Inexact Graph Matching: A Case of Study

Ivan Olmos, Jesus A. Gonzalez
Instituto Nacional de Astrofísica, Óptica y Electrónica,
Luis Enrique Erro No. 1, Sta. María Tonantzintla, Puebla, México
{iolmos,jagonzalez}@inaoep.mx

Mauricio Osorio
Universidad de las Américas Puebla
Sta. Catarina Mártir, Puebla, México
josorio@mail.udlap.mx

Abstract

Inexact graph matching has become an important research area because it is used to find similarities among objects in several real domains such as chemical and biological compounds. Let G and G' be input labeled graphs, we present an algorithm capable to find a graph S of G, where S is isomorphic to G' and the corresponding labels between the vertices and edges of S and G' are not the same (inexact matching). We use a list-code based representation without candidate generation, where a step by step expansion is implemented. The proposed approach is suitable to work with directed and undirected graphs. We conducted a set of experiments in a genome database in order to show the effectiveness of our algorithm. Our experiments show a promissing method to be used with scalable graph matching tools that can be applied to areas such as Machine Learning (ML) and Data Mining (DM).

Introduction

Graphs are a powerful and flexible knowledge representation used to model simple and complex structured domains (Cook & Holder 1994). The representation power and flexibility is the main advantage of why the graph-based representation model has been adopted by researchers in different areas such as ML and DM (Cook & Holder 1994; Kuramochi & Karypis 2002). An important problem in ML and DM is to find similarities between objects. If we use a graph-based representation, the problem turns into finding similarities between graphs, which includes tasks as exact and inexact matching, where the graph / subgraph isomorphism detection is a critical operation (Cook & Holder 1994). The task is not easy, because the subgraph isomorphism problem is known to be in NP-complete (Michael & David 2003) then, in the worst case, the time to solve the decision problem is exponential, unless P=NP.

The exact matching (two graphs are similar if its topology and labeling is identical) is a widespread studied problem, where several works have been developed, each of them with different objectives. For example, Subdue (Cook & Holder 1994) is an algorithm that implements a computationally-constrained beam search, however the algorithm may not always find an isomorphism when it does exist (but it is capable to work the inexact problem). Some algorithms reduce the computational complexity by imposing topological restrictions on the input graphs (Luks 1982). There are other subgraph isomorphism projects such as Ullman (Ullman 1976), VF2 (Cordella *et al.* 2001) and Nauty (Brendan 1981) that are not able to work with labeled graphs, because many of them are oriented only to solve mathematical problems, and do not consider other classes of problems where labels represent important information. There are other works that explore ideas where the completeness is not sacrificed. AGM (Inokuchi & Washio 2003), FSG (Kuramochi & Karypis 2002), gSpan (Xifeng & Han 2002) and SI-COBRA (Olmos, Gonzalez, & Osorio 2005) are some algorithms that make use of strategies and representations with the aims to reduce the number of operations to perform and then being more efficient.

On the other hand, inexact graph matching is an important graph-theoretical problem, because it is used to find inexact similarities between objects. The inexact matching task consists on finding a distortion or variation between two input graphs, where there may not exist an exact match (Cook & Holder 1994; Hlaoui & Wang 2002; Cordella *et al.* 1996). Throughout this work, we consider an inexact match between two graphs in the sense that they have an identical topology (there exists a bijection between the vertices and edges of the graphs), nevertheless the labels of the vertices and edges might not be the same.

In this work, we present an algorithm capable of finding a graph S of G, where S is isomorphic to G' and the corresponding labels between the vertices and edges of S and G' are not the same (inexact matching). We use a list-code based representation without candidate generation, where a step by step expansion with an exploration in depth is implemented. The proposed approach is suitable to work with directed and undirected graphs. We conducted a set of experiments in genome databases in collaboration with biology experts in order to study the effectiveness of our method (our method has already been applied to other theoretical and practical domains). Our experimental results show a promissing method to be used with scalable graph matching tools that can be applied to research areas such as Machine Learning (ML) and Data Mining (DM).

