Cost-Based Optimization of Regular Path Queries on Large Graphs

André Koschmieder
Humboldt-Universität zu Berlin
koschmie@informatik.hu-berlin.de

ABSTRACT

The significance of regular path queries (RPQs) on graph-like data structures has grown steadily over the past decade. Prominent application areas are XML/XPath, RDF/SPARQL, analysis of social networks, and queries on biomedical networks. However, current implementations of RPQ are restricted either in the type of the graph (e.g., only trees), the type of regular expressions (e.g., only single steps), and/or the size of the graphs. No research has yet tried to evaluate general RPQs on large graphs, i.e., with millions of nodes/edges.

The predominant current techniques for dealing with RPQ use automata. However, we show that this approach, developed for tree-structured XML, does not work well in general graphs. We developed a novel approach for answering RPQs using ideas from cost-based query optimization. Essentially, our method exploits the fact that not all labels in a graph are equally frequent. We devise an algorithm which decomposes an RPQ into a series of smaller queries by concentrating on rare labels, i.e., those elements of the query which have fewer matches in the graph. Comparison of this rather simple method to automata-based techniques across a wide range of queries and graphs shows that the automata-based approach is not able to handle large graphs due to the enormous amount of memory that is required, and that the cost-based method outperforms the automata-based approach in all cases.

1. INTRODUCTION

A general regular path query (RPQ) is a regular expression R evaluated on a directed, labeled graph. Its result is the set of all paths in G whose concatenation of labels (edge or node) spells out R. Different flavors of RPQs are used in a wide range of applications. For instance, XPath supports a restricted form of RPQs on XML documents [6]. For RDF data, SPARQL supports a simple form of RPQs, and various proposals exist for enhancing SPARQL with full RPQs (e.g., [2, 7]). In social communities, relations between people, events or facts can be queried using RPQ-like predicates [13]. Another important application domain are the Life Sciences, where understanding the interactions of different biological entities is of great importance [1]. Such interactions are often modeled as graphs and RPQs are used to find specific biochemical pathways between distant nodes [8].

To illustrate the problem and our main idea for solving it efficiently, we give some concrete examples. Suppose a graph of researchers (nodes), either labeled as **P**rofessors or **ST**udents, connected by edges such as **S**upervised or **J**oint work. In this graph, the query $P(JP)(JP)?$ finds all paths between a professor and direct or indirect co-workers. $(PS)(PS)+(P|T)$ finds all paths between a professor and his doctorate descendants. Now suppose we also model research prizes as nodes (such as **N**obel Prize or Sigmod **A**ward), and connect them to researchers with edges labeled **H**onored. Then, we can find the doctorate predecessors of all Nobel Prize winners using the query $(PS)+PHN$. Although answering this query in principle requires to search the entire graph, it is immediately clear that it is sensible to start the search at nodes labeled with N, because (a) one such node must be in any matching path and (b) there are much less Nobel Laureates than professors. Such reasoning is the basis of the approach we propose in this paper.

The usual approach to answer RPQs is using automata [4]. Both graph and query are represented as automata, whose intersection automata is the subgraph specified by the query. In this process, the graph needs to be translated into a DFA, which can be of exponential space and time. Therefore, this only works well for restricted forms of graphs, such as XML, while space consumption is enormous on general graphs.

In this work, we present an alternative technique, which, in principle, directly searches the query in the graph. As searching the whole graph without dedicated start or end points is not feasible, we developed a technique that is built on ideas from cost-based query optimization. We gather labels in the query that only occur a few times in the graph and use these as fix points for a bi-directional search. We split the query at these rare labels into smaller queries, and answer these individually. For each of these queries, occurrences of the rare labels are start and/or end points, which makes graph searching much faster. We search all paths between each two adjacent rare labels as well as at the start and end of the query, and combine the results to answer the original query. The main advantage of this approach is that often we do not need to consider the whole graph, but only those fractions of it that lie between adjacent rare labels.

Figure 1 gives an example of our idea. Suppose we want to answer the RPQ $a+ \ b \ c+$ on a graph (Fig. 1b). Since there is only one edge labeled with b, we use it as rare label and split the query there. Now, the two smaller queries $a+$ and $b+$ have to be answered, using the b edge as end point or start point, respectively. The result of the original query is the combination of the smaller queries and the rare label.

