The Index-Based Subgraph Matching Algorithm (ISMA): Fast Subgraph Enumeration in Large Networks Using Optimized Search Trees

Citation for published version:

Digital Object Identifier (DOI):
10.1371/journal.pone.0061183

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
PLoS One

Publisher Rights Statement:
Copyright: © 2013 Demeyer et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
The Index-Based Subgraph Matching Algorithm (ISMA): Fast Subgraph Enumeration in Large Networks Using Optimized Search Trees

Sofie Demeyer1*, Tom Michoel2,3, Jan Fostier1, Pieter Audenaert1, Mario Pickavet1, Piet Demeester1

1 Department of Information Technology, Ghent University, Ghent, Belgium, 2 School of Life Sciences - LifeNet, Freiburg Institute of Advanced Studies, Freiburg, Germany, 3 Division of Genetics and Genomics, The Roslin Institute - University of Edinburgh, Midlothian, Scotland, United Kingdom

Abstract

Subgraph matching algorithms are designed to find all instances of predefined subgraphs in a large graph or network and play an important role in the discovery and analysis of so-called network motifs, subgraph patterns which occur more often than expected by chance. We present the index-based subgraph matching algorithm (ISMA), a novel tree-based algorithm. ISMA realizes a speedup compared to existing algorithms by carefully selecting the order in which the nodes of a query subgraph are investigated. In order to achieve this, we developed a number of data structures and maximally exploited symmetry characteristics of the subgraph. We compared ISMA to a naïve recursive tree-based algorithm and to a number of well-known subgraph matching algorithms. Our algorithm outperforms the other algorithms, especially on large networks and with large query subgraphs. An implementation of ISMA in Java is freely available at http://sourceforge.net/projects/isma.

Introduction

Over the last decade, network theory has come to play a central role in our understanding of complex systems in fields as diverse as molecular biology, sociology, economics, the internet, and others [1]. The central question is that all these fields is to understand behavior at the level of the whole system from the topology of interactions between its individual constituents. In this respect, the existence of network motifs, small subgraph patterns which occur more often in a network than expected by chance, has turned out to be one of the defining properties of real-world complex networks, in particular biological networks [2]. Network motifs act as the fundamental information processing units in cellular regulatory networks [3] and they form the building blocks of larger functional modules (also known as network communities) [4–6]. The discovery and analysis of network motifs crucially depends on the ability to enumerate all instances of a given query subgraph in a network or graph of interest, a classical problem in pattern recognition [7], that is known to be NP complete [8].

Subgraph matching algorithms are usually classified as either exact algorithms, which require a strict correspondence between the query graph (i.e. the subgraph) and any match in the target graph, or inexact algorithms, where some deformation of the query graph is allowed when searching for a match [7]. Here we are only concerned with exact algorithms. Some of the most well-known algorithms that realize this are the algorithm of Ullmann [9], the VF [10,11] and the VF2 algorithm [12,13]. The algorithm of Ullmann uses the adjacency matrix representation of the networks. A number of auxiliary matrices are defined to determine the set of subgraph isomorphisms iteratively. In the VF algorithms, on the other hand, the networks are represented by graphs. A state space representation is used in which each state depicts a (partial) mapping between nodes of both networks. The algorithm recursively builds a network of states by adding to the present states a pair of nodes that can be mapped on each other. After computing a set of candidate pairs, for each pair it is checked whether it meets the feasibility rules. Only then a new state is created. The difference between the VF and the VF2 algorithm is that the exploration of the search space has been improved in the VF2 algorithm to reduce memory requirements. This means that it is faster and can also be applied in larger graphs.

Most of the other exact algorithms typically find subgraph isomorphisms in a database of graphs. To realize the subgraph matching efficiently, a preprocessing step on this database is introduced. Messmer and Bunke [14] proposed a method consisting of building a decision tree from the database of graphs by a form of indexing. This structure can then be used to find all subgraph instances. This preprocessing step has been further optimized by Weber et al. [15]. Another algorithm, the GraphGrep algorithm [16], uses hash-based fingerprinting to index the database of graphs. GIndex [17] makes use of frequent substructures for its indexing. The GADDI algorithm [18] on the other hand deals with larger graphs and uses an indexing based on a neighborhood structure, similar to the TALE algorithm [19]. Another way to deal with exact subgraph matching is to reformulate it as a constraint satisfaction problem and solving it...
with constraint programming, which is a good approach if there are other constraints that need to be taken into account as well [20,21].

Motivated by problems in biology, where it is necessary to find subgraph instances in graphs with certain characteristics on the links, which define the type of interaction between cellular components (e.g. protein-protein, protein-DNA or protein phosphorylation, etc.) [6,22,23], we developed a novel exact subgraph matching algorithm, which uses a search tree to find all instances of a query subgraph in an edge-colored graph without using an additional, usually time consuming, preprocessing step. The algorithms that resemble our algorithm most are the algorithm of Ullmann [9], the VF [10,11] and the VF2 algorithm [12,13]. Note that our problem differs from for example the SAGA algorithm [24] in which the nodes instead of the edges contain certain characteristics.

At the heart of our algorithm are custom designed data structures (for both the network and the algorithm) which provide, at each step in the subgraph matching procedure, rapid indexing of the candidate nodes for inclusion in a subgraph instance. By carefully selecting the order in which the motif nodes (denoted by an index) are investigated, these sets of candidate nodes are kept as small as possible. This allows to cut unfavorable branches in the search tree as soon as possible and leads to a dramatic speedup compared to existing algorithms. In this paper, we present a formal description of the Index-based Subgraph Matching Algorithm (ISMA), the data structures and how symmetries in the query subgraph are dealt with. This paper is organized as follows. After giving a general overview of the problem, together with the definitions of the concepts that are used in this article, a formal description of the Index-Based Subgraph Matching Algorithm (ISMA), the data structures and how symmetries in the query subgraph are dealt with. This paper is organized as follows. After giving a general overview of the problem, together with the definitions of the concepts that are used in this article, a naive recursive algorithm is presented. The weaknesses of this algorithm are then identified and an improved recursive ISMA algorithm is proposed. As iterative algorithms may achieve a performance gain and require less stack space and function call overhead, an iterative version of the ISMA algorithm is presented, for which a number of custom data structures were designed. We also present comparisons to related subgraph matching algorithms using a variety of biological and non-biological networks.

**Methods**

**General description**

In biological networks, the same set of nodes (typically genes or proteins) can be connected in different ways, representing different physical interaction mechanisms, which may be directed or not [25]. In order to find matches for so-called composite motifs (subgraph patterns with more than one interaction type [22,23]), the ISMA algorithm is designed to find all occurrences of a given query subgraph in graphs $G_t$ with annotated edges. More precisely, $G_t = \{V,E\}$ with $V$ the set of vertices (or nodes) and $E$ the set of edges (or links), where each link is represented by a triplet $(u,v,T)$ with $u$ and $v$ the start and end node respectively, and $T$ the type of the link. Hence, in contrast to ordinary graphs, the links now also have a type, which identifies a number of characteristics of the link such as whether it is directed or undirected. It should be noted that parallel links are allowed in $G_t$ if and only if they are of a different type.

