Within network learning on big graphs using secondary memory-based random walk kernels

Jianyi Lin, Marco Mesiti, Matteo Re and Giorgio Valentini

Abstract Significant advances in high-throughput sequencing technologies raised exponentially the rate of acquisition of novel biological knowledge in the last decade, thus resulting in consistent difficulties in the analysis of vast amount of biological data. This adverse scenario is exacerbated by serious scalability limitations affecting state-of-the art within-network learning methods and by the limited availability of primary memory in off-the-shelf desktop computers. In this contribution we present the application of a novel graph kernel, transductive and secondary memory-based network learning algorithm able to effectively tackle the aforementioned limitations. The proposed algorithm is then evaluated on a large (more than $200,000$ vertices) biological network using ordinary off-the-shelf computers. To our knowledge this is the first time a graph kernel learning method is applied to a so large biological network.

Jianyi Lin
Khalifa University, Department of Applied Mathematics and Sciences, Abu Dhabi, United Arab Emirates, e-mail: jianyi.lin@kustar.ac.ae

Marco Mesiti
Università degli Studi di Milano, Dip. di Informatica, Via Comelico 39/41 - 20135 Milano (MI), e-mail: marco.mesiti@unimi.it

Matteo Re
Università degli Studi di Milano, Dip. di Informatica, Via Comelico 39/41 - 20135 Milano (MI), e-mail: matteo.re@unimi.it

Giorgio Valentini
Università degli Studi di Milano, Dip. di Informatica, Via Comelico 39/41 - 20135 Milano (MI), e-mail: giorgio.valentini@unimi.it

1 Introduction

Many efforts have been devoted in the last decade to developing automated tools for large scale automated function prediction of proteins (AFP) [2, 3]. A recent international challenge for the critical assessment of automated function prediction [6], highlighted that scalability and heterogeneity of the available data represent two of the main challenges posed by AFP. From a learning perspective the problem is further complicated by the different functional annotation coverage in different organisms that make very difficult the effective transfer of the available functional knowledge from one organism to another. A possible approach for gene functional annotation transfer between species relies on the availability of a collection of orthology relationships across interspecies proteins, and on the usage of an evolutionary relationships network as a suitable medium for transferring functional annotations to the proteins of poorly annotated organisms [7]. In this scenario a possible solution could be the application of network based learning methods on multi-species biological networks so that annotations coming from well annotated organisms could be used to effectively transfer functional annotations between species.

Unfortunately this approach is only apparently simple given the serious scalability limitations affecting graph-based learning algorithms (i.e. the popular label propagation and random walks based methods). These approaches usually rely on an in-memory adjacency matrix representation of the graph network, scale poorly with the size of the graph [9], and time complexity may become quickly prohibitive. When the size and structural complexity of the graph becomes so high that it is not possible to maintain it entirely in primary memory, alternative strategies (i.e. parallel/distributed computation [4, 11, 12], or secondary memory-based computation [8, 19, 5]) can be considered. However, at least in the case of the parallel/distributed computation, the identification of the optimal partitioning of the graph that minimizes the message passing requirements across a possibly large number of nodes of an HPC cluster is not immediate, especially in the case of very large and complex networks.

We previously proposed [13] a scalable semi-supervised network-based learning of protein functions algorithm that can be applied to large multi-species networks and is implemented using secondary memory-based technologies. Despite the appealing scalability performances from a learning standpoint this method is a classical random walk on graph. This paper extends the previous proposal by developing a local within network learning method based on the Random walk kernel [17] and a previously developed kernelized functions learning framework [15]. The novel local and secondary memory-based graph kernel method is compared with the classical random walks on graphs both in terms of learning performances and empirical time complexity.

To our knowledge this is the first reported case of application of a local and secondary-memory based graph kernel method to a very large biological network.

This manuscript is organized as follows. In the next section we introduce our proposed approach based on the local and secondary memory-based implementation of network-based algorithms (classical random walks on graph) for the multi-species

AFP problem. We then present the novel random walks kernel local algorithm. We finally compare the algorithms in a multi-species AFP problem on a large biological network including 13 species of Eukaryotes and containing more than $200,000$ proteins.