Interestingly, we observed that this method outperforms

GvD Workshop '10, 25.-28.05.2010, Bad Helmstedt, Germany.

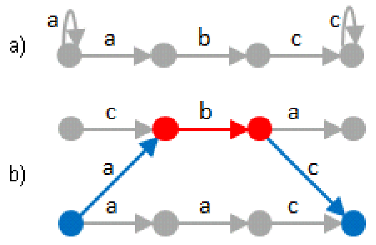

Figure 1: a) The RPQ $a+\ b\ c+$ as an automaton, b) a path fulfilling the RPQ in a small graph.

the automata-based approach even if graphs (or queries) do not contain any rare labels, though its advantages are the more pronounced the less homogeneous the label distribution is. This particularly is the case in the Life Sciences, our main target area, where graphs usually have a Zipfian label distribution (see Section 3). Thus, there are few labels that occur very frequently, and many labels with very few occurrences, which is an ideal case for our optimization.

Our idea can also be used for variations of the general RPQ problem, such as finding the shortest path spelling out the regular expression, or finding all matching paths between two given nodes. In this paper, we focus on finding all matching paths in the graph, which means finding the set of subgraphs specified by the RPQ.

This paper is structured as follows. Section 2 gives an overview on related work. In Section 3, we give definitions of basic concepts. We introduce our cost-based optimization for directly searching a graph in Section 4. We evaluate our method in Section 5 and conclude in Section 6.

2. RELATED WORK

As Mendelzon and Wood show, searching acyclic paths in a graph that match a regular expression is NP-hard [10]. However, they also show that the problem can be solved in polynomial time if specific restrictions apply to the graph or to the regular expression. A few approaches have been proposed to speed up these kinds of queries.

The common approach for answering RPQs is based on automata joining. An implementation using so-called "Representative Objects"are the DataGuides by Goldman and Widom [4]. They use the minimized DFA of the graph as an index. However, this index can become much larger than the original graph, which is a problem for large graphs. Goldman and Widom propose "Approximate DataGuides" [5] which reduce the index size using heuristics, but are still unfeasible for large graphs (see Section 5).

A lot of work has been done for answering path queries on semi-structured data, mainly XML, where XPath has been established as path query language [6]. However, XML data have a tree structure, so these ideas do not necessarily work on graphs in general. Additionally, XPath does not support general RPQs. In contrast, our approach aims at answering general RPQs on arbitrary graphs. Another area thoroughly researched in recent years is querying RDF data. SPARQL is the official W3C recommendation as an RDF query language, but it does not support regular path queries. Several extensions to SPARQL have been proposed to allow RPQs, e.g. [2, 7]. However, there are no new execution plans behind these proposed query languages.

In the context of biomedical research, additional query languages have been proposed more recently. Mork et al. [11] propose a query language for semi-structured biological data. Leser [8] proposes a pathway query language for querying protein interactions or pathway databases. Both languages support a certain form of RPQs, but neither of them presents an efficient resolution technique.

There is little work that is directly related to our aims. Sevon and Eronen [14] describe a method for querying paths in labeled graphs using context-free grammars. They traverse the graph in a breadth-first way and use a context-free parser to find matches. While context-free grammars are more powerful than regular expressions, [14] only focus on finding paths between fix start and end nodes. Fernandez and Suciu present another interesting approach using Graph Schemas [3], which, however, have to be created manually, a step that seems unfeasible for graphs with millions of nodes.

3. TERMS AND DEFINITIONS

We use labeled directed multigraphs, i.e., a graph G is a tuple $G = (V, E, f, l, \Sigma)$, where V is a finite set of nodes, E is a finite set of edges, $l : E \to \Sigma$ specifies the edge labels Σ, and $f : E \to V \times V$ is the connection function, specifying which nodes are connected by which edges.

The topological properties of a graph can be measured through node degree and label distribution. A graph is called scale-free if the node degree distribution asymptotically follows a power law. That means, the number of nodes with degree k is $P(k) \sim k^{-\lambda}$ for large values of k, where λ is a constant. Scale-free graphs are the likely outcome of various random growth processes, and, indeed, many graphs discovered in biological research are scale-free [9]. Similarly, the label distribution in a graph is called Zipfian [15] if the frequency of the labels occurring in the graph follows the power law $F(k) \sim k^{-\delta}$, with $\delta \approx 1$.