A motif or query subgraph $M$ is defined as follows. It is a small graph of $k$ nodes with no (anti-)parallel links. This means that a motif has at most $K = \frac{k(k-1)}{2}$ links. The assumptions of no (anti-)parallel links is not a strict condition and the research presented here can easily be extended to motifs with (anti-)parallel links. There are four possible configurations between two nodes $n_1$ and $n_2$: no link, a directed link from $n_1$ to $n_2$, a directed link from $n_2$ to $n_1$, or an undirected link between $n_1$ and $n_2$. The motif nodes ($n_i$) are ordered and can hence be referred to by a unique index $i$. In the remainder of this article, this index will be used to refer to the motif nodes themselves. Motifs are specified by a list of $K$ link

**Figure 1.** The motif adjacency Matrix. In this article, motifs are denoted by a motif specification which can be deduced from its adjacency matrix as indicated by the red arrow. doi:10.1371/journal.pone.0061183.g001

**Figure 2.** Examples of motifs and their specifications. Here nodes are denoted by their index. Look at for example the motif AAAAB000B0A00BAA, its motif specification can be deduced as follows: a directed link of type $A$ from node 1 to node 2, a directed link of type $A$ from node 1 to node 3, a directed link of type $A$ from node 2 to node 3, a directed link of type $B$ from node 1 to node 4, no link between node 2 and node 4, no link between node 3 and node 4, no link between node 1 and node 5, a directed link of type $A$ from node 4 to node 5, no link between node 1 and node 6, no link between node 2 and node 6, a directed link of type $B$ from node 3 to node 6, a directed link of type $A$ from node 4 to node 6, a directed link of type $A$ from node 5 to node 6. doi:10.1371/journal.pone.0061183.g002
types as follows:

\[ T(1,2), T(1,3), T(2,3), T(1,4), T(2,4), \ldots, T(k-1,k) \]

with \( T(i,j) \) the link type between the \( i \)-th and the \( j \)-th node of the motif. It is defined that if no link exists between two nodes in a motif the corresponding link type is set to null (or ‘0’). This motif specification can easily be deduced from the adjacency matrix of the graph as indicated in figure 1.

Link types may be specified by upper case characters (A, B, etc.). In the case the links are directed, the reverse of a link can be represented by the corresponding lower case character. A number of examples of motifs and their specification are given in figure 2.

The naive recursive subgraph matching algorithm (RSMA)

In this section, a naive recursive subgraph matching algorithm is described which is implemented in a motif clustering software tool (Cyclus 3D) [26]. We describe this algorithm in detail here, as it will form the basis for the ISMA algorithm. It is a depth first tree search procedure in which the motif nodes are investigated in the order in which they are listed in the specification. This means that the algorithm will first map a network node on the first motif node, then on the second motif node, and so on.

It should be noted that this algorithm resembles the VF2 algorithm. However, it is not completely similar. In the VF2 algorithms first a set of candidate pairs (i.e. a network node and a motif node on which this network node can be mapped) is calculated and subsequently this set is filtered according to the feasibility rules. One of these rules, for example, checks whether the links have the correct attributes (i.e. link types). By a careful design of the network data structure in the ISMA algorithm all candidate nodes are feasible nodes, which means that no additional checking operation is needed.

The pseudocode of the recursive subgraph matching algorithm (RSMA) is given below. The algorithm (i.e. the function findMotifs) takes 3 input parameters: a motif specification mspec, the motif instance instance that tracks which network nodes have been mapped onto the motif node, after which the findMotifs routine is called recursively. This way, all instances in \( G_t \) corresponding to the motif specification are enumerated.

```
1 findMotifs(mspec, instance, Gt){
2   if(instance is complete){
3     export(instance);
4     return;
5   }
6   mn = next();
7   set = determineSet(instance, mn, Gt);
8   forall(nodes nn in set){
9     instance.put(mn, nn);
10    findMotifs(mspec, instance, Gt);
11    previous();
12    instance.remove(nn);
13  }
14  }
15 }
16 i = 0;
17 next(){
18   return i++;
19  }
20 }
21 previous(){
22   i--;
23  }
24 }
25 determineSet(instance, mn, Gt){
26   if(index == 1){
27     return V;
28   }else{
29     sets = EMPTY;
30     forall(nodes ni in instance){
31       motiIndex = instance.getIndex(ni);
32       linkType = mspec.getLinkType(motiIndex, mn);
33       set = ni.getNeighborOfType(linkType);
34       sets.add(set);
35     }
36     return intersection(sets);
37  }
38 }
```

The Index-Based Subgraph Matching Algorithm (ISMA)

In this section, a naive recursive subgraph matching algorithm is described which is implemented in a motif clustering software tool (Cyclus 3D) [26]. We describe this algorithm in detail here, as it will form the basis for the ISMA algorithm. It is a depth first tree search procedure in which the motif nodes are investigated in the order in which they are listed in the specification. This means that the algorithm will first map a network node on the first motif node, then on the second motif node, and so on.

It should be noted that this algorithm resembles the VF2 algorithm. However, it is not completely similar. In the VF2 algorithms first a set of candidate pairs (i.e. a network node and a motif node on which this network node can be mapped) is calculated and subsequently this set is filtered according to the feasibility rules. One of these rules, for example, checks whether the links have the correct attributes (i.e. link types). By a careful design of the network data structure in the ISMA algorithm all candidate nodes are feasible nodes, which means that no additional checking operation is needed.

The pseudocode of the recursive subgraph matching algorithm (RSMA) is given below. The algorithm (i.e. the function findMotifs) takes 3 input parameters: a motif specification mspec, the motif instance instance that tracks which network nodes have been mapped onto the motif node, after which the findMotifs routine is called recursively. This way, all instances in \( G_t \) corresponding to the motif specification are enumerated.

```
1 findMotifs(mspec, instance, Gt){
2   if(instance is complete){
3     export(instance);
4     return;
5   }
6   mn = next();
7   set = determineSet(instance, mn, Gt);
8   forall(nodes nn in set){
9     instance.put(mn, nn);
10    findMotifs(mspec, instance, Gt);
11    previous();
12    instance.remove(nn);
13  }
14  }
15 }
16 i = 0;
17 next(){
18   return i++;
19  }
20 }
21 previous(){
22   i--;
23  }
24 }
25 determineSet(instance, mn, Gt){
26   if(index == 1){
27     return V;
28   }else{
29     sets = EMPTY;
30     forall(nodes ni in instance){
31       motiIndex = instance.getIndex(ni);
32       linkType = mspec.getLinkType(motiIndex, mn);
33       set = ni.getNeighborOfType(linkType);
34       sets.add(set);
35     }
36     return intersection(sets);
37  }
38 }
```

The initialization phase.

<table>
<thead>
<tr>
<th>link type</th>
<th>(nn)</th>
<th># nn</th>
<th>link type</th>
<th>(nn)</th>
<th># nn</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{1,2,5}</td>
<td>3</td>
<td>a</td>
<td>{1,5,6}</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>{1,2,4,5}</td>
<td>4</td>
<td>b</td>
<td>{2,3,4}</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>{2,3,4,5,6}</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For each link type the set of network nodes is depicted together with its cardinality.

doi:10.1371/journal.pone.0061183.t001

The RMSA algorithm has some efficiency issues. When the set of candidate network nodes is determined for the first motif node, the complete set of network nodes is returned (line 22). It is possible that a lot of these network nodes don’t even have the correct links (according to the motif specification) departing from them and thus are bad candidates to be mapped on the first motif.
Operation of the ISMA algorithm: an example

In this section, we will sketch the operation of the index-based subgraph matching algorithm by means of an example. In the three following sections, the algorithm is described in full detail.

