Bioinformatics, 32(14), 2016, 2159–2166
doi: 10.1093/bioinformatics/btw223
Advance Access Publication Date: 29 April 2016
Original Paper

Systems biology

APPAGATO: an APproximate PArallel and stochastic GrAph querying TOol for biological networks

Vincenzo Bonnici[1], Federico Busato[1], Giovanni Micale[2], Nicola Bombieri[1], Alfredo Pulvirenti[3] and Rosalba Giugno[1,3,*]

[1]Department of Computer Science, University of Verona, Strada Le Grazie 15 - 37134, Verona, [2]Department of Math and Computer Science, University of Catania, Viale a. Doria 6 - 95125, Catania and [3]Department of Clinical and Experimental Medicine, University of Catania, via Palermo, 636 - 95122, Catania

*To whom correspondence should be addressed

Associate Editor: Alfonso Valencia

Received on October 20, 2015; revised on March 7, 2016; accepted on April 10, 2016

Abstract

Motivation: Biological network querying is a problem requiring a considerable computational effort to be solved. Given a target and a query network, it aims to find occurrences of the query in the target by considering topological and node similarities (i.e. mismatches between nodes, edges, or node labels). Querying tools that deal with similarities are crucial in biological network analysis because they provide meaningful results also in case of noisy data. In addition, as the size of available networks increases steadily, existing algorithms and tools are becoming unsuitable. This is rising new challenges for the design of more efficient and accurate solutions.

Results: This paper presents *APPAGATO*, a stochastic and parallel algorithm to find approximate occurrences of a query network in biological networks. *APPAGATO* handles node, edge and node label mismatches. Thanks to its randomic and parallel nature, it applies to large networks and, compared with existing tools, it provides higher performance as well as statistically significant more accurate results. Tests have been performed on protein–protein interaction networks annotated with synthetic and real gene ontology terms. Case studies have been done by querying protein complexes among different species and tissues.

Availability and implementation: *APPAGATO* has been developed on top of CUDA-C++Toolkit 7.0 framework. The software is available online http://profs.sci.univr.it/∼bombieri/APPAGATO.

Contact: rosalba.giugno@univr.it

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Technological advances have led to the inference and the validation of structured interaction networks involving genes, proteins, drugs, phenotype and diseases (Barabasi and Oltvai, 2004; Kelley *et al.*, 2003; Panni and Rombo, 2015). According to the data type, such networks are referred to as (i) protein–protein interaction (PPI) networks representing either physical or functional interactions among proteins; (ii) gene regulatory networks that express how the activity of genes is regulated; (iii) metabolic networks describing biochemical reactions between chemical compound of cells; and (iv) signalling networks representing inner/outer cell communications.

A typical example that highlights the advantages and possibilities of analysing interaction relationships is protein function prediction. Although sequence homology is commonly used to functionally annotate proteins, a great amount of them remained uncharacterized (Yu *et al.*, 2013). In this context, different algorithms and tools that

compare biological networks have been applied to predict novel protein functions (Jiang *et al.*, 2011; Malod-Dognin and Pržulj, 2015; Wang *et al.*, 2013).

In disease studies, genes showing similar phenotypes tend to be neighbours in protein interaction networks and their aggregation in connected sub-networks is effective to detect biomarkers (Creixell *et al.*, 2015; Fortney *et al.*, 2010; Ideker *et al.*, 2002). Also, finding similar functional and topological sub-networks helps analysing the conservation among species (Lim *et al.*, 2006). In all these applications, graphs serve as the underlying structures for representing biological networks (For the sake of clarity, in this article, we use the terms *graph* and *network* indistinctly.) and graph algorithms solve problems such as network alignment, network querying, motif extractions and network perturbation (Ciriello *et al.*, 2012; Ma and Gao, 2012; Malod-Dognin and Pržulj, 2015; Panni and Rombo, 2015).

In this article we address the problem of *approximate network querying*, which finds, in a *target* network, *similar* occurrences of a so-called *query* network. The notion of similarity takes into account both the similarities between target nodes and query nodes, and a cost measuring the differences of nodes and their connections. An approximate network querying algorithm has to find the query occurrences, among all possible, with the maximum combined similarity.

Querying tools that deal with similarities are effective in biological network analysis because they provide results also in case of noisy data. They are also suitable in the case of partial knowledge of users when formulating queries. Furthermore, they can be used to compare data from different species where some fundamental and functional structures are partially preserved.

Solving approximate network querying implies applying instances of subgraph isomorphism, which is a NP-complete problem (Dost *et al.*, 2008). In literature, several heuristics have been proposed to solve such a problem in reasonable running time. Examples include restricting the topology of queries to paths or trees (Dost *et al.*, 2008; Kelley *et al.*, 2004; Pinter *et al.*, 2005; Shlomi *et al.*, 2006), applying network alignment strategies (Gulsoy and Kahveci, 2011; Tian *et al.*, 2007, 2008; Yuanyuan and Patel, 2008), dealing with node similarities and ignoring the query topology (Blin *et al.*, 2010; Bruckner *et al.*, 2010), fixing the topology and computing differences of node labels (Hong *et al.*, 2015). Other methods consist of building indexes to reduce the query time (Khan *et al.*, 2013; Zhang *et al.*, 2009); filtering the set of possible similar target data (Hong *et al.*, 2015; Pienta *et al.*, 2014; Sahraeian and Yoon, 2012); to find only exact occurrences of the query in the network (Bonnici and Giugno, 2016; Bonnici *et al.*, 2013; Cordella *et al.*, 2004; Sun *et al.*, 2012); finding the largest part of the query exactly contained in the target graph and replace the query edges not present in the target with paths (Pienta *et al.*, 2014).