An RPQ is a regular expression over Σ. We use the common definition for REs: (1) If $a \in \Sigma$, then a is a regular expression specifying $L(a) = \{a\}$. If $a, b \in \Sigma$, then (2) (ab) is a regular expression specifying $L(a)L(b)$, and (3) $(a|b)$ is a regular expression specifying $L(a) \cup L(b)$. (4) If $a \in \Sigma$, then $(a)*$ is a regular expression specifying $L(a)*$. Unnecessary parentheses may be omitted, and the syntactic sugar operators $+$ and ? may be used as usual. We also define a special symbol "." specifying any single symbol of Σ: $L(.) = \Sigma$.

4. COST-BASED OPTIMIZATION

In this section, we present a novel strategy for answering RPQs: searching the graph instead of matching automata. The basic idea is simultaneously to search the graph while advancing the query automaton. The main advantages of this method are that it is simple, no preprocessing is needed, we only use space linear in the size of the original graph, the implementation is easily parallelizable, and cost-based optimization methods are applicable.

We are not aware of any reported previous work on directly searching a graph for RPQs. This is probably due to the fact that the worst-case complexity is bad, and that, indeed, searching the graph without fix start or end points is not feasible (shown as baseline in Section 5). However, we introduce a technique to work around this problem based on the concept of rare labels. We use labels from the query that occur a few times at maximum in the graph, and split the query at these labels. We then answer the subqueries using the rare labels as fix start or end points. In this section, we explain this concept in detail.

4.1 Rare Labels

Definition 4.1. *Let G be a graph and R be a RPQ. We call a label occurrence in R* mandatory *iff it occurs in every possible result of R in G. We call a label* rare *iff it occurs at most m times in G and is mandatory in R.*

For example, in the regular expression $ab + c * d?$, a and b are mandatory, while c and d are not. Note that, according to our definition, rare labels are specific for a query. However, retrieving them is fast if a list of all labels and their occurrences in the graph is stored. If this list is indexed by labels, finding all rare labels for a given query is linear to the size of the query pattern.

If a query contains a rare label, then any match of the query in the graph must contain an occurrence of it. Therefore, we can use the occurrences of rare labels as way-points during search. If we can find two or more rare labels in a graph, we can use a two-way search algorithm to find all matching paths between them, and every additional known node on the path reduces the search space further.

This idea can be visualized intuitively assuming a graph of randomly distributed nodes in a 2D space where every node is connected to its k nearest neighbors. In such a graph, the number of nodes that are visited in a breadth-first search correlates with Euclidean distance, i.e., with the size of a circle around a node. In Figure 2a), we assume that a query contained two rare labels. Thus, we perform a two-way search between the two nodes in a breadth-first manner, during which we visit a number of nodes that correlates with the size of the circles in the figure (for the full answer, we also need to search from the rare labels to possible start and end points; furthermore, rare labels usually are not unique). In Figure 2b), we assume a third rare label, which reduces the number of visited nodes by a factor of 2.67. In such graphs, the search area for n nodes is $S = \frac{n \cdot \pi}{(n-1)^2} \cdot \frac{d^2}{4}$ and thus shrinks linearly to the number of known nodes between the start and end nodes. Note that this example is only given as illustration and that the formulas are not valid for arbitrary graphs. However, the number of visited nodes always correlates with the distance (in hops) to a known node, and thus the general idea also holds for arbitrary graphs.

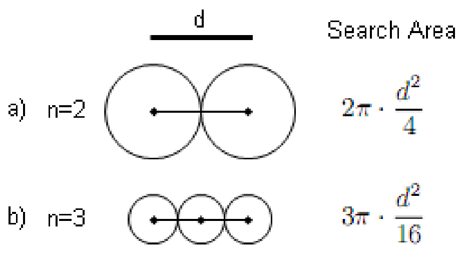

Figure 2: Illustration of the area that needs to be searched in a two-way search with different numbers n of known way-points (only valid for certain type of graphs, see text).

4.2 Searching the Graph using Rare Labels

For queries that include at least one rare label, we split the query at these rare labels and use them as fix points in the search. Searching the graph and advancing in the regular expression at the same time, we search all paths between each two adjacent rare labels, all paths from the first rare label backward to the start of the regular expression, and from the last rare label forward to the end. As shown above, the number of nodes that needs to be visited during this search shrinks with every additional rare label but grows with increasing numbers of occurrences of rare labels.