The Subgraph Isomorphism Problem

As mentioned before, graphs have been used in several research areas. Different authors define a graph with some variations, according to their requirements. In this work, we assume that a ***graph*** is a 6-tuple $G = (V, E, L_V, L_E, \alpha, \beta)$, where:

- $V = \{v_i | i = 1, \ldots, n\}$, is the finite set of vertices, $V \neq \emptyset$
- $E \subseteq VxV$, is the finite set of edges, $E = \{e = \{v_i, v_j\} | v_i, v_j \in V\}$
- L_V, is a set of vertex labels
- L_E, is a set of edge labels
- $\alpha : V \rightarrow L_V$, is a function assigning labels to the vertices
- $\beta : E \rightarrow L_E$, is a function assigning labels to the edges

Let G be a graph, where $G = (V, E, L_V, L_E, \alpha, \beta)$. A ***subgraph*** S of G, denoted by $S \subseteq G$, $S = \{V^S, E^S, L_V^S, L_E^S, \alpha^S, \beta^S\}$ is a graph such that $V^S \subseteq V$, $E^S \subseteq E$, $\alpha^S \subseteq \alpha$ and $\beta^S \subseteq \beta$.

Given two graphs $G' = (V', E', L'_V, L'_E, \alpha', \beta')$ and $G = (V, E, L_V, L_E, \alpha, \beta)$, G' is ***isomorphic*** to G, denoted as $G' \cong G$, if there exist $f : V' \rightarrow V$ and $g : E' \rightarrow E$ as bijections, where:

- $\forall v' \in V'$, $\alpha'(v') = \alpha(f(v'))$
- $\forall \{v'_i, v'_j\} \in E'$, $\beta'(\{v'_i, v'_j\}) = \beta(g(\{v'_i, v'_j\}))$

This definition only applies for exact matching. For inexact graph matching, bijection functions of G' are defined in a different way. Let $G' = (V', E', L'_V, L'_E, \alpha', \beta')$ be a graph, where V', E', L'_V and L'_E are defined as we previously described. On the other hand:

- $\alpha' : V' \rightarrow 2^{L'_V}$, power set of L'_V, where $\forall v'_i, v'_j \in V'$: $\alpha'(v_i) \cap \alpha'(v_j) = \varnothing$ if $\alpha'(v_i) \neq \alpha'(v_j)$
- $\beta' : E' \rightarrow 2^{L'_E}$, power set of L'_E, where $\forall e'_i, e'_j \in E'$: $\alpha'(e_i) \cap \alpha'(e_j) = \varnothing$ if $\alpha'(e_i) \neq \alpha'(e_j)$

Considering the above mentioned, G' is ***inexact isomorphic*** to G, denoted as $G' \cong_I G$, if there exist $f_I : V' \rightarrow V$ and $g_I : E' \rightarrow E$ as bijections, where:

- $\forall v' \in V'$, $f_I(v') = v : \alpha(v) \in \alpha'(v')$
- $\forall e' \in E'$, $g_I(e') = e : \beta(e) \in \beta'(e')$

Based on these concepts, we say that G' is a ***subgraph isomorphic*** of G if there exists $S \subseteq G$ such that $G' \cong S$ for exact matching, or $G' \cong_I S$ for inexact matching.

In other words, an isomorphism between G' and G exists if the topology of both graphs is exactly the same and the labeling is identical for exact match. Note that if $|L'_V| = |L'_E| = 1$, then we have the traditional SI problem. It is clear that we are working an NP - complete problem, where some instances can fall in the worst case, but not all of them.

The IGM-COBRA Algorithm

In a previous work (Olmos, Gonzalez, & Osorio 2005), we developed an algorithm to detect the exact instances of a graph G' in a graph G (called Subgraph Isomorphism - COde Based Representation Algorithm, or SI-COBRA),

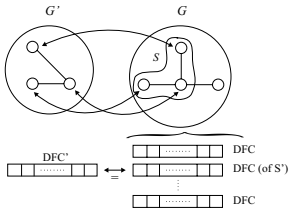

Figure 1: Finding a Subgraph Isomorphism Based on Lists of Codes

where a linear sequence of codes is used to represent the graphs. A code, denoted by c_i, represents the information of an edge label e and its adjacent vertices (a code based representation has also been successfully applied in gSpan (Xifeng & Han 2002)). Each code c_i is sorted in a linear sequence called $L_{VEV} = \{c_i : i = 1, \ldots, s\}$, where s is the number of different combinations of labels. A new lexicographic order is implemented using criteria such as vertices degrees, statistical summaries of the codes and an order based on the labels.