We have created *APPAGATO*, a tool that relies on an iterative sampling method (Lawrence *et al.*, 1993; Micale *et al.*, 2014), to compute functional and topological similarities between a query and a target network. Through a matching probability matrix and a weighted sampling procedure, it selects a seed from which the query–target matching starts. Then, by associating a cost to each approximation, it iteratively extends the match by selecting the approximations with the lowest possible cost. The algorithm runs K times and returns a set of K approximate matches. *APPAGATO* performs approximate network querying by considering the topology of query, taking into account node and edge deletions together with differences on node labels.

To speed-up the querying process in large biological networks, *APPAGATO* has been implemented to run on graphics processing units (GPUs). Owing to their low cost, high-performance and easy integration to any personal computer, GPUs have been increasingly applied to accelerate bioinformatics problems (Dematté and Prandi, 2010; Vouzis and Sahinidis, 2011; Zhao and Chu, 2014). Our aim is to handle large biological networks in a reasonable time yielding accurate results. We compare *APPAGATO* with *RESQUE* (Sahraeian and Yoon, 2012) and *NeMa* (Khan *et al.*, 2013) because, to the best of our knowledge, they are the most efficient and stable tools in literature close to *APPAGATO* on both the problem they address and on the approximation concept they assume. We run the tools with different PPI networks as input and compared nodes by using similarities of protein sequences and functional gene ontology annotations. We extensively compare the tools in terms of running time, costs of returned matches and accuracy in finding protein complexes among different species. The results show that *APPAGATO* outperforms the other two tools, yielding more accurate results on large PPI networks.

2 Materials and methods

2.1 Definitions and notations

A graph G is a pair (V, E), where V is the set of nodes and $E \subseteq (V \times V)$ is the set of edges. If $(u, v) \in E$, we say that v is a neighbour of u. G is *undirected* iff $\forall (u, v) \in E$, then $(v, u) \in E$, i.e. u is a neighbour of v and vice versa. The degree of a node u, $Deg(u)$, is the number of its neighbours. Given a set of labels A, the function $Lab : V \rightarrow A$ assigns a label to each node of G. We assume that graphs are undirected and labelled only on nodes.

2.1.1 Exact SubGraph Isomorphism

Let $Q = (V, E)$ and $T = (V', E')$ be two graphs, named *query* and *target*, respectively. The *exact SubGraph Isomorphism* (SubGI) problem aims to find an injective function, $M : V \rightarrow V'$, which maps each node in Q to a unique node in T, such that $\forall (u, v) \in E$: (i) $(M(u), M(v)) \in E'$; (ii) $Lab(u) = Lab(M(u))$; (iii) $Lab(v) = Lab(M(v))$. A solution of the SubGI problem can be represented as the set $m = \{(v_1, M(v_1)), (v_2, M(v_2)), \ldots, (v_{|V|}, M(v_{|V|}))\}$, called a *match* of Q in T. Q may have different maps m_i in T.

2.1.2 Inexact SubGraph Isomorphism and matching costs

In this article, we deal with the *Inexact SubGraph Isomorphism* (ISubGI) problem (Here called also approximate subgraph querying.), which is a variant of the SubGI problem, and in which we admit node and edge mismatches. A mismatch occurs when (i) two nodes with different labels are mapped through a similarity function, or (ii) a query edge or (iii) a query node is missing in the target graph. The absence of a node implies mismatches for all its edges. A cost c is associated to each mismatch. For the sake of simplicity, the same cost $c = 1$ is associated to each of the three types of mismatch.

We denote with $C = \sum c$ the total cost of mismatches between Q and T. The goal of the ISubGI problem is to find an injective function $M : V \rightarrow V'$, such that C is minimized. In this case, a solution for the ISubGI, $m = \{(v_1, M(v_1)), (v_2, M(v_2)), \ldots, (v_k, M(v_k))\}$ with $k \leq |V|$, is called an *approximate match* with a cost $C \geq 0$.

Let $Q_m = (V_m, E_m)$ be the subgraph of query Q that has been mapped in the match m, that is, $V_m = \{v \in V : (v, M(v)) \in m\}$ and $E_m = \{(u, v) \in E : (u, M(u)) \in m\, (v, M(v)) \in m\, (M(u), M(v)) \in E'\}$. We define $V\bar{m} = V \backslash V_m$ and $E\bar{m} = E \backslash E_m$, the nodes and the edges in Q, respectively, that have not been matched in m. Let $S_{|V| \times |V'|}$ be the

label similarities between each node $q \in Q$ and $t \in T$. The *label similarity* values belong to the interval $[0, 1]$. The computation of S is application dependent. In the case of PPI networks, the similarity can be based on sequences, functional or structural protein similarity.

For example, establishing the conservation of a protein-complex CO of the species A within the species B consists of searching the subgraph Q_{CO}, extracted from the PPI of A (named G_A), into the PPI of B (named G_B). The two PPIs may have different proteins (i.e. nodes with different names), but with similar function, detectable by looking at sequence similarities. An ISUBGI algorithm must search for occurrences of Q_{CO} in G_B that minimize sequences and topology differences. We conclude that CO is conserved in B if we find highly similar occurrences.

The total matching cost C is obtained by summing all node and edge costs and by normalizing them over the number of query elements, as follows:

$$C = \frac{\sum_{q \in V_m} (1 - S(q, M(q)) + |V\bar{m}| + |E\bar{m}|)}{|V| + |E|} \quad (1)$$

2.2 The *APPAGATO* algorithm

The method consists of the following three main phases.

