n_e be the number of edges of $\mathcal{P}$ not on $\mathcal{P}_T$. n_e must not exceed 1. Otherwise, denoting two of them as e_1 and e_2, e_1 and e_2 are two removable elements. Note that the removal of e_1 or e_2 does not increase the radius of $\mathcal{P}$. Next, we enumerate the possible values of n_e.

1. $n_e = 0$: in this case, $\mathcal{P} = \mathcal{P}_T$ is itself a rooted tree with at most one vertex label. This tree must have only one leaf. If there are two leaves, the two dangling edges they are associated with will be the two removable elements. In the case where a leaf carries the only vertex label, we only remove the label instead.

1.1. The tree with only one leaf now is actually a path. However, we still have to prove that the only vertex label,

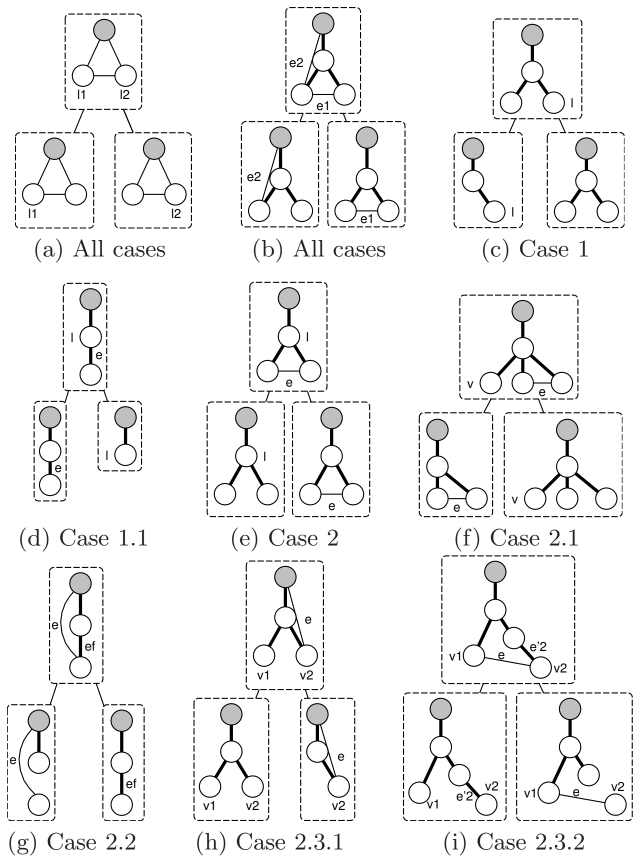


Figure 5: Cases in the proof of Theorem 1. Labels with no direct influence on the proof are omitted.

if exists, must be on the only leaf. If any vertex other than the leaf carries the label, the label and the dangling edge are removable. This branch of proof finally converges to the case of path patterns in Definition 6.

2. $n_e = 1$: we denote the only edge not on $\mathcal{P}_T$ as e. In this case, $\mathcal{P}$ must not contain any vertex label l. Otherwise, e and l are two removable elements. We denote the number of $\mathcal{P}_T$'s leaves as n_f, and enumerate all possible cases.

2.1. $n_f \geq 3$: $\mathcal{P}$ must have at least one dangling vertex v because e can consume two leaves on $\mathcal{P}_T$ at most. In this case, e and the dangling edge are removable.

2.2. $n_f = 1$: in this case, $\mathcal{P}_T$ is a path pattern. We denote the dangling edge associated with the only leaf as e_f. e and e_f are removable. Note that removing e_f does not violate the radius constraint.

2.3. $n_f = 2$: we denote the two leaves as v_1 and v_2. We consider two possible locations of e.

2.3.1. e is not between v_1 and v_2: w.l.o.g., we assume that v_1 is not associated to e, i.e., v_1 is a dangling vertex. In this case, e and the dangling edge are removable.