Our method starts by building a graph model of G', represented by a linear sequence of codes, called DFC'. The DFC' model is a sorted sequence of codes $< dfc'_1, \ldots, dfc'_n >$, where each entry $dfc'_x = (i, j, c_k, t)$ and i, j are the indexes associated to the adjacent vertices of edge $e_x \in E'$, c_k is a code that represents the label's information of the edge $e_x, c_k = L_{VEV}(e_x)$ and t is a special mark to classify the edges as $F(forward)$ or $B(backward)$ edges (Olmos, Gonzalez, & Osorio 2005).

We use three basic criteria to sort the DFC' sequence: a) the degree of the vertices; b) the label that has the largest number of instances and c) the combination of the labels (lexicographic order based on L_{VEV}). With these restrictions and using a DFS strategy, the DFC' code is built as follows:

1. Select a non visited vertex v_i in G' (v_i satisfying restrictions a), b) and c)
2. Expand vertex v_i to vertex v_j, (v_j has not been visited yet and satisfies restrictions a), b) and c)
3. Add $(i, j, L_{VEV}(\{v_i, v_j\}), F)$ to DFC' (a forward edge)
4. Expand all possible backward edges: $\forall \{v_j, v_x\}$ where v_x has already been visited, add $(j, x, L_{VEV}(\{v_j, v_x\}), B)$ to DFC' according to the L_{VEV} order
5. If v_j has an adjacent vertex v_x that has not been visited yet, go to step 2 with v_j as v_i and v_x as v_j
6. If there are vertices that have not been visited yet, go to step 1. In other case, finish

Based on DFC' (the model of G'), the matching process tries to build a linear sequence DFC of G. The size of DFC must be the same of DFC' and the corresponding codes entries must also be identical. If DFC could be generated, then we found $S \subseteq G$, where S is represented by DFC and $G' \cong S$. This idea is shown in Fig. 1.

In the matching proces, DFC is built with a step by step expansion without candidate's generation. First, we ap-

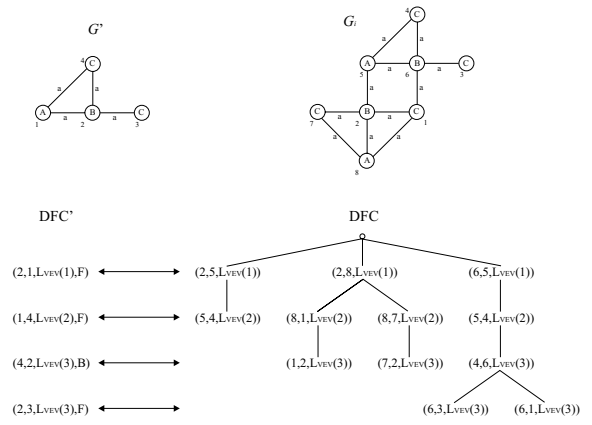

Figure 2: Example of a Width-Depth Search using L_{VEV} Codes

ply a pruning phase in G that aims to reduce the number of operations to perform, where $\forall v \in V : \alpha(v) \notin L'_V$ and $\forall e \in E : \beta(e) \notin L_E$ are removed. This task can be performed with the use of L_{VEV}, removing $\forall e \in G : L_{VEV}(e) \notin L_{VEV}$. Moreover, with a statistical analysis based on L_{VEV}, we can remove edges in G without the minimum number of repetitions. We also use the vertices degrees and their number of repetitions in the graph to further prune G. Without loosing generality, consider that $deg_{min}(\alpha_1), \ldots, deg_{min}(\alpha_n)$ are the minimum degrees associated to each label $\alpha_x \in L'_V$. Then, $\forall v \in V$ can not be common if $deg(\alpha(v)) < deg_{min}(\alpha')$ and $\alpha(v) = \alpha'$ (FSG uses a similar concept to order the vertices in an adjacency matrix). Note that after this preprocessing phase, G might have been partitioned in s connected graphs. Each of those graphs will be compared with G'. Let $GSet$ be the set of graphs derived from this process.