2.2.1 Phase 1: Computation of matching probability matrix

Before starting the search, *APPAGATO* computes a matrix P of matching probabilities between all possible node pairs $< q, t >$ ($q \in Q$ and $t \in T$), by combining (i) the label similarity $S(q, t)$, (ii) the degree similarity $D(q, t)$ and (iii) the breadth-first similarity $BFS_{Sim}(q, t)$. The label similarity has been defined in Section 2.1.2. In *APPAGATO*, the label similarity matrix, S, may be provided as input by the user. Alternatively, *APPAGATO* computes a Boolean similarity function to compare node labels. It assigns 1 if labels are identical, 0 otherwise. The degree similarity is a binary function D(q,t) = 1 if $Deg(q) \leq Deg(t)$, otherwise it is 0. $BFS_{Sim}(q, t)$ is computed by performing breadth-first visits (BFSs) of the query and target graphs by starting from q and t and evaluating label and degree similarities of the visited nodes, level by level. The maximum depth of the BFS visits is a user-defined parameter l_{max}, with $l_{max} \geq 1$. Given a node x, and a level $l \leq l_{max}$ we denote with $BFS_l(x)$ the set of nodes at level l in the BFS tree rooted at x. An edge $e = (u, v)$ in the BFS tree of q is defined *matchable* iff there exists an edge $e' = (u', v')$ in the BFS tree of t such that $S(u, u')$ and $S(v, v')$ are not 0 and $D(u, u') = D(v, v') = 1$. We denote with $MaxMatch(BFS_l(q), BFS_l(t))$ a maximal set of matchable edges in the BFS tree of q at level l, with respect to the BFS tree of level l rooted in t. The BFS similarity between q and t assumes values in $[0, 1]$ and is defined as follows:

$$BFS_{Sim}(q, t) = \frac{\sum_{l=1}^{l_{max}} l \times |MaxMatch(BFS_l(q), BFS_l(t))|}{\sum_{l=1}^{l_{max}} l \times |BFS_l(q)|} \quad (2)$$

Matching probability matrix. The three similarity values are linearly combined in $MScore(q, t) = S(q, t) + D(q, t) + BFS_{Sim}(q, t)$ and normalized to get the matching probability:

$$P(q, t) = \frac{MScore(q, t)}{\sum_{z \in T} MScore(q, z)} \quad (3)$$

Equation (3) ensures that $\sum_{t \in T} P(q, t) = 1$. In phase 2, the probability matrix is used as a transition matrix within an iterative sampling to extract the best possible matches. The upper side of Figure 1 shows an example of such a matrix computation.

2.2.2 Phase 2: Seed selection

APPAGATO searches the first pair of nodes to be matched by randomly selecting q and t according to the probabilities defined in Equation (3) (see the example of Figure 1).

2.2.3 Phase 3: Extension

Gibbs sampling is used to navigate within a Markov chain, where each state represents a possible query-target node match. The initial state corresponds to the seed selected in phase 2. The sampling method iteratively performs a transition from a state to another, by replacing the query–target nodes pair with a new one, according to a properly defined transition probability. As an example, Figure 1 shows the first two iterations of the extension phase. Transition probabilities are defined by starting from similarity scores, and by taking into account the connections of candidate nodes with already matched nodes. Let Q_m and T_m be the set of query–target matched nodes at a certain step of the extension process. We denote with $Q_m[i]$ ($T_m[i]$) the i-th query (target) node added to the partial match. Let q be a query node neighbour to at least one node in Q_m and t be a target node neighbour to at least one node in T_m. We represent the set of connections between q and the nodes in Q_m through a bit vector $CP(q)$ of $|Q_m|$ elements, called *connection profile* of q, where the i-th element is defined as follows:

$$CP(q)[i] = \begin{cases} 1 & \text{if } (q, Q_m[i]) \in E \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

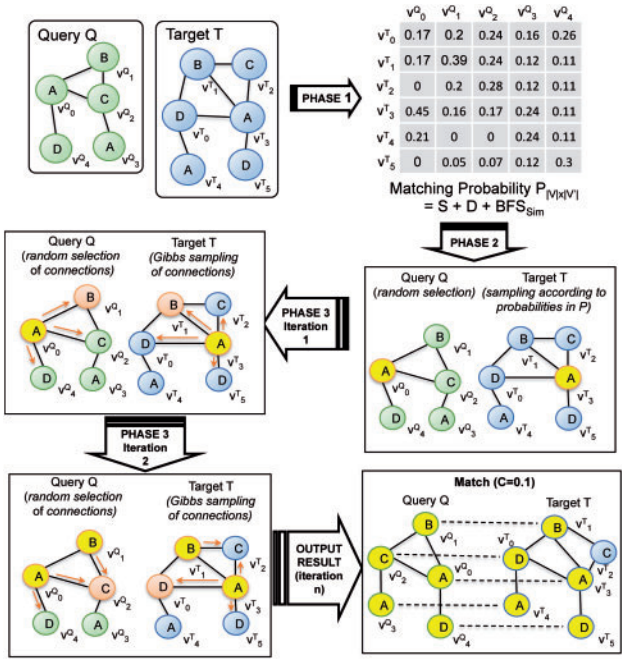

Fig. 1. The *APPAGATO* approximate matching algorithm

We define $CP(t)$ in the same way. The *connection profile similarity* between q and t is the corresponding number of equal bits in the connection profiles of q and t:

$$CP_{Sim}(q,t) = \frac{|\{1 \leq i \leq |CP(q)| : CP(q)[i] = CP(t)[i]\}|}{|CP(q)|} \quad (5)$$

The overall similarity score is $MScoreExt(q,t) = S(q,t) \times CP_{Sim}(q,t)$. The result value is normalized to obtain the final transition probability (Notice that *MScore* is not used in the extension phase. *MScoreExt* strongly influences the convergence of the approach (Lawrence *et al.*, 1993; Micale *et al.*, 2014).):

$$P_T(q,t) = \frac{MScoreExt(q,t)}{\sum_{z \in T} MScoreExt(q,z)} \quad (6)$$

After a number of iterations, n, which is a user-defined parameter, the algorithm returns the reached match between the query and the target node. The quality of such a match is evaluated by summing the costs of node and edge mismatches between Q and T. *APPAGATO* does not require any user-defined threshold for the maximum allowed cost of a match. In Figure 1, the approximate match has only a label mismatch, v_2^Q, whose label C is mapped with v_0^T having label D, and the cost of the match is $C = 0.1$, computed by applying Equation (1). *APPAGATO* iterates K times phases 2 and 3 and, in each iteration, it starts the sampling procedure from a different seed. Each run of *APPAGATO* always returns K solutions (approximate matches), each one with the corresponding cost.