Suppose we want to find all occurrences of the motif ABC (with link types A and B directed, and link type C undirected) in the network depicted in Figure 3. In the initialization phase, the best motif node to be investigated first is determined. As we want to narrow down the search tree, this should be the motif node with the least number of possible network nodes that can be mapped on it. These network nodes are the nodes that have the same types of outgoing links as the motif node. The set of possible network nodes for each motif node can then be determined by calculating the intersection of the sets of start network nodes of the corresponding link types. As calculating these intersections can be quite time-consuming, we opted to calculate the number of start nodes for each link type of the motif and select the motif node from which a link departs of the type with the lowest number of start nodes. This can be determined at initialization. Again, it will be determined which of the two remaining motif nodes (1 or 3) is the best option to be investigated next. To do this, we determine the cardinality of the sets of network nodes that are neighbors of network node 5 according to the correct link types in order to be mapped on the specific motif nodes. For motif node 1 (connected to motif node 2 with link type a) this set of network nodes is \{1,2,5\}, while for motif node 3 (connected to motif node 2 with link type C) this set is \{2,3\}. As there is only one network node that can be mapped on motif node 3, we will consider this motif node to be investigated next.

Network node 4 is mapped on motif node 3. Now, we need to determine which network nodes can be mapped on the last motif node, namely node 1. This is the intersection of the set of neighbors of network node 5 according to link type a (\{1,2\}) and the set of neighboring network nodes of node 4 according to link type b (\{1\}). This results in singleton \{1\}. We now have a complete instance that can be exported.

As there are no other network nodes that can be mapped on motif node 1 (all nodes of the set \{1\} have been mapped), and no other network nodes can be mapped on motif node 3 (all nodes of the set \{4\} have been mapped), we will map the next network node on motif node 2, namely node 6 (from the set \{5,6\} determined at initialization). Again, it will be determined which of the remaining motif nodes (1 or 3) will be investigated first. For motif node 1, the set of possible network nodes (neighbors of network node 6 according to link type a) is \{5\}. For motif node 3 (connected to motif node 2 with link type C) this set is \{2,3\}. Now motif node 1 is the best option to be investigated first as there is only one network node that can be mapped on it. Network node 5 is mapped on motif node 1. To determine which network nodes can be mapped on the last motif node 3, the intersection is calculated between the set of neighbors of network node 6 according to link type C (\{2,3\}) and the set of neighbors of network node 5 according to link type B (\{3\}). This results in the

The Index-Based Subgraph Matching Algorithm (ISMA)
singleton \{3\}. Network node 3 is mapped on motif node 3 and the complete instance can be exported.

There are no more network nodes that can be mapped on motif node 3 (all nodes of the set \{3\} have been mapped), and no more network nodes that can be mapped on motif node 1 (all nodes of the set \{5\} have been mapped). Moreover, we iterated over all network nodes that can be mapped on motif node 2 (all nodes of the set \{5, 6\} have been mapped). This means that the algorithm can terminate and has found all instances of the motif ABC in the network. Two motif instances have been found, one with the network nodes 1, 5 and 4 mapped on motif nodes 1, 2 and 3 respectively, and one with the network nodes 5, 6 and 3.

Similar to the naive recursive algorithm, this algorithm is also a depth-first search algorithm. When \(m\) motif nodes have network nodes mapped on them, first all possibilities for the remaining \(k - m\) nodes (assuming the motif has \(k\) nodes) will be checked, before mapping the next network node on the \(m\)-th motif node. As mentioned before, in this algorithm motif nodes are not always investigated in the same order. In the above example, in both instances motif node 2 was investigated first, but for the first instance motif node 3 was the next to be investigated, while for the second instance this was motif node 1. By carefully selecting the order in which the motif nodes are investigated, the sets of network nodes that can be mapped on the motif nodes are kept as small as possible, reducing the number of branches in the search tree.

The search trees of both the RSMA and the ISMA algorithm are depicted in figure 4. It can be seen that the ISMA algorithm indeed has a significantly smaller search tree, namely 6 nodes instead of 12.

The recursive index-based subgraph matching algorithm

Below, the pseudo code of the recursive index-based subgraph matching algorithm is presented. One can see that the main algorithm (i.e. the findMotifs-function) is similar to the one of the naive recursive algorithm. The determineSet-function only differs in the case in which no network nodes have been mapped on the motif nodes yet. Instead of returning the complete set of network nodes, now a set of good candidate network nodes for this motif node is returned. These are the networks nodes that have the same links (or more accurately link types) departing from them as the specific motif node.

The next-function returns the best motif node to be investigated next. This is the motif node that has the smallest set of candidate network nodes, as this leads to a smaller search tree. In the case the motif instance is empty (i.e. no network nodes have been mapped on the motif nodes), we will determine for each link type the number of networks links of this type. The link type with the lowest cardinality determines the best motif node. As stated in the previous section, when multiple link types have the same cardinality (i.e. the boolean variable multiple is true), there are two options: randomly selecting one of these link types or calculating the sets of possible network nodes that can be mapped on the motif nodes. The code depicts the latter one. For each motif node the set of candidate network nodes is determined. These are the network nodes with the same outgoing link types as the motif node. The motif node with the smallest set of candidate nodes is the best option to be investigated first. In the case where some of the motif nodes have network nodes mapped on them, we will determine for each of the unmapped motif nodes the number of neighbors of the mapped network nodes that can be mapped on this motif node. The minimum number then determines which motif node will be investigated next. By always selecting the motif node with the smallest set of candidate network nodes, the number of branches in the search tree is kept as small as possible, leading to faster calculations.

Recursive ISMA algorithm

\[
\begin{align*}
1 & \text{ findMotifs}(mspec, \text{instance, } G) \{
2 & \quad \text{if(instance is complete)} \{
3 & \quad \quad \text{export(instance);}
\end{align*}
\]
return;
}
nm = next(instance, mspec);
set = determineSet(instance, nm, Gt, mspec);
forall(nodes nn in set){
    instance.put(nn, nm);
    findMotifs(mspec, instance, Gt);
    instance.remove(nn);
}

} else if(
    if(instance is empty){
        return i;
    } else if(
        multiple = FALSE;
        linkTypes = mspec.getLinkTypes();
        forall(types t in linkTypes){
            if(#NBS = #Gt.getStartNodesOfType(t));
            if(#NBS < min){
                min = #NBS;
                i = t.getStartNode();
                multiple = FALSE;
            } else if(#NBS = = min){
                multiple = TRUE;
            }
        }
    } else if(multiple){
        forall(motif nodes mn){
            #NBS = #determineSet(instance, mn, Gt, mspec);
            if(#NBS < min){
                min = #NBS;
                i = mn;
            }
        }
    } else{
       forall(unmapped motif nodes mn){
            forall(mapped network nodes nn){
                #NBS = number of neighbors of nn that can be mapped on mn;
                if(#NBS < min){
                    min = #NBS;
                    i = mn;
                }
            }
        }
    }
return i;
}

determineSet(instance, mn, Gt, mspec){
    if(instance is empty){
        linkTypes = mspec.getLinkTypesFrom(mn);
        forall(types t in linkTypes){
            set = Gt.getStartNodesOfType(t);
            sets.add(set);
        }
        return intersection(sets);
    } else{
        sets = EMPTY;
        forall(nodes ni in instance){
            motifIndex = instance.getIndex(ni);
            linkType = mspec.getLinkType(motifIndex, mn);
            set = ni.getNeighborOfType(linkType);
            sets.add(set);
        }
        return intersection(sets);
    }
}

Data structures

As shown in the example and the recursive ISMA algorithm, the execution time of the recursive algorithm can be reduced by an intelligent choice of the order in which the nodes of a motif are investigated. This way, unfavorable branches of the search tree are pruned as soon as possible. Moreover, by an intelligent design of the network data structure an additional speedup can be realized for both the RSMA and the ISMA algorithm.

This section starts with an overview of the optimizations to the network data structure which enable fast retrieval of the network nodes adjacent to links of a certain type. Subsequently, a number of data structures (see figure 5) are presented in order to realize an iterative version of the ISMA algorithm (which is more efficient and requires less stack space). The checklist keeps track of the order in which the motif nodes are investigated. A motif iterator is a collection of iterators that iterate over the possible network nodes for each of the motif nodes. The priority queue map is used in the algorithm to determine which of the motif nodes is the most lucrative to be investigated next. Investigating a motif node here means determining a set of network nodes that can be mapped on it, and adding each of these nodes to the instance one after the other.