2.3.2. e links v_1 and v_2: in $\mathcal{P}_T$, v_1 and v_2 must both have depth r_{max}. Otherwise, we assume that v_1's depth is $d < r_{max}$. We denote the dangling edge associated with v_2 as e'_2. In this case, e and e'_2 are two removable elements. Note that in $\mathcal{P} \setminus \{e'_2\}$, the depth of v_2 is no greater than $(d+1) \leq r_{max}$ because it is linked to v_1 by e. This branch of proof finally converges to the definition of zipper patterns in Definition 7. □

5.2 Generating Zipper Patterns

In the last section, we graphically and formally describe the structure of zipper patterns for the radius-constrained FNM problem. Similar to the path patterns in the original FNM problem, the new zipper patterns cannot be generated by joining smaller ones, and need special treatment in algorithm design. Next, we present an approach.

Again we assume $r_{max} = 3$, and we find all zippers of the first type in Figure 4a, which belong to level-5. Suppose $\mathcal{P}$ is a frequent zipper pattern. According to the anti-monotonicity property of the mining problem, the "Y"-shaped pattern $\mathcal{P}_Y$ obtained by removing the head of $\mathcal{P}$ must also be frequent, thus appearing in the results at the 4-th level. It is relatively easy to identify them from the list of size-4 patterns. We only need to check whether a pattern is a tree, calculate the number of leaves, and find the depth of the leaves. Once a pattern $\mathcal{P}_Y$ is discovered, we find all its matching instances in the network N, and find all possible labeled edges appearing at the position of the head edge. Only those edge labels that lead to a frequent zipper pattern are assembled into $\mathcal{P}_Y$ to produce a valid $\mathcal{P}$. From now on, we call the radius-constrained version of neighborhood mining algorithm r-FNM.

5.3 Discussion

FNM and r-FNM share some similar optimizations as FSG [19] such as TID (transaction ID) lists and canonical labels. In our current implementation, we adopted the VID (vertex ID) lists optimization [11], which is similar to the TID [19] one under the subgraph mining scenario. VID lists not only reduce the mining time by avoiding unnecessary verifications of unpromising patterns, but also provide the IDs of vertices each pattern matches as a by-product output. This by-product is particularly useful to our classification task because the 0-1 matrix for training the relational model can be directly built with it.

Compared to the apriori-based breadth-first mining methods [14, 19], it is well recognized that depth-first methods, such as gSpan [34], achieve better performances in graph pattern mining tasks. The frequent neighborhood mining problem and its radius-constrained version are both solvable in a similar depth-first manner. We leave them as our future work.

6. EXPERIMENTS

In this section, we carry out the chemical structure completion task on three real-life datasets to empirically evaluate our approaches. A state-of-the-art structure-aware WNC method called RL-RW-Deg [8][2] was employed as the baseline. We aim to empirically answer the following **questions**: **1)** Does neighborhood features consistently outperform the state-of-the-art? **2)** Does the r-FNM algorithm save time as estimated in Table 1? **3)** Does the new parameter r_{max} enable an efficiency-effectiveness tradeoff? **4)** Compared with tuning k, does r_{max} provides a better tradeoff?

6.1 Experimental Terminology

We used the first three datasets in [8] (Table 2), namely Mutagenicity, AIDS, and Protein, provided in the IAM Graph

[2]The paper also proposed another method called RL-RW. However, it is used as a baseline of RL-RW-Deg.

	Mutag.	AIDS	Protein
Graph count	4337	2000	600
Avg. vertices	30.3	15.7	32.6
Avg. edges	30.8	16.2	62.1
$\|\Sigma_V\|$	14	38	3
$\|\Sigma_E\|$	3	3	5
Freq. class(%)	44.3	59.3	49.4

Table 2: Dataset and statistics

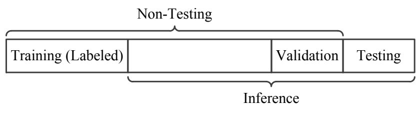


Figure 6: Partitioning data for network cross-validation

Database Repository[3]. We did not explore the other two datasets in [8] since they are homophily-based. Because the three datasets were originally introduced respectively as benchmarks for graph classification, we need to first convert them such that they can be utilized in our task.