2.3 The *APPAGATO* parallel implementation for GPUs

APPAGATO has been implemented to take advantage of massively parallel GPU architectures. All the processing phases presented in Section 2.2 have been implemented through different CUDA *kernels* (http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html), which are invoked by the *host* CPU. This allows performing the most compute-intensive tasks of the search algorithm on the GPU *device*. As for the parallel implementation paradigm for GPUs, each kernel is executed in parallel by several *blocks* of *threads*. Thread blocks spread and run concurrently and independently over *streaming multiprocessors*. Threads of the same block efficiently cooperate through fast shared memory and by synchronizing their execution through extremely fast (i.e. HW implemented) barriers. Groups of 32 threads of the same block are called *warps*. Each warp executes one kernel instruction at a time in parallel on different data (i.e. single instruction multiple data-SIMD architecture) over the many stream processors (cores) of the GPU device. A warp scheduler efficiently switches between warps with the aim of hiding the latency of thread accesses to the memory.

Given the query and the target graphs, Q and T, the three phases have been implemented as follows (see Figure 2).

2.3.1 Phase 1: Parallel computation of matching probability matrix

Computing the matching probability matrix is one of the most computation-intensive part of the whole algorithm. It requires $|V| \times |V'|$ computations of Equation (3) and, in particular, $O(|V| + |V'|)$ BFSs over Q and T and the corresponding comparisons between the visited edges (Equation (2)).

APPAGATO implements such a phase through a customized version of *BFS-4K* (Busato and Bombieri, 2015), a parallel implementation of BFS for GPU architectures. *BFS-4K* relies on the concept of *frontier* (Cormen *et al.*, 2009) (i.e. a FIFO queue that contains the nodes to be visited at each BFS iteration) to implement the graph visit. Through the frontier-based visiting, *BFS-4K* allows Equation (3) to be performed over two levels of parallelism: Each parallel warp of a block is mapped to each node of the frontier, and each parallel thread of a warp is mapped to each outgoing edge from a frontier node.

APPAGATO extends the BFS visit over a third level of parallelism, by running a total number of $|V| + |V'|$ independent BFSs in parallel, one for each node of Q and T. This is done by allocating one block of threads per BFS. The block allocation is automatically done at runtime. A total number of $|V|$ thread blocks perform, in parallel, $|V|$ BFSs (of depth l_{max}) for the query graph. The result consists of *source–destination matrices*, one per node, which are stored in the global memory (the left-most side of Figure 2 shows an example, assuming $l_{max} = 2$). Each matrix contains information on the labels of such edges visited during the BFS from the node along l_{max} levels. In the example of Figure 2, the V_0^Q matrix contains information on the edges of the first-level BFS ($A - B$, $A - C$, $A - D$) as well as the edges of the second-level BFS ($B - A$, $B - C$, $C - A$, $C - B$, $D - A$).

Similarly, and concurrently, a total number of $|V'|$ thread blocks perform the BFSs for the target graph. The result consists of a set of

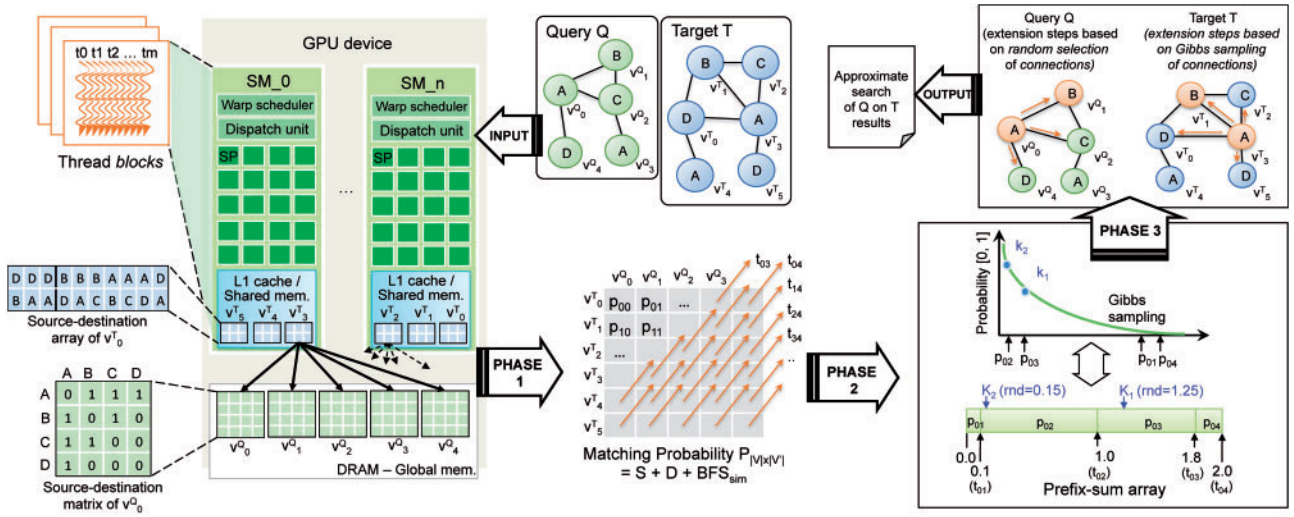

Fig. 2. The parallel search of *APPAGATO* on the GPU device

source–destination arrays, one per node, which are stored in the device *shared memory*. This allows an extremely fast memory access for the following comparisons between the generated node structures. The array data structure has been chosen, as it allows to represent in a more compact way the source–destination information of T in the limited shared memory. In contrast, the matrix data structure has been chosen as it guarantees a faster access to the source–destination information of Q, to be stored in the larger global memory.