To better understand the algorithm specific data structures, figure 6 presents an example in which the possible content of the data structures is given for one instant during the execution of the algorithm.

Optimizations to the network data structure. The network data structure contains a network (i.e. graph). The main structure consists of a list of nodes. Each of these nodes has a list of neighbors, together with the links connecting them. The data structure has been optimized for two specific operations in the ISMA algorithm: the retrieval of the start nodes of all links of a specific type and the retrieval of all neighbors of a node that are connected to that node with a link of a specific type. This has been accomplished by 2 changes:

- In the network structure a map is added with the link type as key and the set of the start nodes of all links of this link type as value. As the number of link types in a network is relatively small, finding the start nodes of all links of a specific type now only requires a small amount of execution time.
- However, this structure requires some additional memory. In a map, for each link type a set of reference to nodes (i.e. start nodes of links of this type) is stored. Assuming that the network has \(|E|\) links and that there are \(|T|\) link types, this means that the additional memory needed is equal to \((|E|+|T|)\) references plus the memory overhead of one map and \(|T|\) sets.
- For every node a map is kept with the link types as keys and the sets of all neighbors according to this link types as values. This speeds up the operation of finding all neighbors that are connected with a link of a specific type.
- As in ‘non-optimized’ networks nodes also contain references to their neighbors, this structure only needs a small amount of additional memory, namely \(|T|\) references to the link types plus the memory overhead of one map and \(|T|\) sets.
Checklist. A checklist (CL) is a data structure that keeps track of the order in which elements (motif nodes) are chosen from a collection. It contains an ordered list of the chosen elements, together with the set of all elements that have not been chosen yet. It should be noted that in the recursive algorithm, this information is kept in the stack. The check list data structure is illustrated in figure 5a.

In the example (see figure 6b) first motif node 2 was removed from the rest set and added to the chosen list. Subsequently motif node 4 was removed from the rest set and added to the chosen list. Two nodes (1 and 3) remain in the rest set.

Following functions are defined on an checklist:

- numberChecked(): returns the number of chosen elements
- lastChecked(): returns the last element that has been chosen
- check(element): removes element from the rest-set and adds it to the list of chosen elements
- uncheck(): removes the last element from the chosen-list and adds it to the rest-set
- checked(): returns the list of checked (i.e. chosen) elements
- rest(): returns the rest-set

Motif iterator. The motif iterator (MI) contains an iterator for each of the motif nodes that iterates over the possible network nodes that can be mapped on this motif node. Additionally, in order to know the order in which the motif nodes have been investigated, it contains a pointer to a checklist. When a motif node has not been investigated yet, the corresponding iterator is investigated, it contains a pointer to a checklist. When a motif node finishes, it is removed from the rest set and added to the checklist. Two nodes (1 and 3) remain in the rest set.

Following functions are defined on a motif iterator:

- put(motifnode, iterator): adds the iterator to the corresponding motifnode
- hasNext(): returns a boolean value indicating if any of the iterators has a next element. It will first check the iterator of the checklist.lastChecked(). If it is empty, it will check the iterator of the previous motif node. And so on.

- next(): returns the next element of the index iterator. This is the next element of the current motif node in the index list

Priorityqueueemap. The priorityqueueemap (PQM) is an instrument to determine the best possible motif node to be investigated next. It contains a priority queue for each motif node. Moreover, in order to know which of the motif nodes have not been investigated yet, a pointer to the checklist object is kept. This data structure is illustrated in figure 5c. It is similar to the motif iterator, but the iterators are substituted by priority queues (PQ).

In order to keep the search tree as small as possible, we want to select the (uninvestigated) motif node with the smallest set of possible network nodes that can be mapped on it. This set is the intersection of all the neighbor sets (a neighbor set is the set of all the network nodes that are connected to a mapped network node according to a specific link type) of nodes that can be mapped on this motif node. Since calculating this intersection can be time-consuming and the maximum cardinality of this intersection is the cardinality of the smallest of these neighbor sets, in the priority queues we will keep track of how many neighbors each mapped network node has according to the types of each of its outgoing links (according to the motif specification). By only taking into account the (cardinality of the) sets of neighbors of the mapped network nodes and not the intersection of these sets for one motif node, we do not produce the optimal (i.e. the smallest) search tree, but it is a good compromise between efficiency (calculating the intersection is time-consuming) and optimality.

The objects that are stored in the priority queues were designed specifically for the ISMA algorithm. We opted to name them priority objects (PO) (see figure 5d). Priority objects consist of 4 fields: a network node, a motif start node, a motif end node and the number of neighbors of the network node according to the type of the link between the motif start node and end node. The network node is the node of the current instance that has been mapped on the motif start node. A priority object contains 2 motif nodes, a start and an end node, from which the link type can be deduced. It should be noted that for one priority queue all the ‘motif end’ field values are equal, and thus could be omitted. To determine the next best motif node, for each of the priority queues of the motif nodes that have not been chosen yet the minimum number of possible neighbors is retrieved. The overall minimum determines which motif node will be chosen next, as the maximum cardinality of the set of possible network nodes is minimal for this motif node.

The objects that are stored in the priority queues were designed specifically for the ISMA algorithm. We opted to name them priority objects (PO) (see figure 5d). Priority objects consist of 4 fields: a network node, a motif start node, a motif end node and the number of neighbors of the network node according to the type of the link between the motif start node and end node. The network node is the node of the current instance that has been mapped on the motif start node. A priority object contains 2 motif nodes, a start and an end node, from which the link type can be deduced. It should be noted that for one priority queue all the ‘motif end’ field values are equal, and thus could be omitted. To determine the next best motif node, for each of the priority queues of the motif nodes that have not been chosen yet the minimum number of possible neighbors is retrieved. The overall minimum determines which motif node will be chosen next, as the maximum cardinality of the set of possible network nodes is minimal for this motif node.

The following functions are defined on a priorityqueueemap:

- add(priorityqueueObject): adds the priorityqueueObject to the correct priority queue according to its motif end node
- poll(): removes and returns the overall best element (i.e. priority object) of the priority queues

The index-based subgraph matching algorithm (ISMA)

As mentioned previously, the recursive ISMA algorithm outperforms the naive recursive algorithm by always selecting the motif node with the smallest set of possible network nodes that
can be mapped on it. In this way, the number of branches in the
search tree is minimized.

Moreover, iterative algorithms can improve the performance
and consume less stack space and function call overhead. In the
previous section a number of data structures were presented in
order to realize an iterative version of the ISMA algorithm. In this
section the actual algorithm is discussed.

The pseudo code of the iterative ISMA algorithm is given
below.