We processed the three datasets in the same manner. For each dataset, we sampled a subset of graph transactions consisting of 100 molecules (or proteins). We merged all transactions into a network, where they were regarded as connected components of the network. We performed 10-fold *network cross-validation* [26] to generate training, validation, and testing sets as Figure 6 shows. Specifically, we first divided the vertex set into 10 folds. Each time we used one fold for testing, and merged the other nine as "non-testing set". We sampled V^K from the non-testing set as training set such that the label ratio $\frac{|V^K|}{|V|}$ ranged from 10% to 80%. In the non-testing set, we also sampled $10\%|V|$ vertices as the validation set, which was disjoint from the training set V^K. Following [26], we denote $V^U = V \setminus V^K$ as the "inference set", whose labels were removed and predicted in each classification run. However, in V^U, only vertices in the testing set were considered in evaluation.

In our methods, neighborhood features were mined on V^K with the minimum support[4] set to 2. We used SVM$^{\text{multiclass}}$ [33][5] with linear kernel to train models with neighborhood features. The parameter C of SVM, which controls the balance between empirical loss and regularization, was chosen from $\{10^{-1}, 10^0, ..., 10^5\}$ using ground truths on the validation set. For RL-RW-Deg, which does not involve a training phase, we used the validation set to tune its four parameters, which include $\alpha = \{0.5, 0.75, 1\}$, $\beta = \{1, 1.5, 2, 2.5, 3\}$, $N_{max} = \{2, 3\}$, $\gamma = \{0.1, 0.3, 0.5\}$. Readers can refer to [8] for the specific meaning of the parameters.

[3]`http://www.iam.unibe.ch/fki/databases/iam-graph-database`

[4]According to [6], an appropriate support threshold prevents features without statistical significance from causing over-fitting in training. However, currently we do not address techniques for automatically deciding the support threshold. Given that the linear kernel has a low risk of over-fitting and that an improperly high support threshold can potentially block useful features, we decided to take this safe choice and leave the study of this parameter for future work.

[5]`http://www.cs.cornell.edu/People/tj/svm_light/svm_multiclass.html`

When running the collective inferencing component in both methods, the number of iteration was set to 3 because it ensures convergence by observation. We evaluated the effectiveness of all methods using the metric of *weighted F1 score*, which is the average of the conventional F1 score of different classes weighted by their true distribution. Each reported F1 score was an average on all 10 testing runs.

All experiments were conducted on a PC with two Xeon 2.50GHz processors and 64GB memory. All algorithms were implemented in C# and run on a single core. We report all efficiency results in milliseconds.

6.2 Results

We report experimental results through answering the four questions raised at the beginning of Section 6.

Question 1: In Section 4.1 we mentioned that, on one of our datasets, neighborhood features of size below 4 build a strong relational model. Equipped with ICA [27] as collective inferencing component, our solution outperforms RL-RW-Deg by at least 11.7% in terms of weighted F1.

