

RelaxGrep: Approximate graph searching by query relaxation

Vincenzo Bonnici¹, Alfredo Ferro¹, Rosalba Giugno¹, Alfredo Pulvirenti¹, and Dennis Shasha²

¹ Dipartimento di Matematica ed Informatica, Università di Catania, Catania, Italy
vincenzo.bonnici@hotmail.it ferro@dmi.unict.it giugno@dmi.unict.it
apulvirenti@dmi.unict.it

² Courant Institute of Mathematical Sciences, New York University, New York, USA
shasha@cs.nyu.edu

Abstract. Approximate graph searching plays a key role from biology to computer graphics. Here we propose RelaxGrep, a tool that guides the user to find subgraphs in the database that are isomorphic to a “relaxation” of an original query Q , (i.e., a graph obtained after removing edges or nodes from Q). This is done through a modification of the filtering and verification phase of GraphGrep. Experiments yield promising results on molecular databases.

Keywords: Approximate Subgraph Matching, Graph Database, Database Indexing

1 Introduction and Related Work

Graphs can be used to represent complex and structured data. Usually, the graphs structure is shared among the data. For this reason, graph searching finds applications ranging from biology to chemistry and computer graphics.

Given a database of graphs $D = \{G_1, G_2, \dots, G_n\}$ (e.g. collection of molecules, etc.) and a query graph Q (e.g pattern), a graph searching tool finds all graphs in D containing Q as a subgraph. All occurrences of Q in those graphs are detected. A more general problem is to find the occurrences in $G_i \in D$ which are “similar to” Q . Although there are many possible definitions of similarity, one the most natural is the following: *Find subgraphs in D that are isomorphic to a relaxation of query Q , that is a new graph Q' obtained after removing edges or nodes from the original query Q .* The aim is to minimize the number deletions to obtain significant matches. Relaxed subgraph search problem finds a large number of applications. For example, chemists often seek chemical structures with similar structural characteristics (see Fig. 1). Since subgraph matching is an NP-Complete problem, we make use of heuristics.

The steps of an exact subgraph searching tool are usually the following: (i) global index construction (once offline); (ii) the filtering phase (at query time); (iii) verification phase (by using subgraph matching algorithms). In the index construction step, all graphs in the database are examined and decomposed into

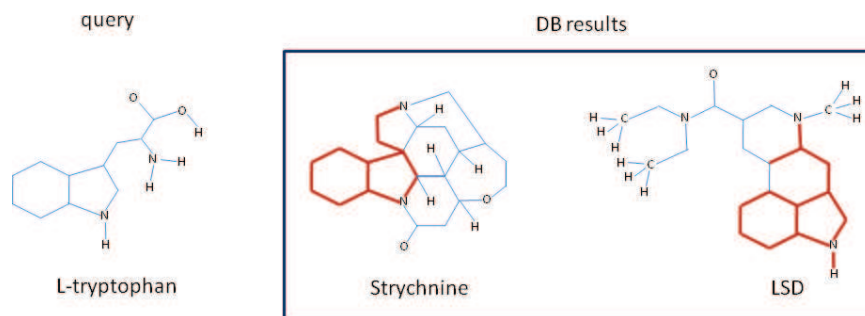


Fig. 1. An example of relaxed matching on molecular compounds. The compounds are represented as graphs where vertices are atoms labeled with their element symbol (unlabeled vertices corresponds to C atoms), and edges are bonds (double bounds are represented as single edges). The red-colored part of Strychnine and LSD matches a part of the Tryptophan structure. Finding this match allows a user to identify compounds which share chemical properties.

simpler structures, called features. A global index is built to access those features. In the filtering step, features are extracted from the query and searched into the global index. Database graphs containing all query features are retained as candidates; the others are discarded. In the verification step, a full graph search verifies if the retained graphs contain the query. Most existing systems are classified by their kinds of features. GraphGrep [4] uses paths as features, gIndex [6] and Fgindex [1] use frequents subgraphs, and GCoding [8] uses graph signatures made by concatenating vertex signatures.

Approximate searching tools use features also. For example, GraphGrep [4] supports similarity search by allowing queries having wildcards which represent non specified nodes or paths. Indexing is used to search for the wildcard-free query producing a set of candidate graphs. The search for the approximate part of the query is performed by a full search in the candidate graphs.

Grafil [7] models each query graph as a set of features. It introduces two data structures: a feature-graph matrix to compute the difference in the number of features between a query graph and graphs in the database and an edge-feature matrix to compute a bound on the maximum allowed feature misses based on a query relaxation ratio. In [2], authors introduced SIGMA. The system is based on associating a feature set with each edge of the query and looking which of those can be removed to obtain an exact match. The problem is then modeled through the multi-set multi-cover problem. SIGMA outperforms Grafil showing a better filtering power.