Finally, $|V'|$ thread blocks compare, in parallel, their own source–destination array stored in the local shared memory with all the source–destination matrices in global memory. Such a data structure organization over the GPU memory hierarchy allows the complexity of Equation (3) to be reduced from $O(|V| \times |V'|)$ as for the sequential algorithm, to a parallel complexity of $O(1)$. The result of Phase 1 is the matrix $P_{|V|x|V'|}$, which is stored in the device global memory (see centre part of Figure 2).

2.3.2 Phase 2: Parallel seed selection

APPAGATO emulates the Gibbs sampling to select the K seeds for the successive extension phase. The emulation relies on two parallel primitives, *prefix-sum* (Billeter *et al.*, 2009; Harris *et al.*, 2008) and *weighed random number generation* (https://developer.nvidia.com/curand), which are efficiently implemented in the literature for GPUs. Given the similarity value of each query–target node pair p_{xy} of $P_{|V|\times|V'|}$, *APPAGATO* performs the parallel prefix-sum of such values through $|V| \times |V'|$ threads (i.e. one thread per similarity value). The result is a prefix-sum array, in which each element is associated to a thread and the corresponding similarity value. As an example, Figure 2 shows the prefix-sum array of four threads, t_{01}, t_{02}, t_{03} and t_{04}, having similarity value $0.1, 0.9, 0.8$ and 0.2, respectively. The array elements have been depicted through different sizes to better represent the corresponding similarity values. Then, all the threads generate a random sequence of K values in the interval $[0, \sum p_{xy}]$ (i.e. $[0, 2]$ in the example). The parallel primitive for the random number generation allows the threads to share the generation seed and, as a consequence, to generate the same sequence of random values. This allows the threads to concurrently recognize whether the own boundaries in the prefix-sum array include any randomly generated value. In the example, the sequence of random values $K_1 = 1.25$ and $K_2 = 0.15$ leads to the pair of nodes (v_0^Q, v_3^T) and (v_0^Q, v_2^T) associated to threads t_{03} and t_{02}, respectively, to be selected for the extension phase.

2.3.3 Phase 3: Parallel extension

The extension phase has been implemented through primitives of BFS, prefix-sum, weighed random number generation over different levels of parallelism. As a first level, the K query-target nodes selected in phase 2 are mapped to thread blocks (i.e. one pair of query-target nodes per block). They are concurrently processed as follows. Given a node pair (e.g. (v_0^Q, v_3^T) in Figure 2) the two nodes are processed in parallel by two thread warps (second level of parallelism). The two warps perform a one-step parallel BFS (third level of parallelism) on Q and T, respectively, to visit the neighbour nodes (i.e. candidate *connections*) of v_0^Q and v_3^T. The result is two frontiers of neighbours ($\{v_1^Q, v_2^Q, v_4^Q\}$ and $\{v_0^T, v_1^T, v_2^T, v_5^T\}$ in the example). One step of extension over Q performs through a random selection of a node (connection) from the first frontier (v_1^Q in the example). For such a node, *APPAGATO* generates the connection profile through a one-step parallel BFS. Such a connection profile strongly affects the extension over T, which is performed as follows. Starting from all the nodes of the second frontier, *APPAGATO* (i) runs one step of parallel BFS (one per node), (ii) generates the connection profiles of the visited nodes and (iii) generates the connection profile similarity of each of such nodes with the connection of Q. Through an emulation of the Gibbs sampling similar to that implemented in phase 2, *APPAGATO* selects the new connection for T. The algorithm iterates over the new pair of nodes (i.e. connection of Q and connection T) for a total number $n = |V|$ iterations.

2.4 Datasets

Physical Interaction Networks. We used the PPI networks taken from the STRING v10.0 databases (Szklarczyk *et al.*, 2011) of three species: *Mus musculus*, *Homo sapiens* and *Danio rerio*. These networks differ significantly in size (number of nodes and edges) and density (i.e. the average number of neighbours per node). For each network, we used up to 250 synthetic labels and gene ontologies annotation downloaded from *BioDbNet* (http://biodbnet.abcc.ncifcrf.gov). This yielded 12 different PPIs (i.e. three species, each one labelled in four different ways). We constructed the queries by randomly extracting sets of 100 connected subgraphs, from each network, by varying the size of the queries up to 128 nodes. In this dataset, the similarities matrix $S_{|V|x|V'|}(q, t) = 1$ if $Lab(q) = Lab(t)$, otherwise is set to 0.

Functional Interaction Networks. The STRING database reports, among two proteins and beside the direct physical interactions used above, indirect functional relations such as structural similarity, similarity between the transcript sequences encoding them and functional correlations. It gives a score, ranging from 0 (namely no relation is known) to 999, which combines physical and functional (i.e. co-expression data analysis) interactions. We constructed a second dataset by taking into account such a combined score. We extracted four PPI networks related to the species *M.musculus*, *H.sapiens*, *D.rerio* and *Saccharomyces cerevisiae*. We fixed the interaction score threshold at 998 to get few but highly functional related interactions within each network. As queries, we used 10 human protein complexes taken from the CORUM database (Ruepp *et al.*, 2010). Because CORUM only reports the set of proteins belonging to a given complex, and not their interactions, we reconstructed the topology of the complex by taking into account the interactions reported in the full STRING database with respect to the *H.sapiens* species. Finally, we labelled target and query nodes with the protein sequences. We computed the query-target node similarities matrix $S_{|V|x|V'|}$, by making use of *CUDASW* (http://cudasw.sourceforge.net), which implements a parallel version for GPUs of the Smith-Waterman algorithm for local alignment of sequences. We normalized the matrix by row to set to 1 the maximum similarity of the target and query node. We used this dataset to investigate the biological significance of the results. The approximate subgraph matching algorithms were capable to identify functional conservation of protein complexes among different species. We refer the reader to Supplementary Section 1 and Tables S1–S2 for more details.