Next, the processing phase consists on finding the mapping between vertices and edges of G' and G_i, where $G_i \in GSet$ (the process is applied to each graph in $GSet$, while a DFC code has not been found). We implement a backtracking algorithm, because this technique is fairly stable and performs well in most cases, since it does not require more resources than those strictly necessary and a new partial result is based on a previous result. Since DFC' is an array, the backtracking strategy can be used.

The process starts by finding the vertices $v \in G_i$ where $\alpha(v) = \alpha(v_i)$, $v_i \in G'$, $deg(v) \geq deg(v_i)$, and $dfc'_1 = (i, j, c_x, F)$ where $c_x = L_{VEV}(\{v_i, v_j\})$ (each vertex forms a root of expansion). Taking these vertices, the construction process of DFC begins, where the algorithm finds all the possible mappings that can be associated to entry dfc'_i of DFC', that is, each edge where the code c_x is the same. This process is performed with a step by step expansion without candidate generation, where the L_{VEV} values are used to compare the mappings. Moreover, for each expansion step where there does not exist the minimun number of L_{VEV} combinations we pruned that expansion path and we avoid exploring these combinations.

Fig. 2 shows an example based on the L_{VEV} codes and a width-depth search strategy. In this example, we can see how the partial result vectors are generated and pruned. It is also possible to find all the instances of a graph G' in a graph G, just by leaving the algorithm run over all the possibilities. In this example, there are two mappings.

However, the before mentioned concepts are oriented to find exact associations between graphs, because for each $dfc'_x \in DFC'$ and $dfc_x \in DFC$, $c'_x = c_x$. With the aims to identify graphs with an identical topology but where the vertices and edges labels may vary (inexact matching), we propose the IGM-COBRA (Inexact Graph Matching) algorithm based on the following:

1. A new label l_x is associated to each $\alpha'(v_i)$ where $|\alpha'(v_i)| > 1$ (if two or more vertices in G' have the same $\alpha'(v_i)$, then they will have the same label l_x). This sustitution process will also be applied to edge labels.
2. L_{VEV} codes are built based on G' labels (considering the new l_x labels).
3. The DFC' sequence is built following the rules described in the SI-COBRA algorithm description.
4. The prunning phase of G is now modified. Vertices and edges are removed only considering the degree of the vertices. We are not using the L_{VEV} codes for prunning because there might exist codes of G that are not in L_{VEV} but, may produce a valid association.
5. G is explored in a similar way as in the SI-COBRA algorithm, with a step by step expansion in depth without candidate generation. However, there exists a variaton: we perform the degree test without considering the L_{VEV} codes.
6. Each dfc'_x entry is processed in two phases: if dfc'_x contains a l_x label, then it is necessary to consider each possible label that can produce a valid mapping. On the other hand, the comparison process is based on L_{VEV} codes.

Clearly, the IGM-COBRA algorithm can not prune a high number of path expansions as SI-COBRA does. However, this is a consecuence of the inexact maching process, where a label in G' can take different values. For example, let dfc'_x be a code, where $dfc'_x = (i, j, c_k, t)$, $c_k = (a, l_s, b)$ and $l_s = \{c, d\}$. Consider that there exist two edges in G, where their codes are $c_m = (a, c, b)$ and $c_n = (a, d, b)$. Clearly, c_m and c_n are not equal to c_k. Nevertheless, c_m and c_n can be associated to c_k, because the last one produces two label combinations that are identical to the c_m and c_n codes. For this reason, if we are processing an entry with a label l_x then we need to test each possible combination to detect valid mappings.