2 Local version of the classical Random Walk algorithm

Network-based algorithms learn by exploiting the overall topology of the networks [14, 15, 18], and their implementations usually require to process in primary memory a large part or the overall underlying graph. The main drawback of these implementations is that big networks cannot be entirely loaded into primary memory using off-the-shelf machines.

In [13] we developed local implementations of global network algorithms (classical random walks (RW) on graphs) by iteratively processing only one vertex and its incident edges at a time. In other words we do not reject to think globally by exploiting the overall topology of the network, but at the same time we solve locally by designing implementations of these algorithms through a vertex-centric programming model [4, 12].

A key feature of all the presented implementations is that the potentially very large matrices used by the primary-memory based versions of the classical random walk algorithm as well as of their kernelized versions are never computed nor stored entirely in main memory. All the learning algorithms presented in this paper were implemented by considering that:

- the existence of an edge in the network can be exploited as the only medium to propagate information between the vertices located at its endpoints;
- the knowledge of the set of edges originating from a vertex is enough to define its direct neighbourhood;
- the computation of the final score of a vertex can be decomposed in many steps each depending uniquely on the topology of the network and on the status of the vertices directly connected to the considered vertex;
- the score can be progressively accumulated into a single variable that is local to the vertices of the network.

RW algorithms [10] explore and exploit the topology of the functional network, starting and walking around from a subset $V_M \subset V$ of nodes belonging to a specific class M by using a transition probability matrix $Q = D^{-1}W$, where D is a diagonal matrix with diagonal elements $d_{ii} = \sum_j w_{ij}$. The elements q_{ij} of Q represent the probability of a random step from i to j. The initial probability of belonging to M can be set to $p^o = 1/|V_M|$ for the nodes $i \in V_M$ and to $p^o = 0$ for the nodes $i \in V \setminus V_M$. If p^t represents the probability vector of finding a "random walker" at step t in the nodes $i \in V$ (that is, p_i^t represents the probability for a random walk of reaching node i at step t), then the probability at step $t+1$ is:

$$p^{t+1} = Q^T p^t \tag{1}$$

and the update (1) is iterated until convergence. Given that a too deep exploration of the network can lead to a steady state with suboptimal learning performance, it is common practice to try with different number of predefined steps.

With the *RW* method at the steady state or at an optimized number of steps we can rank the vector p to prioritize nodes according to their likelihood to belong to the class M under study.

Looking from a "local" perspective at *RW* algorithm, the update rule (1) becomes:

$$p_i^{t+1} = Q_i \cdot p^t \tag{2}$$

where p_i is the probability of the i^{th} node, and Q_i represents the i^{th} column of the Q probability transition matrix. By recalling that W represents the original adjacency matrix of the graph and W_i its i^{th} column, from (2) we obtain:

$$p_i^{t+1} = D^{-1} W_i \cdot p^t = \sum_{j=1}^{n} d_{jj}^{-1} \, w_{ji} \, p_j^t \tag{3}$$

This is the update rule of the random walk resolved at the i^{th} node of the graph, and can be viewed as a "local" version of (1): by updating all the nodes i of the graph, $1 \le i \le n$, we update the probability vector p^{t+1} exactly in the same way of (1).

To compute (3) we need the following "local" data:

1. p_i^o (that is, the probability of the i^{th} node at start)
2. $d_{jj}^{-1} = \frac{1}{\sum_i w_{ji}}$ (that is, the sum of weights of the edges coming from j)
3. $w_{ji}, 1 \le j \le n$ (that is, the weights of the edges going to i)
4. $p_j^t, 1 \le j \le n$ (that is, the probabilities of nodes at the previous step).

If the graph is indirect (an this is the case for AFP problems), the weights of incoming and outcoming edges are the same, that is $\forall i, \forall j \;\; w_{ij} = w_{ji}$. This implies that we need to store only the list of edge weights outcoming from i: $L(i) = \{w_{ij} | w_{ij} > 0\}$. This in turn implies that in sparse graphs the spatial (and temporal) complexity at each node is sublinear. It is easy to see from (3) that the complexity of each iteration of the algorithm is $\mathscr{O}(n^2)$, but with sparse graphs, i.e. when $\forall i, |\{(j,i)|w_{ji} > 0\}| << n$, the complexity is $\mathscr{O}(n)$.