3 Results and discussion

We compared *APPAGATO* with *NeMA* (Khan *et al.*, 2013) and *RESQUE* (Sahraeian and Yoon, 2012) on both the physical and functional datasets described in Section 2.4. All the tools solve ISubGI by taking into account the query topology. Unless differently specified, with the term *APPAGATO* we refer to its implementation on top of CUDA. In the Supplementary, Section 2, we report details on the *APPAGATO* implementation and tuning of parameters

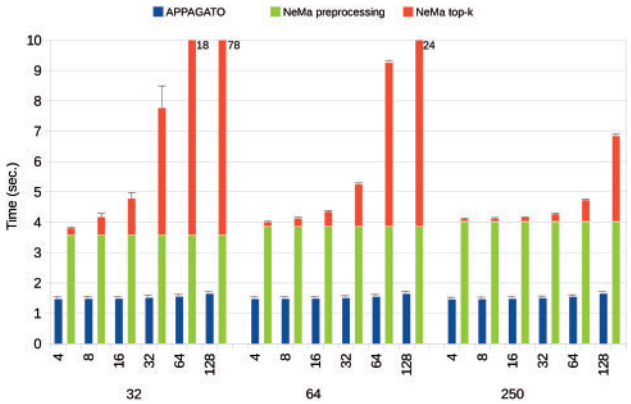

Fig. 3. The running time comparison between *APPAGATO* and *NeMa* on the *D.rerio* PPI network, randomly labelled with 32, 64 and 250 labels. Chart values report the average time on 100 queries. Queries are grouped with respect to the number of nodes, namely 4, 8, 16, 32, 64, 128. For each query, the tools have been run to find 10, 50 and 100 matches

(Supplementary Figs S1–S3), we assess the robustness of *APPAGATO* over query construction (Supplementary Figs S4 and S5) and the efficiency of both sequential and parallel versions of *APPAGATO* (Supplementary Figs S6 and S7).

3.1 Performance

For the physical interaction networks, we report the comparison results only between *APPAGATO* and *NeMA*, as *RESQUE* does not support such a large dataset. Figure 3 shows the average running times of the two tools on the *D.rerio* network. In the total running time of *NeMa*, we distinguish the target preprocessing and the querying time. Note that *APPAGATO* does not perform any preprocessing step. The results show that *APPAGATO* is at least three times faster than *NeMA* in case of small queries (i.e. 4, 8, 16 nodes). The performance difference sensibly increases with larger queries. The plots clearly show that the *APPAGATO* running time is almost constant when increasing the query size and the number of labels. We do not report the comparison results on *M.musculus* and *H.sapiens* because, in those networks, the running time difference is even more evident (i.e. *NeMa* requires >10 000 s for the preprocessing phase and >6000 s for the execution phase, while *APPAGATO* always requires around 2 s). Supplementary Figure S8 in Section 3 reports the details on the *APPAGATO* running time in all the physical interaction networks, by showing its efficiency varying the number of labels, query size and network size. Figure 4 reports the comparison of *APPAGATO* with *RESQUE* on the functional interaction networks. For the sake of clarity, we do not include the *NeMa* results in the comparison because in this kind of networks, *RESQUE* outperforms *NeMa*. The performance of *RESQUE* mainly depends on the size of query and target and on the number of possible candidates for each query node. *RESQUE* requires, as an input, a similarity matrix between query and target nodes. Such a matrix can be partially defined and this affects the quality of the results. If the similarity matrix is fully defined, then the algorithm execution becomes infeasible (i.e. *RESQUE* takes hours for a single query run). Therefore, we run several tests by changing the percentage of target nodes that can match to a specific query node. Given a threshold t, we set all entries in the similarity matrix with values less than t to 0 (i.e. making them not possible candidates). We then normalized each row by the row maximum value. We chose the percentages 10%, 5% and 1% to obtain reasonable *RESQUE* running

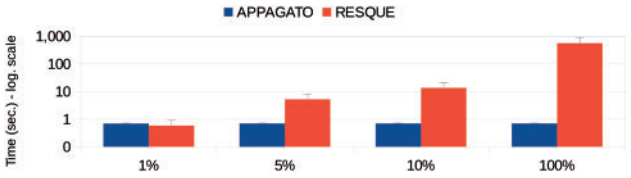

Fig. 4. Running times of *APPAGATO* and *RESQUE* on the functional interaction networks. Results are grouped by the similarity thresholds. The running time of *RESQUE* highly depends on the number of target nodes that can be matched with a query node (i.e. on the similarity threshold *t*)

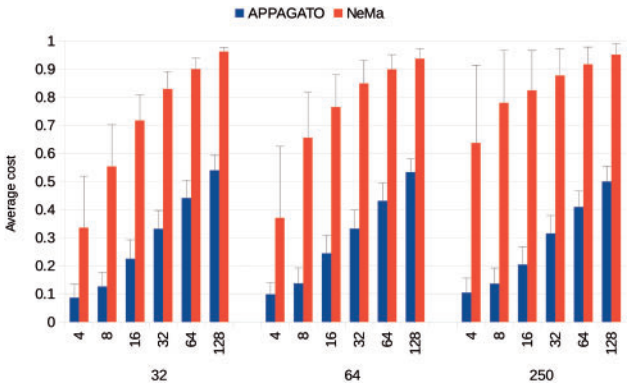

Fig. 5. Average costs (and their standard deviations) by taking into account the set of distinct output matches. Analysis have been performed on the physical interaction PPI of *D.rerio*. Results are grouped with respect to the number of target labels and query size

times (i.e. 14, 5 and 1 s, respectively. *APPAGATO* always requires around 0.69 s). The *RESQUE* running time rapidly rises as the t threshold increases. In contrast, the *APPAGATO* running times are always <1 s.