**Index-based Subgraph Matching Algorithm (ISMA)**
```
1 findMotifs(mspec, Gt){
2     instance = EMPTY INSTANCE;
3     mn = determineFirstMotifNode();
4     CL.check(mn);
5     {startsetg = determineSet(instance, mn, Gt, mspec);
6         MI.put(mn, startset.iterator());
7     }
8     while(MI.hasNext()){
9         nn = MI.next();
10        backtrack();
11        if(instance is complete){
12            export(instance);
13            continue on line 8;
14        }
15       forall(i in CL.rest()){
16            PQM.add(new PO(nn, CL.lastChecked(), i, 
#NBS));
17        }
18        mn = PQM.poll().getEndNode();
19        CL.check(mn);
20        set = determineSet(instance, mn, Gt);
21        MI.put(mn, set.iterator());
22    }
23 }
```

In the initialization phase (line 2–6) we will stipulate which of
the motif nodes is best suited to be handled first (line 3). To
determine this, for each link type present in the motif the number
of occurrences in the network is counted. The start (motif) node of
the link of the type with the least instances in the network is chosen
as the first motif node. If multiple motif nodes have an outgoing
link of this type or multiple link types have the same number
of instances, there are two options. One could randomly select one of
these link types to determine the first motif node fast, but at the
risk of creating unnecessary branches in the search tree. The
other option is to calculate the actual sets of candidate network
nodes for these motif nodes by intersecting the sets for each of the
link types departing from this motif node. Then the motif node
with the smallest set of candidate network nodes is selected. This
comes down to calculating the following intersection value (IV) for
all motif nodes \( mn \) from which links of the specific types depart:

\[
IV(mn) = \#IS(mn) = \#\left( \bigcap_{j \in M, j \neq mn} \{ k((k,l), type(mn,j)) \in E \} \right)
\]

This is in fact the cardinality of the intersection (IS) of all sets of
network start nodes of the links of the types that depart from \( mn \).
For \( mn \) all types of the links departing from it are determined. For
each of these link types the set of starting nodes in the network is
collected and the cardinality of the intersection of all these sets is

\[
\text{Figure 7. Examples of reflection and cyclic rotation symme-
tries.} \quad \text{The motif on the left has a reflection symmetry between nodes 2 and 3. The motif on the right has a cyclic rotation symmetry between the three nodes.}
\]

doi:10.1371/journal.pone.0061183.g007

\[
\text{Figure 8. Reflection symmetry. Enumeration of all possibilities
to map 5 network nodes on 2 reflection symmetric motif nodes. The squares represent motif nodes, the circles represent network nodes.}
\]

\[
\text{Once a network node has been mapped on a motif node that is part of a reflection symmetry, it will never be mapped on one of the other}
\]

\[
\text{nodes of the symmetry.}
\]

doi:10.1371/journal.pone.0061183.g008

\[
\text{Figure 9. Cyclic rotation symmetry. Enumeration of all possibilities
to map 5 network nodes on 3 motif nodes that are part of a cyclic}
\]

\[
\text{rotation. The squares represent motif nodes, the circles represent}
\]

\[
\text{network nodes. Once a network node is mapped on the ‘first’ motif of the}
\]

\[
\text{symmetry, it will never be mapped on the other nodes of the}
\]

\[
\text{symmetry. For the other motif nodes of the symmetry, all possibilities}
\]

\[
\text{still need to be explored.}
\]

doi:10.1371/journal.pone.0061183.g009
calculated. The minimal value (for the different motif nodes) of this parameter then determines which motif node will be handled first. The determination of the first motif node is heuristic in the sense that we want to find a good first motif node as soon as possible, while we cannot guarantee that the search tree we build is indeed the smallest one possible. Once it is known which motif node is the first to be investigated, a start set of network nodes is calculated (line 5). It consists of all network nodes that can be mapped onto this motif node. It should be noted that the determineSet-function is identical to the one used in the recursive ISMA algorithm. The chosen motif node is checked in the checklist (line 4), and an iterator over the determined start set is added to the motif iterator (line 6).

The main part of the algorithm executes the following as long as there are network nodes in the motif iterator (line 9). It retrieves the next network node from the motif iterator, and adds it to the instance on the position of the last checked motif node (line 9 and 11). If the instance is complete (i.e. all motif nodes have network nodes mapped on them), it is exported and the algorithm continues by retrieving the following network node from the motif iterator (line 12–15). If this is not the case, for all the motif nodes that have not been handled yet it is determined how many neighbors of this network node can be mapped on them and the results are added, in the form of a priority object, to the priorityqueuemap (line 16–18). Subsequently, the next best motif node is determined by retrieving the best priority object from the priorityqueuemap (line 19). This motif node is checked in the checklist, and an iterator over a set of network nodes is added to the motif iterator (line 20–22). This set is the result of the function determineSet that is identical to the one used in the recursive ISMA algorithm. For a complete description of this function we would like to refer to the section of the recursive algorithm.

The algorithm terminates when there are no more network nodes left in the motif iterator. This means that there are no more network nodes that are good candidates to be mapped on the motif nodes. All motif instances have thus been found.

It should be noted that, when the next motif node is retrieved from the motif iterator (line 9), the data structures are updated to allow backtracking. This is indicated by the backtrack-procedure (line 10). There are three possible situations. When a new iterator was added in the previous iteration, no updates are needed, and the algorithm can continue. If the motif iterator retrieves a motif node from the same iterator as in the previous iteration, both the motif instance and the priorityqueuemap need to be updated. The network node that was mapped in the previous iteration needs to be removed from the instance, so that a new network node can be mapped. Moreover, all priority objects that are associated with this previous network node need to be removed from the priorityqueuemap. If, on the other hand, the network node that is returned comes from an iterator that is associated with a motif node that was handled previously, all data structures are updated. In the checklist the motif nodes that have been investigated completely (i.e. the algorithm has iterated through all network nodes that can be mapped on them in the current situation) need to be unchecked. In the priorityqueuemap all priority objects that have these motif nodes as start node are removed. Moreover, all network nodes that are mapped on motif nodes of the rest-set of the checklist are removed from the instance, as well as the network node that is mapped on the current motif node (i.e. lastChecked()). The priority object associated with this last network node are also removed from the priorityqueuemap.

Dealing with symmetry

In this section it will be explained how the ISMA algorithm can be further optimized when dealing with symmetric motifs. A motif is called to be symmetric if it is identical to a motif from which the nodes are permuted in a certain way. The most common symmetries in motifs are reflections, rotations, translations and combinations of these three. By making use of the symmetry characteristics of a motif, the search tree of the (iterative) ISMA algorithm can be pruned further. Once a network node has been mapped on a motif node that takes part in a symmetric permutation, it should not be mapped again on the other nodes of this symmetry if this would lead to the same motif instance. In this paper, we will focus on two kind of symmetries, namely reflections (or mirror symmetry) and cyclic rotations, as these can easily be exploited to speed up the calculations. In the future, we plan to take into account all sorts of symmetries (like the algorithm of [27] does). At this time, for all other symmetries, duplicate instances will be eliminated once they have been found.

A motif contains a reflection symmetry if and only if two or more nodes can be swapped without changing the motif’s configuration. For example, the motif AAX in figure 7 has a reflection symmetry between node 2 and node 3. A cyclic rotation symmetry is a symmetry in which nodes can be moved in circles. An example of a motif with this type of symmetry is Aaa (figure 7) where the motif with nodes [1,2,3] is equal to the motif with nodes [2,3,1] and the one with nodes [3,1,2].

The idea behind ‘dealing with symmetry’ is that once we have mapped one network node on a motif node that is part of a symmetry, we do not want it mapped again on another motif node of the symmetry. Suppose that we have a reflection symmetry with s motif nodes and that there are t network nodes that can be mapped on them, then mapping the network nodes on these symmetric motif nodes comes down to choosing s distinct elements out of a set of t elements, not taking into account the order of the elements, which is in fact a combination of s elements out of a set of t elements. One way to enumerate all these combinations is by first summing up all sets with the first element, then all sets with the second element that have not been encountered yet, etc. This is illustrated in figure 8. From this, it can be seen that, once one network node is mapped on a symmetric motif node (in the figure the first motif node), it will never be mapped again on one of the other symmetric motif nodes that are investigated thereafter.