3 Local version of the Random walk kernel and Kernelized Score Functions

In [15] we proposed the kernelized score functions algorithmic framework that generalizes the notion of average, nearest neighbour and k-nearest neighbour distance from the set of positive nodes in a given network annotated to a specific functional class, and embeds a general kernel to model the functional similarity between nodes. This semi-supervised transductive learning method generalizes the guilt-by-association (GBA) approach [6] by introducing fast and efficient local learning strategies based on an extended notion of functional distance between the vertices, and adopts also a global learning strategy by using kernel functions able to exploit the relationships and the overall topology of the underlying network. The implementations presented in [15] were all "global" (primary memory-based). In order to significantly improve the scalability of the kernelized score functions we need to consider local implementations of:

1. The score function
2. The kernel embedded in the score function

The Average, Nearest Neighbour and k-nearest Neighbour score functions [16, 15] can be naturally implemented in "local" form once we are able to cast in local form the computation of the underlying kernel. In this work we focus on the Average score function:

$$S_{AV}(i,V_C) = \frac{1}{|V_C|} \sum_{j \in V_C} K(x_i,x_j) \quad (4)$$

where V_C is the set of positive vertices $i \in V$ that belongs to a given functional class and x_i, x_j are features associated respectively with node i and j, usually represented as real vectors. To compute (4) we need the following "local" data:

1. The i^{th} row K_i of the kernel matrix K
2. The set V_C (indices of the positive columns)

The complexity is $\mathscr{O}(|V_C|)$, that is constant if $|V_C| << n$.

The "global" version of the 1-step random walk kernel is the following [17]:

$$K_{rw} = (a-1)I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \quad (5)$$

Where I is the identity matrix, D is a diagonal matrix with elements $d_{ii} = \sum_j w_{ij}$ and W is the symmetric adjacency matrix of an indirect graph $G = (V,E)$. It is easy to derive from (5) the following "local" implementation of the 1-step random walk kernel, where $K(x_i,x_j)$, for the sake of simplicity is represented as k_{ij}:

$$k_{ij} = \begin{cases} d_{ii}^{-\frac{1}{2}} w_{ij} d_{jj}^{-\frac{1}{2}} & \text{if } i \neq j \\ (a-1) + d_{ii}^{-\frac{1}{2}} w_{ij} d_{jj}^{-\frac{1}{2}} & \text{if } i = j \end{cases} \quad (6)$$

To compute (6) we need the following "local" data for each edge (i,j):

1. its weight w_{ij}
2. the values $d_{ii}^{-\frac{1}{2}}$ and $d_{jj}^{-\frac{1}{2}}$

The local computation complexity is constant. To compute the overall matrix the complexity is $\mathscr{O}(n^2)$. The q-step random walk kernel with $q > 1$ can be computed by following a step-by-step strategy based on this recursive formula: $K_{rw}^{q} = K_{rw}^{q-1} K_{rw}$.

3.1 Putting together the Average score and the local version of the random walk kernel

By putting in (4) the local version of the random walk kernel (6), we obtain a "local" version of the *average score* with 1-step random walk kernel:

$$S_{AV}(i, V_C) = \frac{1}{|V_C|} \sum_{j \in V_C} K(x_i, x_j) = \frac{1}{|V_C|} \sum_{j \in V_C} k_{ij} =$$

$$= \frac{1}{|V_C|} \sum_{j \in V_C} \left(d_{ii}^{-\frac{1}{2}} w_{ij} d_{jj}^{-\frac{1}{2}} + \langle i = j \rangle (a-1) \right) \tag{7}$$

where $\langle z \rangle$ is 1 if z is true and 0 otherwise. To compute (7) we need the following "local" data:

1. The row W_i of the adjacency matrix W
2. The set V_C (indices of the positive columns)
3. the values $d_{ii}^{-\frac{1}{2}}$ and $\{d_{jj}^{-\frac{1}{2}} | j \in V_C\}$

The local complexity is $\mathscr{O}(|V_C|)$, that is constant if $|V_C| << n$.

The main problem affecting the proposed solution for the local computation of the Average score based on a random walk kernel is that, using an approach starting from the adjacency matrix, a certain locality is maintained in the computation of the 1-step and 2-steps but when we compute RWK with 3 or more steps we need the overall matrix and the "locality" is completely lost.