3.2 Quality measurements of matches

Figure 5 shows a comparison of the average response costs of *APPAGATO* and *NeMA* on the *D.rerio* physical PPI network. We removed the duplicated matches from the results of *APPAGATO* to avoid the bias coming from low-cost matches. Both algorithms are executed to return the best 10, 50, 100 matches. As expected, both algorithms are highly dependent on the query size. However, there is a clear difference in their output quality. The cost of *NeMa* results are often close to 1, which means they involve a high number of mismatches. In contrast, the averages of the *APPAGATO* costs range from 0.1 to 0.55. Supplementary Figures S9 and S10 in Section 3 confirm the accuracy of *APPAGATO*, also on *H.sapiens* and *M. musculus*. We measured the statistical significance of the differences between the *APPAGATO* and *NeMa* performance. We computed the *P*-values with a Wilcoxon rank-sum test together with a false discovery rate correction for multiple testing. Supplementary Figure S11 in Section 3 shows that *APPAGATO* significantly outperforms *NeMa*. The number of tested queries having lower *P*-values increases as the output size becomes larger, particularly when the number of required output matches increases.

3.3 Querying protein complexes among different species

We compared *APPAGATO* and *RESQUE* using 10 human protein complexes taken from CORUM and queried on the functional interaction dataset composed by *M.musculus*, *H.sapiens*, *Drosophila*

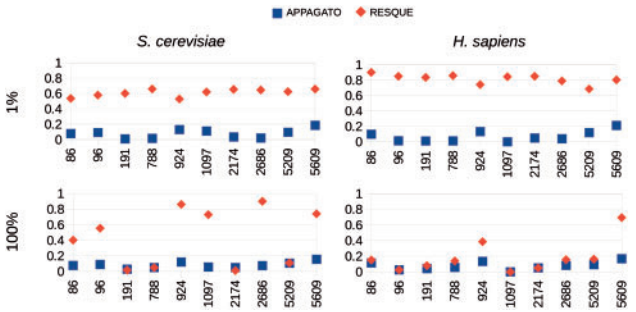

Fig. 6. A chart showing the costs of the 10 protein complexes over the *S.cerevisiae* and *H.sapiens* networks. The CORUM ID of the protein complexes is reported on the x-axis. In the top charts, the similarity threshold is equal to 1%. For those reported in the bottom side the similarity matrix has not been filtered

melanogaster and *S.cerevisiae* networks (see Fig. 6 and Supplementary Fig. S12). We test *RESQUE* using two similarity threshold values, 1% and 100%. *RESQUE* shows the main performance limitation with a similarity threshold equal to 1% on every target network, while it provides better performance by increasing the cut-off. In all cases, *APPAGATO* outperforms *RESQUE* even on the quality of the results. To confirm this, we run the Wilcoxons rank-sum tests (see Supplementary Fig. S13). For low similarity thresholds (from 1% to 10%), *APPAGATO* provides *P*-values close to 1×10^{-12}. Better *P*-values (between 1×10^{-5} and 1×10^{-6}) are shown when we defined the whole similarity matrix. Nevertheless, this turned out to be unfeasible from the running time point of view. Supplementary Figure S14 in Section 4 shows the functional coherence of results with respect to gene ontology. We computed the average *P*-value for both algorithms obtained by querying the 10 protein complexes for each of the four species. *APPAGATO* outperforms *RESQUE* on every type of target networks and similarity threshold. We refer the reader to Supplementary Sections 4 and 5 (Figs S15–S17) for details and further application of *APPAGATO* to compare disease modules over tissue-specific protein interaction networks.

4 Conclusions

We have developed *APPAGATO*, a stochastic and parallel algorithm to find approximate occurrences of a query in biological networks. *APPAGATO* deals with node, edge and node label mismatches. It is implemented for GPUs. The choice of such devices is motivated by their accessible costs, high-performance and widespread availability on any personal computer. All above features allow *APPAGATO* to compute efficiently functional and topological node similarity together with fast searching of a large number of query matching within the target graph. The results show that *APPAGATO* outperforms the existing tools in terms of running time and result accuracy and, unlike competitors, it scales also on large PPI networks.

Acknowledgement

The authors thank S M E Sahraeian and Byung-Jun Yoon for all their help to use and test their software RESQUE. We thank the authors of NeMA, Arijit Khan, Yinghui Wu, Charu C. Aggarwal and Xifeng Yan for distributing their software and their prompt support to evaluate it. We thank Dr Anna Privitera for her helpful discussion on *APPAGATO* application.

Conflict of Interest: none declared.

References

Barabasi, A.L. and Oltvai, Z.N. (2004) Network biology: understanding the cell's functional organization. *Nat. Rev. Genet.*, **5**, 101–113.

Billeter, M. *et al.* (2009) Efficient stream compaction on wide SIMD many-core architectures. In *Proceedings of the Conference on High Performance Graphics 2009*, pp. 159–166. New Orleans, Louisiana, USA.

Blin, G. *et al.* (2010) Querying graphs in protein-protein interactions networks using feedback vertex set. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **7**, 628–635.

Bonnici, V. and Giugno, R. (2016) On the variable ordering in subgraph isomorphism algorithms. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **PP**(99), 1545–5963.

Bonnici, V. *et al.* (2013) A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, **14(Suppl. 7)**, S13.