Besides keeping the search space smaller, rare labels often also allow for early stops. If there is no path between any two adjacent rare labels, then there can be no path fulfilling the original query, and the search can be stopped immediately.

In the following, we use the term *first rare nodes* for all nodes that are starting point of an edge of which the label is the first rare label, according to the regular expression. Analogously, *last rare nodes* are the end nodes of all edges with the last rare label. Answering RPQs using rare labels is done in the following 6 steps.

1. Gather all rare labels for the query in the graph.
2. If more than one rare label exists, find the paths between the first and second rare label, the second and third etc. using a two-way search algorithm. If no path can be found in any of these search processes, stop the search and return an empty result for the query.
3. If more than one rare label exists: Using the results from step 2, find all paths from the *first rare nodes* to the *last rare nodes* and remove all rare nodes that are on no path, as these cannot be on a result path.
4. Beginning at all remaining *first rare nodes*, find all paths to the beginning of the regular expression, searching backward.
5. Beginning at all remaining *last rare nodes*, find all paths to the end of the regular expression (forward).
6. Using the results, enumerate all paths in the graph that fulfill the regular expression and return the result.

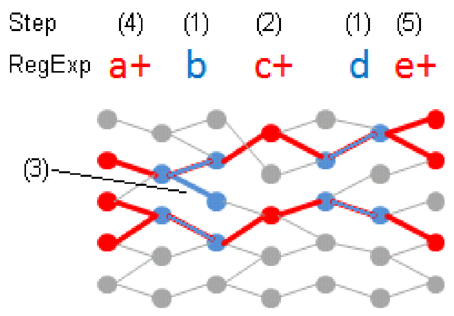

Figure 3: Search process example for the query $a+\ b\ c+\ d\ e+$ (edge labels and directions omitted).

Figure 3 shows the principle of the algorithm. On a sample graph (edge labels and directions are omitted), the query $a+\ b\ c+\ d\ e+$ is executed, assuming that b and d are rare labels. In step 1, rare label edges are gathered (b and d edges). In step 2, we search all paths between the end nodes of the b edges and the beginning of the d edges. These paths must fulfill the regular expression between the two rare labels, in this case $c+$. In our example, two such paths can be found. For one rare edge, no path could be found, thus it is removed from further consideration in step 3.

In step 4, a one-way backward search is performed, starting at the start nodes of the b edges. The search ends once all paths have been found that fulfill the first part of the regular expression $(a+)$. In step 5, we search all paths from the end of the last rare label to the end of the regular expression in forward direction. As a last step (not shown in the picture), we enumerate all paths, combining the results of the previous steps. In this case, there are 4 distinct paths.

Our approach specifically aims at queries that include labels that do not occur often in the graph. While most queries used in bioinformatics are interested in these rare labels, there are also queries in which no rare label is present. These queries cannot be accelerated using the algorithm presented above. Our implementation uses a brute force method to resolve these kinds of queries: start a simple search at every node in the graph. Although this approach is also highly parallelizable, queries with rare labels can be answered much faster.

4.3 Cost-based Parameters

The choice of m, the parameter determining which labels to consider rare, is a compromise between treating as many labels as possible as rare and keeping the number of occurrences of rare labels small. If a rare label has many occurrences in the graph, the search space increases because each occurrence needs to be included in the search. On the other hand, multiple different rare labels in a query speed up its execution, as partial paths to be searched are shorter.

The best value for m depends on the graph as well as the query. Thus, using a fixed value is not the best approach. We therefore use a cost-based technique for determining which labels to consider rare. The idea is, if, for a given query, there are several possible rare labels, we set the threshold for rare labels higher than if only very few rare labels can be found. This produces less queries without any rare labels, while for queries with many potential rare labels, only labels with a small number of occurrences are included.

Our proposed heuristic works as follows. We acquire a list of all potential rare labels for a given query. We then reduce the list depending on the overall number of paths that would need to be searched in the current configuration. Labels that produce the most paths are removed first; we can compute the number of paths between any two adjacent rare labels r_1, r_2 as $|r_1| \cdot |r_2|$. The overall number of paths is the sum of all paths between all adjacent pairs. We repeatedly remove the rare label that produces the most paths, until the sum of all paths is below a threshold. We found a threshold of 100 to produce good results in most cases.