To overcome the complexity problems raising from the local computation of the p-step RWK with $p > 1$, we propose an iterative version of the kernelized score functions with random walk kernels.

3.2 Iterative computation of the kernelized Average Score function with p-step RWK

As stated in Section 3.1 the iterative nature of the p-step RWK computation poses serious challenges from a local implementation perspective and a progressive locality loss make solutions based on simple modification of the classical random walks unsuitable for real world big graph. In order to overcome this limitation we propose a novel representation of the combined RWK-kernelized score functions computation that better fits the constraints imposed by the analysis of very large graphs and by the vertex-centric programming paradigm.

More precisely, we propose an iterative formula for computing the Average kernel score with a p-step random walk for the whole graph. Such formula consists in a simple matrix-vector multiplication at each iteration.

Recall that for every node $i \in V$ the average score of the p-step random walk kernel starting from $V_C \subset V$ is $S_{AV}(i,V_C) = \frac{1}{|V_C|}\sum_{j\in V_C}(K^p)_{ij}$. Let's denote the column vector constructed from $S_{AV}(i,V_C)$ by varying i with

$$S_{AV}(V_C) = [S_{AV}(1,V_C), \cdots, S_{AV}(n,V_C)]^T.$$

It can be shown that the vector $S_{AV}(V_C)$ of average scores for the whole graph $G =< V,E >$ with a p-step random walk kernel starting from nodes of $V_C \subset V$ can be computed as $S_{AV}(V_C) = D^{\frac{1}{2}} v^p$ by the iterative formula

$$v^p = Mv^{p-1} \qquad \text{where } M = [(a-1)I + D^{-1}W]$$

with the initialization vector v^0 having element

$$v_i^0 = \begin{cases} \frac{1}{|V_C|\sqrt{d_{ii}}} & \text{if } i \in V_C; \\ 0 & \text{otherwise.} \end{cases}$$

We will now show how to compute the average score for the p-step RWK using a local implementation, i.e. on a vertex-based graph computation model.

Consider the "global" iterative formula $v^p = Mv^{p-1}$; denote by v_i^p the i-th element of v^p and by $I(i) = \{j \in V : w_{ij} > 0\}$ the incoming neighbors of i in the weighted graph G. We have

$$v_i^p = \sum_{j\in V} M_{ij} v_j^{p-1} = \sum_{j:M_{ij}>0} M_{ij} v_j^{p-1}.$$

Since

$$M_{ij} = \begin{cases} a-1 & \text{if } i = j \\ \frac{w_{ij}}{d_{ii}} & \text{otherwise.} \end{cases}$$

we can establish the rule for updating the value of a vertex i in the graph:

$$v_i^p = \sum_{j\neq i:w_{ij}>0} \frac{w_{ij}}{d_{ii}} v_j^{p-1} + (a-1)v_i^{p-1} = d_{ii}^{-1} \sum_{j\in I(i)} w_{ij} v_j^{p-1} + (a-1)v_i^{p-1}. \tag{8}$$

Finally, using the iteratively computed v_i^p value, the vertex-centric score S_{AV} can be easily obtained:

$$S_{AV}(i,V_C) = d_{ii}^{\frac{1}{2}} v_i^p \tag{9}$$

Therefore, in a vertex-based graph computation model, at every update step it is sufficient to take into account for each vertex $i \in V$:

- the old value v_j^{p-1} of all neighboring vertices $j \in I(i)$,
- the weight w_{ij} of all incoming edges,
- the weighted in-degree d_{ii}

and then use the previous rule for the updating. This kind of computation scheme can be implemented in any vertex-based graph analytics programming framework.

4 Experimental settings

We applied our methods based on the local implementation of network-based algorithms and secondary memory-based computation to the multi-species protein function prediction in eukarya. In all the experiments we implemented the network-based methods using *GraphChi*, a software library for large-scale graph computation using secondary memory [8]. All the experiments have been performed using off-the-shelf desktop computers with a limited amount of RAM memory (4 GB). It is worth noting that in these experimental conditions random-walk algorithms that store in primary memory the adjacency matrix of the graph described in Section 4.1 run out-of-memory due to the limited amount of available RAM.

In the remainder of this section we summarize the experimental set-up and the characteristics of the data, and then we compare the empirical computational time and the performance of secondary memory-based implementations of network learning algorithms for AFP.