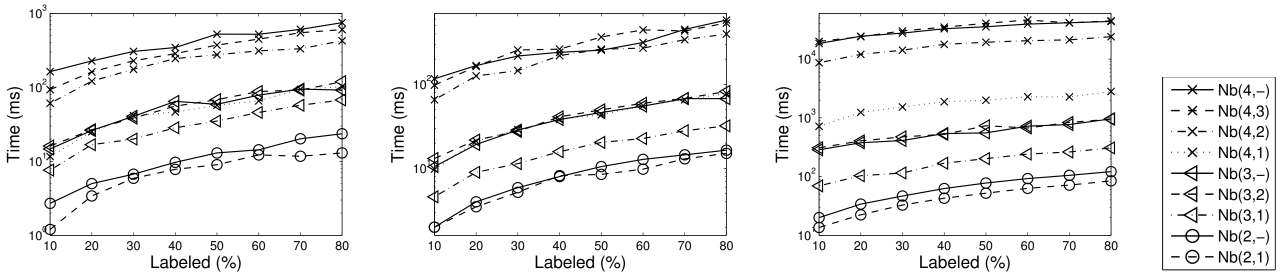
In fact, neighborhood-feature-based classifiers can compete with the baseline, even with weaker classes of features. In Table 3 we list the performance of RL-RW-Deg and neighborhood features of size up to 2, 3, and 4, respectively (denoted by $Nb(2, -)$ and so on). When $k = 1$, our features only consider the types of adjacent edges, which cannot exploit vertex label dependency. This case is theoretically trivial and empirically weak, and is thus not reported here. Clearly, as k grows, the classification performance of neighborhood features increases. Equipped with features of sizes up to 3 and 4 respectively, our methods consistently outperform the baseline (passed the $p < 0.01$ paired t-test). In the case of $k = 2$, the improvements are not statistically significant for some label ratios. Although there are two cases (marked bold) where our performance is worse in average, the performance difference is neither large nor statistically significant.

Question 2: In Table 1, we estimated the time spent on mining features of different radius by adopting some assumptions. When the r-FNM algorithm is applied, it is natural to ask, does it really save time as estimated, by not generating large-radius features? In Figure 7, we report the feature mining time on different datasets and under different parameter settings. Again we use the notation $Nb(k, -)$ to denote features mined using the original neighborhood mining algorithm FNM. We also use $Nb(k, r_{max})$ to denote features mined by the r-FNM algorithm described in Section 5. For example, $Nb(4, 2)$ stands for features generated by r-FNM using $k = 4$ and $r_{max} = 2$. All time costs are averaged over the 10 training sets.

From Figure 7 it is obvious that, when k is fixed, the feature mining time generally decreases as r_{max} decreases. This indicates that when we need to use r_{max} to filter the features by their radius, it is beneficial to call r-FNM with r_{max} instead of filtering the features after calling FNM. There are some exceptions, for example, in Figure 7b at ratio 50%, the time of $(4, 3)$ exceeds that of $(4, -)$. We note that the zipper generating component is actually an extra, though not significant, computation overhead of the entire task. In the cases where r_{max} approaches k (e.g., $r_{max} = k - 1$), r-FNM may spend some time dealing with many zippers, only to avoid generating a small number of large-radius patterns. However, this does not deteriorate the significance of r-FNM

Dataset	Method	Label Ratio(%)							
		10%	20%	30%	40%	50%	60%	70%	80%
Mutag.	Nb(4,-)	0.926	0.948	0.949	0.959	0.964	0.964	0.968	0.970
	Nb(3,-)	0.914	0.924	0.935	0.942	0.941	0.942	0.949	0.947
	Nb(2,-)	**0.882**‡	**0.882**‡	0.890‡	0.890‡	0.894‡	0.899	0.901†	0.902†
	RL-RW-Deg	0.888	0.884	0.881	0.878	0.886	0.880	0.885	0.886
AIDS	Nb(4,-)	0.645	0.671	0.691	0.702	0.709	0.735	0.768	0.779
	Nb(3,-)	0.631	0.668	0.681	0.699	0.711	0.720	0.729	0.739
	Nb(2,-)	0.607	0.625	0.655	0.672	0.668	0.672	0.659†	0.674
	RL-RW-Deg	0.561	0.555	0.574	0.589	0.567	0.567	0.601	0.623
Protein	Nb(4,-)	0.717	0.765	0.786	0.828	0.836	0.848	0.867	0.870
	Nb(3,-)	0.705	0.764	0.778	0.822	0.826	0.843	0.854	0.858
	Nb(2,-)	0.675	0.742	0.759‡	0.799‡	0.813‡	0.833†	0.835	0.846
	RL-RW-Deg	0.642	0.707	0.754	0.787	0.804	0.814	0.814	0.820

Table 3: Classification effectiveness in terms of weighted F1 score. **Bold texts** denote cases where our method did NOT outperform the baseline. Except these ‡ (failed the $p < 0.05$ test) and † (failed the $p < 0.01$ test but passed the $p < 0.05$ one), all improvements are statistically significant under the $p < 0.01$ paired t-test.