SAGA [5] allows node gaps, node mismatches, and graph structural differences. It uses a similarity distance containing three components. The *Struct-Dist* component measures the structural differences of the match, the *NodeMismatches* component represents the penalty associated with matching two nodes with different labels, and the *NodeGaps* component is used to measure the

penalty for the gap nodes. An index is built on small substructures of the graphs in the database. This index is then used to match fragments of the query with fragments in the database. Finally, the matching fragments are assembled into larger matches.

In this paper, we propose an extension of GraphGrep [4] to perform approximate searches. Given a query, it first searches for exact matches, then it advise the user to relax the query by suggesting the deletion of nodes or edges in order to find a larger number of occurrences. This is done by using the information stored in the global index. RelaxGrep was tested and compared with the state of art SIGMA [2] using the molecular datasets Antiviral Screen Dataset [3] which contains the topological structures of 42,000 chemical compounds that have been tested for evidence of anti-HIV activity. Experiments show that RelaxGrep is a simple solution allowing a fast running time. It represents an alternative solutions for large datasets, in particular for multi-edge removing.

2 RelaxGrep

RelaxGrep is a novel approach for approximate graph search. It uses indexing and matching algorithms of GraphGrep [4]. The filtering and the verification phases are properly modified to generate relaxed queries from the original one.

Given a graph, it assigns to each node a unique identification number and a label. In the index construction phase, graphs of the database are examined to extract their features in the following way. Visits each node of a graph by a depth-first traversal limited to a depth equals to a fixed number lp . For each visited path, store in the index the *labels-path* and the *ids-path*. The labels-path is the sequence of the corresponding labels of the nodes of the path, whereas the ids-path is the sequence of the corresponding nodes ids. The number of the occurrences of each path into each graph is also stored.

In the filtering phase the set of features of the query graph are extracted and compared with the index of the database. If a path present in the query has a number of occurrences greater than the number of times it appears on a database graph G_i , RelaxGrep will increase the *number of negative matches* corresponding to the graph G_i . Moreover, the label and ids of that path will be stored. This information will be used to advise the user which edges or nodes have to be deleted in the query.

The graphs of the database are divided into three sets. The first stores the graphs with positive matches (i.e. graphs that may contain the original query); the second maintains the graphs that have a number of negative tests less than a fixed threshold; the third contains all remaining graphs. The first two sets are considered to be candidates graphs that contain exact or relaxed occurrences of the query.

RelaxGrep uses an *edge-removal approach* to relax the query graph. The features are stored as sets of pairs of nodes representing the edges of the paths. Given a path (v_1, v_2, \dots, v_n) all pairs $\{(v_i, v_j) \text{ for } i = 1, i \leq n - 1, j = i + 1\}$ are stored into a hash-table indexed by the nodes ids. For each negative occurrence found during

the filtering step of a query path (v_1, v_2, \dots, v_n) , all the counters associated to the edges in the path are increased. RelaxGrep erases from the query the edge that has the minimum count and runs the verification phase. The process can be repeated until no edges can be deleted or the found occurrences are satisfactory. The algorithm for one edge removing can be formally modeled as follows. Let $F(Q)$ be the list of features extracted from the query Q . Since two features may have the same label path but having two different id-paths, $F(Q)$ contains repeated label paths. The list of features $F(Q)$ can be grouped into overlapping subsets with respect edges. For each edge e of the query graph let $F_e(Q)$ be the subset of features of $F(Q)$ containing the edge e . Let $F(G_i)$ the features of the i -th graph of the database. We can obtain the set of features of Q not contained into G_i as follows: $\bar{F}(Q) = F(Q) \setminus F(G_i)$. On the other hand, we can define the features contained in both Q and G_i as: $F'(Q) = F(Q) \cap F(G_i)$. Then, the edge, e^* , with highest probability to be deleted is the one that satisfies the following equation (there could be more than one):

$$e^* = \operatorname{argmax}_e |\bar{F}(Q) \cap F_e(Q)|.$$

More in general, we can conclude that e^* satisfies also: $F_{e^*}(Q) \cap F'(Q) = \emptyset$ or $\operatorname{argmin}_e |F'(Q) \cap F_e(Q)|$. By applying such a strategy we can progressively remove edges from the query Q until the relaxed query does not pass the filtering phase or the number of admitted deleted edges is reached.

Although this greedy strategy is heuristic and it does not guarantee finding the relaxed query with larger number of occurrences, it gives promising experimental results.