Table 1 Selected species from the core region of the STRING protein networks database

NCBI taxon ID.	Species	n. proteins
3218	Physcomitrella patens	10352
3702	Arabidopsis thaliana	23576
7227	Drosophila melanogaster	12845
7739	Branchiostoma floridae	16418
8364	Xenopus (Silurana) tropicalis	13678
9031	Gallus gallus	13119
9258	Ornithorhynchus anatinus	13333
9606	Homo sapiens	20140
9615	Canis lupus familiaris	16912
10090	Mus musculus	20023
13616	Monodelphis domestica	15409
39947	Oryza sativa Japonica	13330
69293	Gasterosteus aculeatus	13307

Table 2 Average per-term empirical time complexity between the compared local and secondary memory-based network learning methods implementations

Local algorithm	per-term empirical time complexity evaluation
1-step RW	21.46s
2-step RW	33.19s
3-step RW	46.69s
1-step RWK (Average score)	21.05s
2-step RWK (Average score)	34.05s
3-step RWK (Average score)	46.25s

4.1 Dataset description and performance evaluation

In order to test the ability of the proposed local methods to scale to large multi-species networks, we constructed a large genes network (hereafter referred to as Eukarya-net). All the proteins interactions composing Eukarya-net were downloaded in precomputed form from the STRING protein-protein interactions database. STRING (http://string-db.org/) is a collection of networks composed by real and predicted gene-gene interactions (based on genetic data, physical data and literature data) and aims at providing a global view of all the available interaction data, including lower-quality data and/or computational predictions for as many organisms as feasible. Starting from the STRING interaction data (version 9.05), we selected all the Eukaryotic species having $10,000$ or more proteins. The selected Eukaryotic species are listed in Table 1.

As class labels for the proteins included in Eukarya-net we used the Gene ontology [1] (GO) annotations available in STRING (version 9.05). The STRING website provides flat text files containing a mapping from GO annotations to STRING proteins and a STRING internal confidence score for each GO annotation, ranging from 1 (low confidence) to 5 (high confidence). While extracting the GO labels we considered only the annotations with confidence score 5. We then filtered out all the GO terms associated with less than 20 and more than 100 proteins (473 GO terms). We finally randomly selected from this set 50 GO terms.

Performance were evaluated in terms of runtime, Area under the Receiver Operating curve (AUROC), and Precision at fixed Recall levels using a canonical 5-fold stratified cross validation scheme.

4.2 Results

Table 2 summarizes the average per-term runtime required to complete a 5-fold cross validation with the Eukarya-net involving more than $200,000$ proteins of 13 multi-cellular eukarya organisms.

We observe that the average computational time is very similar for both the RW and RWK-based kernelized functions secondary memory-based implementations. The performance (see Table 3) in terms of the average precision at fixed recall levels obtained in this test are relatively low, especially when compared with the high average AUC obtained with the RW at 1, 2 and 3 steps. The observed relatively low precision can be explained by taking into account that it is more negatively affected by class unbalance and, in the Eukarya-net network task, the positives are at most 100 while the number of vertices in the network is $202,442$ (i.e. the positives are less than 0.05% of the vertices at best). Note that in this case the 2-steps RW achieves the best AUROC results: it is likely that these results could be due to the connections between nodes representing proteins coming from different species but further evaluation is required in order to clarify the observed results.

Table 3 Average AUC, precision at 20% recall (P20R) and precision at 40% recall of the compared local and secondary memory-based network learning methods across 50 GO terms. Performance estimated through 5-fold cross-validation.

Algorithm	AUROC	P20R	P40R
RW - 1 step	0.8601	0.1449	0.0943
RW - 2 steps	0.9667	0.1329	0.0929
RW - 3 steps	0.9598	0.0927	0.0785
RWK 1 step (Average score)	0.9106	0.2115	0.1422
RWK 2 steps (Average score)	0.9902	0.2670	0.1605
RWK - 3 steps (Average score)	0.9680	0.2314	0.1498

As we can see the best performances in terms of AUROC are obtained at two steps also with the RWK-based average score function. While the differences in AUROC performances between the classical and kernelized random walks based methods are not so big, the same does not hold with respect to the performances in terms of P20R and P40R (respectively precision at 20% and 40% recall), where the random walk kernel clearly outperforms the local classical random walk-based gene function predictor.