(a) Mutag (b) AIDS (c) Protein (d) Legend

Figure 7: Feature mining time of r-FNM and FNM under different parameter settings.

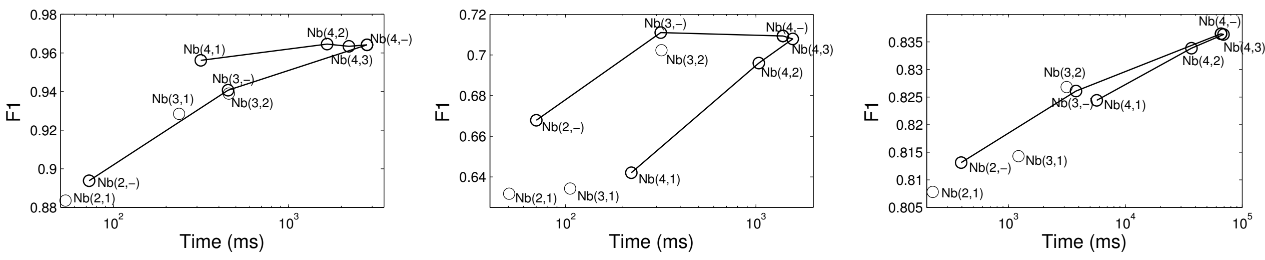
because no other obvious methods can ensure such efficiency when r_{max} takes small or medium values.

Question 3: Since r-FNM enables the choice of not generating large-radius features, how will the classification effectiveness and efficiency change when large-radius features are blocked by r_{max}? To study this issue, we fix the label ratio at 50%, and jointly plot the classification time (including feature mining and generating test data) and F1 scores of different parameter settings in Figure 8. We do not include RL-RW-Deg in the figures because it is generally several times slower than our methods, but only provides a similar performance as $Nb(2,-)$ does.

In Figure 8, we observe that if we connect the data points of $Nb(4,-)$, $Nb(4,3)$,$Nb(4,2)$ and $Nb(4,1)$ in order, we end up with a polyline roughly from upper right to lower left. Since the lower left direction represents the trend of worse performance and less time, it is clear that r-FNM provides a valid tradeoff of effectiveness and efficiency, by allowing specifying the maximum feature radius r_{max}. There may be exceptions when a polyline goes lower right at some point. For example, in Figure 8b, $Nb(4,3)$ has lower F1 score than $Nb(4,-)$, but costs more time. Recall that, when answering question 2, we mentioned that for large r_{max}, r-FNM may not be efficient. Besides these "bad" exceptions, we also observe "good" exceptions. For example, in Figure 8a, $Nb(4,2)$ is both more efficient and slightly more effective than $Nb(4,-)$.

Question 4: Since r_{max} encodes the Markov-assumption-based tradeoff, how about the other parameter k? Through reading Table 3 and Figure 7, it seems that k also provides means for an efficiency-effectiveness tradeoff. Generally speaking, k controls the complexity of the features generated by the neighborhood pattern miner, which is irrelevant from the Markov assumption. When a tradeoff is required, which parameter we should try first actually depends on the data and task.