Experimental Results

We conducted an experimental investigation in order to find low-complexity DNA sequences using the IGM-COBRA algorithm. For our experiments, we worked with the Aspergillus nidulans, Neurospora crassa and Ustilago maydis DNA databases, where each database is a sequence that consists of the four nitrogen-compound bases a (adenine), c (cytosine), g (guanine) and t (thymine). These databases are

Organism	Database	Dimension (MBytes)	#Scaffolds
Aspergillus nidulans	http://www.broad.itm.edu/cig-bin/annotation/fungi/aspergillus/download_licence.cgi/aspergillus_nidulans_1.fasta.gz	30.5	248
Neurospora crassa	http://www.broad.mit.edu/cgi-bin/annotation/fungi/neurospora/download_licence.cgi/neurospora_3.fasta.gz	39.8	251
Ustillago maydis	http://www.broad.mit.edu/cgi-bin/annotation/fungi/ustilago_maydis/download_license.cgi/ustilago_maydis_1.fasta.gz	19.9	341

Figure 3: DNA Description Databases

Figure 4: Example of a DNA Sequence using a Graph-based Representation

divided in sections, called scaffolds. These DNA databases are free available through the Center for Genome Research (see figure 3) .

We represent a DNA sequence as follows. Given the DNA ordered sequence $\rho =< \sigma_1, \ldots, \sigma_n >$, $\sigma_x \epsilon \{a, c, g, t\}$ ($a$denine, $c$ytosine, $g$uanine, $t$hymine), then the graph-based representation for ρ is the graph $G = (V, E, L_V, L_E, \alpha, \beta)$ where:

- $V = \{v_x : x = 1, \ldots, n\}$ where $\exists$ a bijective function $\Phi : \rho \rightarrow V$
- $E = \{e_x : x = 1, \ldots, n-1, e_x = (v_i, v_{i+1}) \forall e_x\}$
- $L_V = \{a, g, c, t\}$
- $L_E = \{next\}$
- $\alpha : V \rightarrow L_V$
- $\beta : E \rightarrow L_E$

In our representation we map every base in the sequence to a vertex in the graph with the name of the base as the vertex label. In order to keep the sequence we add edges that connect vertices with the "next" label. For example, for the DNA sequence "cagctgcag" we create the graph shown in figure 4.

The low-complexity sequences $\rho' =< \sigma'_1, \ldots, \sigma'_{n'} >$ are represented with a graph $G' = (V', E', L'_V, L'_E, \alpha', \beta')$ defined as follows:

- $V' = \{v'_x : x = 1, \ldots, n'\}$ where $\exists$ a biyective function $\Phi' : \rho' \rightarrow V'$
- $E' = \{e'_x : x = 1, \ldots, n'-1, \forall e'_x : e'_x = (v'_i, v'_{i+1})\}$
- $L'_{V'} = \{a, g, c, t, x\}$
- $L'_{E'} = \{next\}$
- $\alpha' : V' \rightarrow L_{V'}$
- $\beta' : E' \rightarrow L_{E'}$

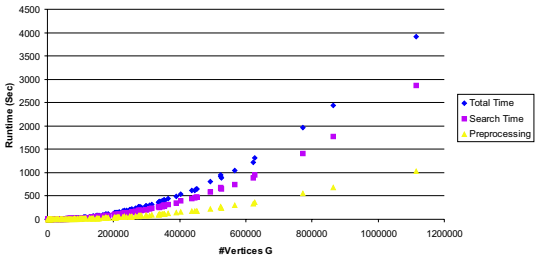

Figure 5: Runtime for the Aspergillus nidulans DNA Experiments

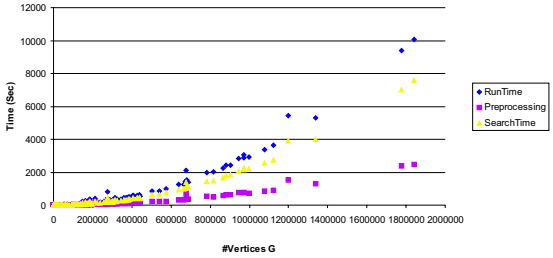

Figure 6: Runtime for the Neurospora crassa DNA Experiments

Note that $L'_{V'}$ includes a label x, where $x = \{a, t\}$. Then, the problem to find a low-complexity sequence in a DNA sequence using a graph-based representation is defined as follows: if G' represent the low-complexity sequence ρ' and G the DNA sequence, then the problem to find ρ' in the DNA sequence is reduced to find a subgraph $S \subseteq G$ such that $G' \cong_I S$.