Overall, these results show that the secondary memory-based implementation of kernelized score functions allow the analysis of big networks using off-the-shelf desktop computers, and achieve results competitive with the classical random walk algorithm in the multi-species prediction of protein functions.

5 Conclusions

In this work we presented a novel secondary memory-based implementation of a Random walk kernel network learning method. More precisely, we developed a novel local and secondary memory-based algorithm able to compute a kernelized score function (the average score) embedding a p-step random walk kernel. The proposed algorithm has been applied to the prediction of protein functions in the context of a multi-species large biological network involving more than $200,000$ proteins. The experimental results show that the local version of the kernelized version of the random walk exhibits an empirical time complexity comparable with a local RW and secondary memory-based within network learning algorithm, and outperforms the classical random walk algorithm for the multi-species prediction of protein functions. From a more general standpoint we believe that "local" versions of network-based algorithms, together with an efficient secondary memory-based implementation, can open new avenues for the analysis of big and complex networks in computational biology, without the mandatory need of complex clusters of computers or expensive stand-alone workstations equipped with very large RAM memory.

References

1. M. Ashburner et al. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1), 2000.
2. I. Friedberg. Automated protein function prediction-the genomic challenge. *Brief Bioinform.*, 7:225–242, 2006.
3. J. Gillis and P. Pavlidis. Characterizing the state of the art in the computational assignment of gene function: lessons from the first critical assessment of functional annotation (cafa). *BMC Bioinformatics*, 14(3):S15–10, 2013.
4. J.E. Gonzalez et al. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proc. of the 10th USENIX Conf. on Operating Systems Design and Implementation*, pages 17–30, 2012.
5. W.S. Han et al. Turbograph: a fast parallel graph engine handling billion-scale graphs in a single PC. In *Proc. of the 19th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 77–85, 2013.
6. Y. Jiang et al. An expanded evaluation of protein function prediction methods shows an improvement in accuracy. *Genome Biology*, 17(184), 2016.
7. A. Kuzniar et al. The quest for orthologs: finding the corresponding gene across genomes. *Trends Genet.*, 24(11):539–551, 2008.
8. A. Kyrola et al. Graphchi: large-scale graph computation on just a pc. In *Proceedings of the 10th USENIX Conf. on Operating Systems Design and Implementation*, pages 31–46, 2012.
9. W. Liu, J. Wang, and S.F. Chang. Robust and scalable graph-based semisupervised learning. In *Proc. IEEE*, volume 100, pages 2624–2638, 2012.
10. L. Lovasz. Random Walks on Graphs: a Survey. *Combinatorics, Paul Erdos is Eighty*, 2:1–46, 1993.
11. Y. Low et al. Graphlab: a new parallel framework for machine learning. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2010.
12. G. Malewicz et al. Pregel: a system for large-scale graph processing. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 135–146, 2010.
13. M. Mesiti, M. Re, and G. Valentini. Think globally and solve locally: secondary memory-based network learning for automated multi-species function prediction. *GigaScience*, 3(1):1, 2014.
14. S. Mostafavi et al. GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function. *Genome Biology*, 9(S4), 2008.
15. M. Re, M. Mesiti, and G. Valentini. A Fast Ranking Algorithm for Predicting Gene Functions in Biomolecular Networks. *IEEE ACM Transactions on Computational Biology and Bioinformatics*, 9(6):1812–1818, 2012.

16. M. Re and G. Valentini. Cancer module genes ranking using kernelized score functions. *BMC Bioinformatics*, 13(Suppl 14/S3), 2012.
17. A.J. Smola and I.R. Kondor. Kernel and regularization on graphs. In *Proc. of the Annual Conf. on Computational Learning Theory*, LNCS, pages 144–158. Springer, 2003.
18. G. Valentini, G. Armano, M. Frasca, J. Lin, M. Mesiti, and M. Re. RANKS: a flexible tool for node label ranking and classification in biological networks. *Bioinformatics*, 32:2872–2874, 2016.
19. J. Webber et al. A programmatic introduction to neo4j. In *Proc. of the 3rd Annual Conf. on Systems, Programming, and Applications: Software for Humanity*, pages 217–218, 2012.