5. RESULTS

In this section, we compare our cost-based method with an automata-based implementation, and do further analysis concerning scalability and influence of different kinds of graphs and queries. All tests were executed on a Quad-Core AMD Opteron machine with 16 GB of main memory.

5.1 Graphs and Queries

To evaluate the performance of our method, we used graphs from biological research as well as artificially created graphs. As an example for real world graphs, we present results for *AliBaba* [12], which is a network of protein-protein-interactions extracted by text mining on all of PubMed. The graph has about 50,000 nodes and 340,000 edges. Results for several other biological networks we tested were similar.All graphs used for the results are roughly scale-free and roughly have a Zipfian distribution of edge label frequencies. The artificial graphs were created to specifically fulfill these properties.

Execution speed for the cost-based method depends on the query in question. We randomly created sets of 1000 different RPQs with varying properties, differing in query length, frequency of modifiers (+, *, ?), alternatives and bracketed parts, frequency of rare labels, and the maximum number of occurrences of rare labels. We omit results for different classes of queries due to space constraints and only report aggregated times. Essentially, the behavior of our method was as expected: Longer queries are slower, frequency of modifiers may slow down or speed up times, depending on the concrete modifiers, more rare labels increase performance, and more frequent rare labels decrease speed.

5.2 Comparison

To compare our cost-based method with an automata-based implementation, we used queries containing rare labels as well as completely random queries to show the difference. However, we could only use graphs of relatively small size due to excessive runtime and memory requirements of the automata-based approach.

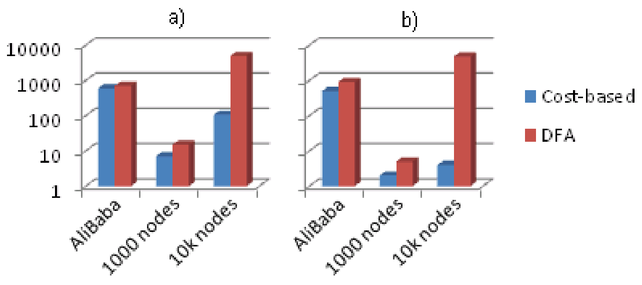

Figure 4: Runtime in seconds (logarithmic) to answer 1000 queries a) without and b) with rare labels.

Figure 4 shows the runtime for answering 1000 RPQs for the cost-based and automata-based implementation. For queries without rare labels (Figure 4a), the runtimes do not differ much for small graphs, but for larger graphs, the cost-based approach executes much faster. We found the same behavior for graphs with equal distribution of edge labels (note that not using rare labels in a query essentially has the same effect as running random queries on graphs with equally distributed label frequencies).

Figure 4b) compares the automata-based method and the cost-based method using queries that contain at least one rare label. It can be seen that the latter always outperforms the former, and that the difference grows with the graph size, reaching orders of magnitude for synthetic graphs with more than 10,000 nodes. The difference is less pronounced, but still considerable (about 20%) for the AliBaba graph, as there, the frequency of labels diverges more from the Zipf-distribution than in the synthetic graphs.

Comparing Fig. 4a) and b), one can see that the query type has a big influence on the runtime of our method.

Queries that contain rare labels can be answered much faster than those without. In contrast, we found that for the automata-based implementation, the runtime for different queries on the same graph is about equal. The reason is that preprocessing of the graph (converting it to a minimized DFA) takes most time, while answering queries is fast.

5.3 Scalability

To test the scalability of our rare label method, we used artificially created graphs with varying properties. The queries contain at least one rare labels. We evaluated the effect of our cost-based optimization by comparing it to a baseline method, which performs a brute-force search starting at every node without considering label frequencies.

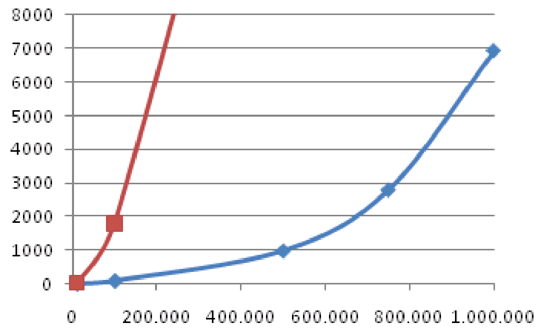

Figure 5: Runtime for answering 1000 queries on synthetic graphs (number of nodes shown). The steeper line shows the runtime for the baseline.