For our experiments we transformed each database into its graph-based representation. After this, our domain experts suggested to look the cxg_m sequence, where x can be interchanged by any of the two bases (a or t) and $10 \leq m \leq 60$ (that is, m represents the consecutive number of repetitions of the sequence, for example, $cxg_3 = cxgcxgcxg$). In order to find the sequences in the genome, we ran the IGM-COBRA algorithm for each scaffold in its graph-based representation. Since we need to find each instance, we just leave the algorithm run over all the possibilities.

Figs. 5, 6 and 7 show the runtime execution performance of IGM-COBRA. These runtimes are divided in three sections: pre-processing time, which includes the reading input file and prunning phases; searching time, where G is explored and total time, that includes the pre-processing time and the searching time.

Note that the running time increases according to the dimension of the DNA inputs and the behavior was consistent for all the databases (in other words, the running time was not exponential as it could be thought). Moreover, the pre-processing time is not expensive (considering that in this phase all codes are built). These are positive characteristics

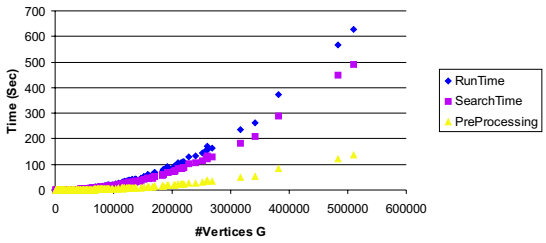

Figure 7: Runtime for the Ustilago maydis DNA Experiments

	G		G after Pruned		#Partitions	RunTime	Pruned Proportion	
	#Vertices	#Edges	#Vertices	#Edges			Vertices	Edges
Aspergillus	2259	2258	418	263	155	0.016	81.50%	88.35%
	10851	10850	1996	1337	659	0.14	81.61%	87.68%
	168814	168813	24466	15615	8851	90.563	85.51%	90.75%
	127977	127976	18991	11965	7026	50.672	85.16%	90.65%
	155056	155055	22901	14368	8533	75.453	85.23%	90.73%
Neurospora	18603	18602	2402	1498	904	0.25	87.09%	91.95%
	1842096	1842095	259549	163529	96020	10075.359	85.91%	91.12%
	1776986	1776985	247684	154847	92837	9375.297	86.06%	91.29%
	1337370	1337369	185798	116185	69613	5306.234	86.11%	91.31%
	1198271	1198270	199494	131579	67915	5457.328	83.35%	89.02%
Ustillago	3049	3048	333	202	131	0.031	89.08%	93.37%
	30180	30179	3819	2328	1491	0.906	87.35%	92.29%
	510685	510684	58011	34757	23254	626.703	88.64%	93.19%
	86435	86434	10124	6092	4032	16.234	88.29%	92.95%
	82161	82160	9459	5696	3763	14.344	88.49%	93.07%

Figure 8: Results in three DNA Databases

of the IGM-COBRA algorithm showing a high performance.

Figure 8 shows five scaffold results (randomly selected) of each DNA database, where we can see the original G graphs dimension (G columns) and their dimension after the pruning phase. We also illustrated the number of partitions induced by the algorithm (#Partitions column), the runtime (RunTime column) and the vertices and edges proportion (percentage) pruned by the algorithm from G (Pruned Proportion column). It is interesting to note the high number of partitions induced by the algorithm. As a consequence, some of those graphs can be pruned (eliminating the whole graph) because they are smaller than the graph to search for. Another important consequence is the fact that the algorithm requires less computational resources to store and process the data, because only one partition is processed at a time, and this requires less memory than that used to process the original database. We can also see in the last two columns the effectiveness of the pruning phase because a high proportion of vertices and edges were eliminated.

The number of discovered sequences was variable. For example, we found 16464 sequences in the Ustillago maydis scaffold $1\_29 - 2$, (some of them are shown in Fig. 9). The dimension of the found sequences is also variable, for example, in Fig. 10 we show the dimension of some sequences according with its start possition (these sequences belong to Linkage Group 1, which is a classification defined in the domain).