In Figure 8, we also mark out the polyline $(4,-) \rightarrow (3,-) \rightarrow (2,-)$ to illustrate the tradeoff by varying k. We denote it as the k-line, and the previous one as the r_{max}-line. We observe that on the Mutagenicity dataset, the r_{max}-line is to the upper left of the k-line, indicating that on this dataset, the r_{max} parameter provides a better tradeoff than k. This is because the valency information of atoms on the Mutagenicity dataset serves as a strong cue in classification, which falls under the scope of first-order Markov assumption. However, the two lines exchanges their positions on the other two datasets. This is because the valency information is noisy on the second dataset (bonds to hydrogen atoms are omitted) and unavailable on the third one (vertices represent secondary structures, not atoms). The different results of applying the first-order Markov assumption indicate that, pre-knowledge about the data and task should be incorporated into the stage of choosing algorithms and parameters. In this sense, r-FNM does not always ensure a better tradeoff, but actually enables Markov-based assumptions to be adopted when necessary.



(a) Mutag (b) AIDS (c) Protein

Figure 8: Two ways of tradeoff on three datasets, with label ratio 50%.

7. CONCLUSION

In this paper, we proposed using neighborhood patterns of vertices to build relational classifiers for within-network classification. We experimentally showed that neighborhood features are not only feasible, but also competitive, in structure-dependent WNC tasks. Moreover, we formulated the problem of incorporating the Markov assumption into feature mining as the radius-constrained version of FNM. We algorithmically solved it by identifying zipper patterns as new building blocks of the pattern space. The new r-FNM algorithm was shown to help providing more effective choices on the efficiency-effectiveness trade-off in WNC tasks.

8. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, 1993.

[2] L. Akoglu, M. McGlohon, and C. Faloutsos. oddball: Spotting anomalies in weighted graphs. In *PAKDD (2)*, 2010.

[3] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *ICDM*, 2005.

[4] G. Casella and E. I. George. Explaining the gibbs sampler. *The American Statistician*, 46(3), 1992.

[5] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD*, 1998.

[6] H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *ICDE*, 2007.

[7] M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. In *ICDM*, 2003.

[8] C. Desrosiers and G. Karypis. Within-network classification using local structure similarity. In *ECML/PKDD (1)*, 2009.

[9] T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *COLT*, 2003.

[10] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, 15(1), 2007.

[11] J. Han and J.-R. Wen. Mining frequent neighborhood patterns in a large labeled graph. In *CIKM*, 2013.

[12] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos. It's who you know: graph mining using recursive structural features. In *KDD*, 2011.

[13] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD*, 2004.

[14] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, 2000.

[15] D. Jensen, J. Neville, and B. Gallagher. Why collective inference improves relational classification. In *KDD*, 2004.

[16] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, 2003.

[17] D. Kollar and N. Friedman. *Probabilistic graphical models: principles and techniques.* The MIT Press, 2009.

[18] X. Kong and P. S. Yu. Semi-supervised feature selection for graph classification. In *KDD*, 2010.

[19] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *Knowledge and Data Engineering*, 16(9), 2004.

[20] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, 11(3), 2005.

[21] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD*, 1998.

[22] Q. Lu and L. Getoor. Link-based classification. In *ICML*, 2003.

[23] S. A. Macskassy and F. Provost. A simple relational classifier. In *Proc. of the 2nd Workshop on Multi-Relational Data Mining (MRDM) at KDD*, 2003.

[24] S. A. Macskassy and F. J. Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8, 2007.

[25] M. Neumann, R. Garnett, and K. Kersting. Coinciding walk kernels: Parallel absorbing random walks for learning with graphs and few labels. In *Asian Conference on Machine Learning* 2013.

[26] J. Neville, B. Gallagher, T. Eliassi-Rad, and T. Wang. Correcting evaluation bias of relational classifiers with network cross validation. *Knowl. Inf. Syst.*, 30(1), 2012.

[27] J. Neville and D. Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000.

[28] C. Perlich and F. J. Provost. Distribution-based aggregation for relational learning with identifier attributes. *Machine Learning*, 62(1-2), 2006.

[29] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3), 2008.

[30] N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In *NIPS*, 2009.

[31] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, 2009.

[32] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. J. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *SDM*, 2009.

[33] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, 2004.

[34] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, 2002.

[35] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, 2004.