Figure 5 shows how our implementation scales with the size of the graph at a fixed node/edge ratio (1:2). The smallest graph has 10,000 nodes and 20,000 edges, and the largest graph has 1 million nodes and 2 million edges. The scaling of the cost-based approach is much better than the exponential scaling of the baseline. Even for the largest graphs we tested, we can answer a RPQ in few seconds on average.

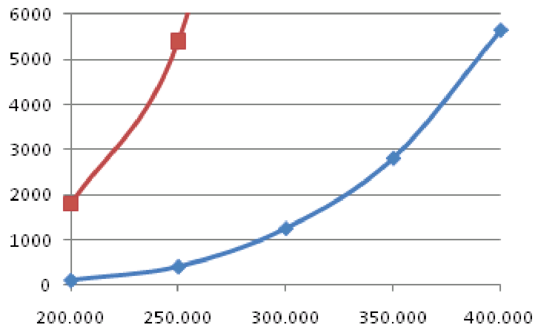

Figure 6: Runtime for answering 1000 queries on synthetic graphs with 100,000 nodes and different numbers of edges. The steep line shows the runtime for the baseline.

Figure 6 shows how our implementation scales with the average degree of the graph. Using multiple artificial graphs with 100,000 nodes, we increased the number of edges (and thus the average degree), leaving all other properties constant. As a baseline, the execution time for answering the queries without using rare labels is shown (steep line on the left). Again, the increase in execution time is favorable compared to the exponential growth of the number of possible paths in the graph.

6. CONCLUSION

In this paper, we presented a novel approach for answering regular path queries using cost-based optimization. We use labels that are infrequent in the graph and also appear in the query. We use these rare labels as start-, end-, and waypoints when searching the graph using the query converted into an automaton.

We showed that for a graphs with properties like many real-world graphs, queries containing rare labels can be answered much faster with the cost-based approach than with the traditional automata-based method. Furthermore, the former requires only linear time to preprocess the graph and is highly parallelizable. We also showed that the cost-based implementation scales well with the size of the graph and with its density.

7. REFERENCES

[1] L. Alberghina and H. V. Westerhoff, editors. *Systems Biology: Definitions and Perspectives*. Springer, 2005.

[2] K. Anyanwu, A. Maduko, and A. Sheth. Sparq2l: towards support for subgraph extraction queries in rdf databases. In *WWW '07*, pages 797–806. ACM, 2007.

[3] M. F. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *ICDE '98*, pages 14–23, Washington, DC, USA, 1998. IEEE.

[4] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB '97*, pages 436–445, 1997.

[5] R. Goldman and J. Widom. Approximate dataguides. In *Workshop on Query Processing*, 1999.

[6] T. Grust. Accelerating xpath location steps. In *SIGMOD 2002*, pages 109–120, New York, NY, USA.

[7] K. J. Kochut and M. Janik. Sparqler: Extended sparql for semantic association discovery. In *ESWC '07*, pages 145–159, Berlin, Heidelberg, 2007.

[8] U. Leser. A query language for biological networks. *Bioinformatics*, 21(2):33–39, 2005.

[9] L. Li, D. Alderson, and R. T. et al. Towards a theory of scale-free graphs: Definition, properties, and implications. *Internet Mathematics*, 2(4), Mar. 2006.

[10] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

[11] P. Mork, R. Shaker, A. Halevy, and P. Tarczy. Pql: a declarative query language over dynamic biological schemata. In *American Medical*, pages 533–537, 2002.

[12] C. Plake, T. Schiemann, M. Pankalla, J. Hakenberg, and U. Leser. Alibaba: Pubmed as a graph. *Bioinformatics*, 22(19):2444–2445, October 2006.

[13] M. San Martín and C. Gutierrez. Representing, querying and transforming social networks with RDF/SPARQL. In *ESWC 2009*, pages 293–307.

[14] P. Sevon and L. Eronen. Subgraph queries by context-free grammars. *Journal of Integrative Bioinformatics*, 5(2):100, 2008.

[15] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Hafner, New York, 1949.