According with our results, our algorithm was capable to finding sequences of different dimension in polynomial

Dimension	Start Position	End Position	Sequence
12	77142	77153	cagctgcagcag
12	82646	82657	cagcagctgctg
12	85520	85531	cagcagcagcag
12	85544	85555	cagcagcagcag
12	104554	104565	cagcagctgctg
12	155329	155340	cagcagcagctg
12	322568	322579	ctgctgctgctg
15	105088	105102	ctgctgctgctgctg
15	268708	268722	cagcagcagcagcag
18	112404	112421	cagcagcagcagcagcag
18	286708	286725	cagcagcagcagcagctg
21	77668	77688	ctgctgctgctgctgctgctg
21	121449	121469	ctgctgctgctgctgctgctg
24	143567	143590	ctgctgctgctgctgctgctgctg
39	139246	139284	ctgctgctgctgctgctgctgctgctgctgctgctgctgctg
57	160502	160558	ctgctgctgctgctgctgctgctgctgctgctgctgctgctgctgctgctgctgctgctg

Figure 9: Some Sequences Discovered in Ustillago maydis.

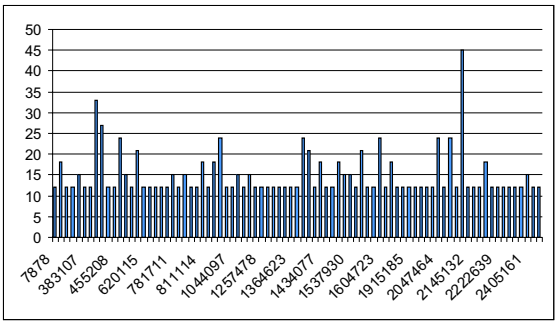

Figure 10: Example of Different Sequences Founded in Linkage Group 1 of Ustillago maydis.

time. Moreover, the response time was bounded by a quadratic polynomial with respect to the number of vertices.

Conclusions and Future Work

In this work we presented an approach to solve the inexact subgraph isomorphism problem. We implement concepts like a list code based representation, a step by step expansion model without candidate generation and a prunning phase. In our experiments we use a DNA domain, with three different databases, with the aims to get a good measure of the proposed algorithm. Our experiments showed that our approach finds the mappings very quickly. Consequently, the global performance is attractive. We also tested the scalability of the algorithm, since the dimension of the databases is considerable. Currently, we are working with our domain experts to study how possible low-complexity sequences can be found with IGM-COBRA.

We will continue our research testing our approach with other algorithms focused to solve the inexact match problem and also, we will do experiments using graphs with different dimensions and topologies. Moreover, we are analysing new problems where it is possible to use the IGM-COBRA algorithm to find inexact instances.

Acknowledgments

We thank our domain experts Patricia Sanchez and Candelario Vazquez for their collaboration in the experiments guidance and validation.

References

Brendan, D. M. 1981. Practial graph isomorphism. *Congressus Numerantium* 30:45–87.

Cook, D. J., and Holder, L. B. 1994. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research* (1):231–255.

Cordella, L. P.; Foggia, P.; Sansone, C.; and Vento, M. 1996. An efficient algorithm for the inexact matching of arg graphs using a contextual transformational model. *Proceedings of the International Conference on Pattern Recognition* 7276:180.

Cordella, L. P.; Foggia, P.; Sansone, C.; and Vento, M. 2001. An improved algorithm for matching large graphs. *Proceedings of the 3rd IAPR-TC15 Workshop on Graph-Based Representations in Pattern Recognition*.

Hlaoui, A., and Wang, S. 2002. A new algorithm for inexact graph matching. *16th International Conference on Pattern Recognition*.

Inokuchi, A., and Washio, T. M. 2003. Complete mining of frequent pattern from graphs: Mining graph data. *Machine Learning, Kluwer Academic Publishers* 321–354.

Kuramochi, M., and Karypis. 2002. An efficient algorithm for discovering frequent subgraphs. *Tech. Report. Dept. of Computing Science, University of Minnesota*.

Luks, E. 1982. Isomoprhism of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences* 25:42–65.

Michael, G. R., and David, J. S. 2003. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

Olmos, I.; Gonzalez, J. A.; and Osorio, M. 2005. Subgraph isomorphism detection using a code based representation. *Proceedings of the 18th International FLAIRS Conference*.

Ullman, R. J. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM* 23(1):31–42.

Xifeng, Y., and Han, J. 2002. gspan: Graph - based substructure pattern mining. *Technical Report, University of Illinois*.