For cyclic rotation symmetries this is slightly different. Here one motif node needs to be chosen as the ‘first’ node of the rotation and the idea is that once a network node is mapped on this ‘first’ node, it cannot be mapped on one of the other motif nodes in the symmetry, while a network node that is mapped on the ‘second’ (or higher) motif node still could be mapped on the other nodes of the symmetry. This is illustrated in figure 9 where all possibilities are given to map 5 network nodes on 3 motif nodes that form a cyclic rotation. Once a network node has been mapped on the ‘first’ motif node of the rotation, it will not be mapped again on one of the other nodes in the rotation, while network nodes that are mapped on the ‘second’ motif node, still can be mapped on the ‘third’ motif node later on.

In order to realize a speedup by making use of these symmetry characteristics, some changes were made to the motif data structure and the algorithm. Additional information is stored in the motif, namely for each motif node a list is kept of the motif nodes with which it is symmetric. In the case of reflection symmetry, for each node of the symmetry this list contains all other nodes of the symmetry. In the case of cyclic rotations for the motif node, that has been chosen as the ‘first’ node, this list contains all other symmetric nodes, while the lists of the other motif nodes only
contain one element, namely the ‘first’ motif node. Next to these changes in the motif definition, a novel data structure, called symmetry sets, was developed. For each symmetric motif node it contains a set of all possible network nodes that have not been mapped on it yet. These sets are used to help determining the new set of candidate network nodes (line 21 of the ISMA algorithm). If the motif node is symmetric, the (symmetry) sets of all nodes that are symmetric to it (according to the symmetry structure that was added in the motif definition), are added to the set of sets in the determineSet procedure. As these sets only contain the network nodes that have not been mapped yet, they make sure that all network nodes that have been mapped on symmetric motif nodes are eliminated when calculating the intersection.

In order to deal with symmetry (reflection or cyclic rotation) and further speed up the calculations, the ISMA algorithm is adapted in three ways:

- Every time it is determined which motif node will be investigated next (line 19), it is checked whether this node is part of a symmetry with motif nodes that have not been investigated yet. If this is the case, the set of network nodes that can be mapped on this motif node is added to the symmetry sets data structure. This is realized by adding following code after line 21.

  21a if(mn.isSymmetric) {
  21b symmetrySets.put(mn, set);
  21c }

- Every time a network node is retrieved from the motif iterator (line 9), the symmetry sets are updated. This means that, if this network node is mapped on a symmetric motif node, this network node is removed from the set associated with this motif mode. In this way it will not be mapped on the other motif nodes of the symmetry. This is realized by adding following code after line 11.

  11a if(CL.lastChecked.isSymmetric) {
  11b symmetrySets.remove(CL.lastChecked, nn);
  11c }

- When the determineSet procedure is called, it is checked whether the motif mode (i.e. index) is symmetric to nodes that have been investigated before. If this is the case, the corresponding (symmetry) sets are retrieved from the symmetry sets data structure and added to the set of sets, from which the intersection is calculated (line 55 of the recursive ISMA algorithm). This is realized by adding following code after line 54 in the determineSet procedure.

  54a for all (mn in CL.checked) {
  54b if(mn.symmetricTo(motifnode)) {

Table 2. Network configurations of biological networks.

<table>
<thead>
<tr>
<th>PGS network</th>
<th>XYZ network</th>
</tr>
</thead>
<tbody>
<tr>
<td># nodes</td>
<td>1 255</td>
</tr>
<tr>
<td># S links</td>
<td>667</td>
</tr>
<tr>
<td># G links</td>
<td>8 102</td>
</tr>
<tr>
<td># P links</td>
<td>3 688</td>
</tr>
</tbody>
</table>

The Index-Based Subgraph Matching Algorithm (ISMA)

As mentioned before, the above adaptations narrow down the search tree of the ISMA algorithm by making use of the characteristics of the reflection and the cyclic rotation symmetries. Besides this, for all other types of symmetry, every time a new instance has been found, it will be checked whether it is symmetric to a previously found instance. If this is the case, the instance will not be exported (line 13). In order to check this, all motifs that are symmetric to this motif (except for reflections and cyclic rotations) need to be identified. For a motif with \( k \) nodes, all symmetric motifs can be found by enumerating all permutations of the \( k \) motif nodes, connecting them according to the motif specification, and comparing this newly formed motif to the original one. From this list of symmetric motifs, the reflection and cyclic rotation symmetries are eliminated, as they are already accounted for in the algorithm. By mapping the motif nodes of an instance to all symmetric motifs, it can easily be checked whether the instance is symmetric to a previous one.

In order to enumerate all permutations, a 17th century algorithm, called ‘plain changes’ by the English bell ringers, was used. In computer science, it is known as the Steinhaus-Johnson-Trotter algorithm [28–30], and it has been improved by Even [31]. It is a powerful algorithm that generates an ordering of all permutations and is able to find all \( n! \) permutations of \( n \) elements by swapping two adjacent elements \( n! = 1 \) times. Due to the small differences between two consecutive permutations, this algorithm can be implemented in a constant time per permutation.

Results and Discussion

This section starts with a description of the software that incorporates the index-based subgraph matching algorithm. Subsequently, a number of results are presented that indicate the strength of the algorithm in comparison with other subgraph matching algorithms.

Software

A software implementation of the iterative ISMA algorithm is freely available at https://sourceforge.net/projects/isma/. It can be used from a command line interface as follows:

```
java -jar "<directory>/ISMA.jar" -folder "<folder of input files>" -linkfiles "<list of typename u/d filename> separated by spaces>" -motif "<motif>" -output "<reference to output file>"
```

The first two words indicate we want to execute a .jar-file. Subsequently, it is indicated where the .jar-file in question is situated. The program takes four arguments: folder, linkfiles, motif and output. The folder argument contains the directory where all input files are situated. It avoids retyping it for every inputfile. The linkfiles are the files that compose the network. Each linkfile contains all links for one specific type. It is denoted in the command by three arguments: the name of the link type (mostly an upper case character), a character indicating whether the links are directed (d) or undirected (u) and the name of the file. The different linkfiles are separated by spaces. The next argument is the motif. This is the motif specification as defined previously in this article. The last argument determines where the output should be stored.
The input files contain all links of one type. These links are represented by the name of the start node and the name of the end node separated by a tab. Every line contains one link. Example input files can be found online. The output file has one line for every motif that has been exported. A motif is represented as follows: Motif [motif specification]: [node 1, node 2, ...].

As the above notation is quite complicated, we will explain it in more detail by means of an example.

```
java -jar "ISMA/ISMA.jar" -folder "ISMA/input/"
-linkfiles "A d linksAtype.txt B d linksBtype.txt C u linksCtype.txt" -motif "ABC"
-output "ISMA/output/results.txt"
```

In this example ISMA.jar can be found in ISMA/. All input files are present in the folder ISMA/input/. The network contains links of three types: directed links of type A, directed links of type B and undirected links of type C. All these links can be found in the files linksAtype.txt, linksBtype.txt and linksCtype.txt respectively. The motif that is searched for is ABC, which is the motif we used in the example (see figure 3). The result of the ISMA algorithm (i.e. a list of all motifs found) is written to the file results.txt in the directory ISMA/output/.

In the ‘Files’ tab of this SourceForge project, all input (network) files that were used in the experiments are available. For a complete description of these networks, we would like to refer to the following section.