Bruckner, S. *et al.* (2010) Topology-free querying of protein interaction networks. *J. Comput. Biol.*, **17**, 237–252.

Busato, F. and Bombieri, N. (2015) BFS-4K: an efficient implementation of BFS for kepler GPU architectures. *IEEE Trans. Parallel Distrib. Syst.*, **26**, 1826–1838.

Ciriello, G. *et al.* (2012) Mutual exclusivity analysis identifies oncogenic network modules. *Genome Res.*, **22**, 398–406.

Cordella, L.P. *et al.* (2004) A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, **26**, 1367–1372.

Cormen, T. *et al.* (2009) *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts.

Creixell, P. *et al.* (2015) Pathway and network analysis of cancer genomes. *Nat. Methods*, **12**, 615–621.

Dematté, L. and Prandi, D. (2010) Gpu computing for systems biology. *Brief. Bioinform.*, **11**, 323–333. cited By 56.

Dost, B. *et al.* (2008) Qnet: a tool for querying protein interaction networks. *J. Comput. Biol.*, **15**, 913–925.

Fortney, K. *et al.* (2010). Method inferring the functions of longevity genes with modular subnetwork biomarkers of Caenorhabditis elegans aging. *Genom Biol.*, **13**.

Gulsoy, G. and Kahveci, T. (2011) RINQ: reference-based indexing for network queries. *Bioinformatics*, **27**, i149–i158.

Harris, M. *et al.* (2008). *GPU Gems 3: Parallel Prefix Sum (Scan) with CUDA*. Addison Wesley Professional, chapter 3.

Hong, L. *et al.* (2015) Subgraph matching with set similarity in a large graph database. *IEEE Trans. Knowl. Data Eng.*, **27**, 2507–2521.

Ideker, T. *et al.* (2002) Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, **18(Suppl. 1)**, S233–S240.

Jiang, X. *et al.* (2011) Network-based auto-probit modeling for protein function prediction. *Biometrics*, **67**, 958–966.

Kelley, B. *et al.* (2003) Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *PNAS*, **100**, 11394–11399.

Kelley, B. *et al.* (2004) PathBLAST: a tool for alignment of protein interaction networks. *Nucleic Acids Res.*, **1**, W83–W88.

Khan, A. *et al.* (2013). NeMa: fast graph search with label similarity. In Proceedings of the 39th International Conference on Very Large Data Bases, *PVLDB'13*. VLDB Endowment, pp. 181–192. Riva del Garda, Trento, Italy.

Lawrence, C. *et al.* (1993) Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *Science*, **262**, 208–214.

Lim, J. *et al.* (2006) A protein–protein interaction network for human inherited ataxias and disorders of purkinje cell degeneration. *Cell*, **125**, 801–814.

Ma, X. and Gao, L. (2012) Biological network analysis: insights into structure and functions. *Brief. Funct. Genomics*, **11**, 434–442.

Malod-Dognin, N. and Pržulj, N. (2015) L-GRAAL: Lagrangian graphlet-based network aligner. *Bioinformatics*, **31**, 2182–2189.

Micale, G. *et al.* (2014) GASOLINE: a greedy and stochastic algorithm for optimal local multiple alignment of interaction networks. *PLoS ONE*, **9**, e98750.

Panni, S. and Rombo, S.E. (2015) Searching for repetitions in biological networks: methods, resources and tools. *Brief. Bioinform.*, **16**, 118–136.

Pienta, R. *et al*. (2014). MAGE: matching approximate patterns in richly-attributed graphs. In *2014 IEEE International Conference on Big Data, Big Data 2014*, Washington, DC, October 27-30, pp. 585–590.

Pinter, R.Y. *et al*. (2005) Alignment of metabolic pathways. *Bioinformatics*, **21**, 3401–3408.

Ruepp, A. *et al*. (2010) CORUM: the comprehensive resource of mammalian protein complexes. *Nucleic Acids Res*., **38(Suppl. 1)**, D497–D501.

Sahraeian, S.M.E. and Yoon, B.J. (2012) RESQUE: Network reduction using semi-Markov random walk scores for efficient querying of biological networks. *Bioinformatics*, **28**, 2129–2136.

Shlomi, T. *et al*. (2006) QPath: a method for querying pathways in a protein-protein interaction network. *BMC Bioinformatics*, **10**, 199.

Sun, Z. *et al*. (2012) Efficient subgraph matching on billion node graphs. *Proc. VLDB Endow*., **5**, 788–799.

Szklarczyk, D. *et al*. (2011) The STRING database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nucleic Acids Res*., **39**, D561–D568.

Tian, Y. *et al*. (2007) SAGA: a subgraph matching tool for biological graphs. *Bioinformatics*, **15**, 232–239.

Tian, Y. *et al*. (2008) Periscope/gq: a graph querying toolkit. *Proc. VLDB Endow*., **1**, 1404–1407.

Vouzis, P.D. and Sahinidis, N.V. (2011) GPU-BLAST: using graphics processors to accelerate protein sequence alignment. *Bioinformatics*, **27**, 182–188.

Wang, H. *et al*. (2013) Function–function correlated multi-label protein function prediction over interaction networks. *J. Comput. Biol*., **20**, 322–343.

Yu, D. *et al*. (2013) Review of biological network data and its applications. *Genomics Inform*, **11**, 200–210.

Yuanyuan, T. and Patel, J. (2008). Tale: a tool for approximate large graph matching. In *Data IEEE 24th International Conference on Engineering, 2008, ICDE 2008*, pp. 963–972. Cancun, Mexico.

Zhang, S. *et al*. (2009). Gaddi: distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pp. 192–203. ACM, New York, NY.

Zhao, K. and Chu, X. (2014) G-BLASTN: accelerating nucleotide alignment by graphics processors. *Bioinformatics*, **30**, 1384–1391.