We developed a graphical interface for the tool in Visual C++ by using the library Graphviz (Ellson, 2004) for graph visualization. The user can execute all the phases of RelaxGrep from the preprocessing step to the matching statistics. It shows the execution times, number of matching graphs, the matching substructures and the graphs in the database. The user can select the threshold used for the filtering and visualize the query before and after the edges deletion. Moreover, user can select which edge to delete, if the one suggested by RelaxGrep visually is not satisfactory.

3 Experimental results

RelaxGrep was implemented in C++ and compiled with the GNU compiler 3.3. In the experimental analysis we measure the preprocessing and filtering time, the number of candidates graphs and the number of matches obtained by query relaxation. The system has been tested using the Antiviral Screen Dataset [3]. The AIDS database contains the topological structures of 42,000 chemical compounds that have been tested for evidence of anti-HIV activity. Queries were randomly extracted from the AIDS database. This process generates groups of 100 queries from each database having a number of edges equal to 4, 8, 16, and 32. The filtering threshold has been set to 50%.

Table 1 shows the filtering time, the number of candidates and the number of

matches before and after the deletion of an edge selected by RelaxGrep and SIGMA [2]. The filtering time refers to the total time up to the generation of candidate graphs, including the time to relax the query.

In Table 2, we report a comparison of RelaxGrep and SIGMA with one and two edges deletion using a dataset of 1000 graphs. Queries have 16 and 32 edges respectively. SIGMA generates candidate graphs for query with a n edge deletions. It does not suggest which of the n edges have to be deleted. The number of matches reported are generated by deleting all possible combinations of n (in our experiments all possible pairwise edges combinations) edges from the query and the verification is performed only on the candidates graphs. On contrary RelaxGrep is a fast heuristics method able to detect the n edges in the query that have been less used in the features that match in the graphs (i.e. less frequent or absent in the features of the graphs). Therefore, RelaxGrep detects and removes the query edges producing negative matches with high probability and not all the possible combination of edges.

Table 1. Comparisons between the approximate systems SIGMA and RelaxGrep (sec).

Query dim.	Filtering SIGMA	Filtering RelaxGrep	Candidates SIGMA	Candidates RelaxGrep	Matching SIGMA	Matching RelaxGrep
4	3.84	0.355	32362	39198	31857	31724
8	10.33	0.822	23842	29739	17945	18263
16	15.66	1.115	8247	20534	1208	481
32	10.88	1.275	588	12273	15	5

Table 2. Comparisons between the approximate systems SIGMA and RelaxGrep (sec). using a dataset of 1000 graphs

Query dim. (Deletions)	Filtering RelaxGrep	Filtering SIGMA	Candidates RelaxGrep	Candidates SIGMA	Matching RelaxGrep	Matching SIGMA
16(1)	0.036	0.266	114	189	5.28	16.76
16(2)	0.031	0.361	174.85	411.21	13.91	92.32
32(1)	0.072	0.19	17.24	28.48	1.19	1.55
32(2)	0.066	0.303	24.31	54.13	1.26	2.82

4 Conclusion

RelaxGrep gives a new fast solution to approximate graph searching, guiding user through deletion of edges to relax the query in order to increase the number of occurrences. This allows user to obtain more interesting results than the exact one. Moreover, RelaxGrep is equipped with a friendly interface to visualize queries, database graphs, relaxed queries and matches. Users may also choose graphically, under the suggestion of RelaxGrep, which edges to delete. RelaxGrep represents an alternative compared to existing methods for large datasets. Future work includes the biological soundness verification of the compared results.

Acknowledgements

We thanks Raffaele Di Natale, Giovanni Di Pietro and Claudia Puzzo for their useful work on this project during the Di Pietro and Puzzo's computer science Bachelor Thesis 2008.

References

1. J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. *Proceedings of ACM SIGMOD international conference on Management of data*, pages 857–872, 2007.
2. M. Mongiovi, R. Di Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. Sigma: A set-cover-based inexact graph matching algorithm. *Journal of Bioinformatics and Computational Biology*, 8(2):199–218, 2010.
3. National cancer institute. u.s. national institute of health. <http://www.cancer.gov/>.
4. D. Shasha, J.T-L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. *Proceeding of the ACM Symposium on Principles of Database Systems (PODS)*, pages 39–52, 2002.
5. Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel. Saga: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(2):232–239, 2007.
6. X. Yan, P. S. Yu, and J. Han. Graph indexing based on discriminative frequent structure analysis. *ACM Transactions on Database Systems*, 30(4):960–993, 2005.
7. X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. *Proceedings of ACM SIGMOD international conference on Management of data*, pages 766–777, 2005.
8. L. Zou, L. Chen, J.X. Yu, and Y. Lu. A novel spectral coding in a large graph database. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 181–192. ACM, 2008.