Results

To demonstrate the strength of the ISMA algorithm, we compared it to the naïve recursive subgraph matching algorithm (RSMA) as well as the algorithm of Ullmann [9], the VF algorithm [10,11] and the VF2 algorithm [12,13], which are state-of-the-art subgraph matching algorithms. We used two networks with multiple edge types as test networks. The first is an integrated network of physical (P, undirected), genetic (G, undirected) and signaling (S, directed) interactions between kinases and phosphatases in yeast [32,33], also used in [26] (see left panel of Table 2 for basic network characteristics). The second consists of protein-protein interactions in yeast (X, undirected, obtained from the BioGRID [34] database), protein-protein interactions in human (Y, undirected, obtained from the BioGRID [34] and STRING [35] databases), and orthology relations between human and yeast proteins (Z, bipartite, from the InParanoid database [36]) (see right panel of Table 2 for basic network characteristics). Experiments were carried out on a 64-bit machine with a processor of the type Intel(R) Core(TM) 2 Duo CPU P8600, 2.40 GHz and 4 GB of RAM. Both the Recursive Subgraph Matching algorithm (RSMA) and the Index-based Subgraph Matching Algorithm (ISMA) were implemented in Java (version 1.6.0_18). The Ullmann, VF and VF2 algorithms are all present in the VFlibrary, a (sub)graph matching library implemented in C. These algorithms are known for finding subgraph isomorphisms in Attributed Relational Graphs (ARGs) fast, which is exactly what we are dealing with here.

We first searched for a number of three-node motifs in the PGS-network, that are useful in the clustering algorithm of [26], and monitored the execution time for each of the algorithms. In table 3 an overview is given of these execution times (in milliseconds). Here, it can be seen that there is a major difference between the algorithms of the VFlibrary and the (naïve) RSMA and (iterative) ISMA algorithm. Despite the fact that they are implemented in C, which is a language with less execution overhead, the VFlibrary algorithms are remarkably slower. One of the decisive reasons for this discrepancy is the way how the network is stored in memory. For the ISMA and RSMA algorithm, the network structure has been optimized for fast retrieval of all start nodes of a certain link type (see Data Structures section). Looking at the RSMA and the ISMA algorithm separately (figure 10), we see that the ISMA algorithm indeed has lower execution times than the RSMA algorithm.
Table 3. The execution times (in milliseconds) for the different subgraph matching algorithms.

<table>
<thead>
<tr>
<th>motif</th>
<th># motifs</th>
<th>Ullmann</th>
<th>VF</th>
<th>VF2</th>
<th>RSMA</th>
<th>ISMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGG</td>
<td>9008</td>
<td>3653.5</td>
<td>4825.1</td>
<td>3421.1</td>
<td>181.0</td>
<td>76.4</td>
</tr>
<tr>
<td>SSS</td>
<td>81</td>
<td>851.0</td>
<td>145.6</td>
<td>80.4</td>
<td>4.6</td>
<td>0.9</td>
</tr>
<tr>
<td>GPS</td>
<td>47</td>
<td>1142.0</td>
<td>1061.9</td>
<td>769.4</td>
<td>12.8</td>
<td>2.3</td>
</tr>
<tr>
<td>SsS</td>
<td>3</td>
<td>815.2</td>
<td>140.3</td>
<td>76.7</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>SSG</td>
<td>103</td>
<td>861.4</td>
<td>155.7</td>
<td>87.0</td>
<td>4.9</td>
<td>3.8</td>
</tr>
<tr>
<td>sSg</td>
<td>31</td>
<td>838.4</td>
<td>145.5</td>
<td>83.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>GSS</td>
<td>312</td>
<td>1659.8</td>
<td>1167.6</td>
<td>866.7</td>
<td>106.0</td>
<td>2.6</td>
</tr>
<tr>
<td>GGP</td>
<td>418</td>
<td>1680.4</td>
<td>1201.5</td>
<td>898.7</td>
<td>108.2</td>
<td>7.3</td>
</tr>
<tr>
<td>ssG</td>
<td>112</td>
<td>868.2</td>
<td>163.6</td>
<td>94.5</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>8-clique</td>
<td>226</td>
<td>1465.250</td>
<td>4826 735</td>
<td>3140 065</td>
<td>2h</td>
<td>954</td>
</tr>
<tr>
<td>10-clique</td>
<td>1</td>
<td>4759.462</td>
<td>&gt;2h</td>
<td>&gt;2h</td>
<td>&gt;2h</td>
<td>1009</td>
</tr>
<tr>
<td>PGSPGS</td>
<td>0</td>
<td>1020.1</td>
<td>251.3</td>
<td>170.8</td>
<td>83.9</td>
<td>14.2</td>
</tr>
<tr>
<td>XZ00ZY</td>
<td>2558</td>
<td>535 657.0</td>
<td>111 610.0</td>
<td>42 514.0</td>
<td>153.7</td>
<td>93.9</td>
</tr>
<tr>
<td>XXXZ000Z0Y00ZYY</td>
<td>4745</td>
<td>1 828 683.0</td>
<td>825 828.0</td>
<td>182 734.0</td>
<td>2354.6</td>
<td>270.1</td>
</tr>
<tr>
<td>AAAZ000Z0B00ZBB</td>
<td>840</td>
<td>726 732.0</td>
<td>166 608.0</td>
<td>34 053.0</td>
<td>595.2</td>
<td>123.2</td>
</tr>
</tbody>
</table>

It should be noted that the experiments were interrupted after 2 hours.

doi:10.1371/journal.pone.0061183.t003

Subsequently, we looked for a number of larger motifs in this network. Table 3 shows the execution times of the algorithms when searching for all 8-cliques and 10-cliques (i.e. complete graphs of 8 and 10 nodes respectively). While for the ISMA algorithm, the execution times are around 1 second, they run up to more than two hours for the other algorithms. Here, the RSMA algorithm even performs worse that the algorithms of the VFlibrary. It should be noted that the difference between the ISMA algorithm and the other algorithms is most extreme for ‘complete’ motifs (i.e. cliques). However, for sparser motifs still significant speedups are realized, as will be shown in the experiments with the XYZ-network.

Moreover, we looked for a motif that is not present in the PGS-network, viz the 4-node motif PGSPGS. Results are shown in table 3. While the execution times are relatively low, the difference between the algorithms of the VFlibrary and the RSMA and ISMA algorithms is not as remarkable as in the previous cases.

Next we searched in the XYZ-network for so-called interologs, 4-node subgraphs (with motif specification XZ00ZY) consisting of conserved protein-protein interactions between orthologous protein pairs in yeast and human [37]. Generalizing this concept, we also searched for conserved triangles (6 nodes, motif specification XXXZ000Z0Y00ZYY). Despite of the fact that protein-protein interactions are undirected, in the experiments we also assumed the links to be directed. The directions were determined by the input files as the first and the second proteins were considered tails and heads respectively. In this directed network, we looked for the 6-node motif AAAZ000Z0B00ZBB. Here as well, we observe dramatic reduction in execution times for ISMA (and to a lesser extent RSMA) compared to the VF library algorithms, as these tend to be quite slow for larger motifs.

In conclusion, by an intelligent design of the network data structure a remarkable speedup is realized for the RSMA and ISMA algorithm in comparison to the VFlibrary algorithms. Moreover, this speedup is increased even more by carefully selecting the order in which the motif nodes are investigated (i.e. ISMA vs. RSMA).

To quantify the relative speedup realized by the ISMA algorithm, we defined the calculation time multiplicator (CTM) as

\[
\text{CTM}_{\text{ISMA algorithm}} = \frac{\text{execution time (algorithm)}}{\text{execution time (ISMA)}}
\]

For clarity reasons, in the remainder of this article we will use \(\text{CTM}_{\text{algorithm}}\) in stead of \(\text{CTM}_{\text{ISMA (algorithm)}}\).

These calculation time multipliers are given in table 4. It shows that the highest speedup factors are achieved for the motifs with an average number of occurrences. These figures show that the larger the motif, the larger the speedup that can be realized (XZ00ZY vs. XXXZ000Z0Y00ZYY and that these speedups are higher when there are more occurrences in the network (XXXZ000Z0Y00ZYY vs. AAAZ000Z0B00ZBB). In conclusion, 2 factors contribute in reducing the calculation time: the network data structure that has been optimized for fast retrieval of all links of a certain type and the order in which the motif nodes are investigated in order to reduce the search tree. While the former one explains the difference between the VFlibrary algorithms and the RSMA and ISMA algorithm, the latter one causes the execution time of the ISMA algorithm to be smaller than that of the RSMA algorithm.

As explained in the Methods section, the ISMA algorithm achieves its speedup compared to the RSMA algorithm by reducing the size of the search tree. We counted the number of nodes in the search tree for both algorithms when searching for 3-node motifs in the PGS-network (see figure 11). On average, the search trees of the ISMA algorithm are around 100 times smaller than the corresponding search trees of the RSMA algorithm. It should be noted that the size of the search tree (and the execution time) is dependent on both the motif configuration and the network in which these motifs are searched for.

Although the development of the ISMA algorithm was motivated by the problem of identifying composite motifs in biological networks with multiple interaction types, or more generally Attributed Relational Graphs, it can of course be applied equally well to non-biological networks. We illustrate this by
searching for all 3-node cliques in a number of networks from the SNAP database ([http://snap.stanford.edu/data/index.html](http://snap.stanford.edu/data/index.html)), where we treated all networks as undirected. Assuming all link are of the type $X$, this means the motif $XXX$ is searched. Table 5 shows the configurations of the networks that were used in the experiments, together with the number of 3-node cliques that were found in these networks. In table 6 the CTM’s are given of the ISMA algorithm over the algorithms of the VFlibrary and over the recursive algorithm. Moreover, the search tree reduction factors (i.e. the number of nodes in the search tree of RSMA divided by the number of nodes in the search tree of ISMA) are depicted. This table shows that, similar to the experiments on biological networks, the execution times of the algorithm presented in this article are much lower that those of the algorithms of the VFlibrary. Again, this can be explained by the network data structure that allows fast retrieval of all links of a certain type. If we take into account the large numbers of $XXX$ motifs that are present in these networks, the CTM’s are relatively small in

<table>
<thead>
<tr>
<th>motif</th>
<th># motifs</th>
<th>CTM(Ullmann)</th>
<th>CTM(VF)</th>
<th>CTM(VF2)</th>
<th>CTM(RSMA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGG</td>
<td>9008</td>
<td>47.8</td>
<td>63.1</td>
<td>44.8</td>
<td>2.4</td>
</tr>
<tr>
<td>SSS</td>
<td>81</td>
<td>873.7</td>
<td>149.5</td>
<td>82.5</td>
<td>4.8</td>
</tr>
<tr>
<td>GPS</td>
<td>47</td>
<td>505.5</td>
<td>470.1</td>
<td>340.6</td>
<td>5.7</td>
</tr>
<tr>
<td>SsS</td>
<td>3</td>
<td>1000.2</td>
<td>172.1</td>
<td>94.1</td>
<td>1.0</td>
</tr>
<tr>
<td>SSG</td>
<td>103</td>
<td>225.0</td>
<td>40.7</td>
<td>22.7</td>
<td>1.3</td>
</tr>
<tr>
<td>SsG</td>
<td>31</td>
<td>831.7</td>
<td>144.3</td>
<td>82.3</td>
<td>1.0</td>
</tr>
<tr>
<td>GGS</td>
<td>312</td>
<td>630.6</td>
<td>443.6</td>
<td>329.3</td>
<td>40.3</td>
</tr>
<tr>
<td>GSP</td>
<td>418</td>
<td>231.2</td>
<td>165.3</td>
<td>123.6</td>
<td>14.9</td>
</tr>
<tr>
<td>ssG</td>
<td>112</td>
<td>457.4</td>
<td>86.2</td>
<td>49.8</td>
<td>1.0</td>
</tr>
<tr>
<td>8-clique</td>
<td>226</td>
<td>1 535.9</td>
<td>5 059.5</td>
<td>3 291.5</td>
<td>&gt; 7 547.2</td>
</tr>
<tr>
<td>10-clique</td>
<td>1</td>
<td>4 717.0</td>
<td>&gt; 7 135.8</td>
<td>&gt; 7 135.8</td>
<td>&gt; 7 135.8</td>
</tr>
<tr>
<td>PGSPGS</td>
<td>0</td>
<td>72.1</td>
<td>17.8</td>
<td>12.1</td>
<td>5.9</td>
</tr>
<tr>
<td>XZ00ZY</td>
<td>2558</td>
<td>5704.5</td>
<td>1188.6</td>
<td>452.8</td>
<td>1.7</td>
</tr>
<tr>
<td>XXXZ00Z0OY00ZYY</td>
<td>4745</td>
<td>6771.3</td>
<td>3057.9</td>
<td>676.6</td>
<td>8.7</td>
</tr>
<tr>
<td>AAAZ00Z0O0ZBB</td>
<td>840</td>
<td>5900.3</td>
<td>1352.7</td>
<td>276.5</td>
<td>4.8</td>
</tr>
</tbody>
</table>

![Figure 11. Size search tree.](http://www.plosone.org/figure11.jpg) Comparision of the number of nodes in the search tree of the RSMA and the ISMA algorithm for finding 3-node motifs in the PGS network. The search tree reduction factor is defined as the size of the search tree of RSMA divided by the size of the search tree of ISMA. doi:10.1371/journal.pone.0061183.g011
motif node, all neighbors of the mapped network nodes need to be determining the set of network nodes that can be mapped on biological networks. The reason for this is that, when comparison to those in biological networks. This is also confirmed.

select the best motif node to be investigated next.

mapped on a motif node, and a priorityqueuemap in order to track of the order in which the motif nodes are investigated, a number of data structures were developed: a checklist that keeps a reduced search tree.

node with the smallest corresponding set of network nodes leads to remarkable speedup can be realized. In each iteration of the algorithm, sets of network nodes are determined that can be mapped on the remaining motif nodes. Always selecting the motif node with the smallest corresponding set of network nodes leads to less branches closer to the root of the search tree and consequently a reduced search tree.

In order to realize an iterative version of this algorithm, a number of data structures were developed: a checklist that keeps track of the order in which the motif nodes are investigated, a motif iterator to iterate over all the network nodes that can be mapped on a motif node, and a priorityqueue map in order to select the best motif node to be investigated next.

**Conclusion**

Motivated by problems in the analysis of biological networks composed of multiple directed and undirected interaction types, we have developed a novel exact subgraph matching algorithm that is optimized for graphs with specific link characteristics. By carefully selecting the order in which the nodes of a network motif are investigated and by designing appropriate data structures, a remarkable speedup can be realized. In each iteration of the algorithm, sets of network nodes are determined that can be mapped on the remaining motif nodes. Always selecting the motif node with the smallest corresponding set of network nodes leads to less branches closer to the root of the search tree and consequently a reduced search tree.

In order to realize an iterative version of this algorithm, a number of data structures were developed: a checklist that keeps track of the order in which the motif nodes are investigated, a motif iterator to iterate over all the network nodes that can be mapped on a motif node, and a priorityqueue map in order to select the best motif node to be investigated next.

**Author Contributions**

Conceived and designed the experiments: SD TM JF PA. Performed the experiments: SD. Analyzed the data: SD JF PA MP PD. Wrote the paper: SD TM JF MP PD.

**References**


**Table 5.** Network configurations of the non-biological networks, together with the number of 3-node cliques present in these networks.

<table>
<thead>
<tr>
<th>network</th>
<th># nodes</th>
<th># links</th>
<th># motifs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wiki-Vote</td>
<td>7115</td>
<td>103689</td>
<td>608389</td>
</tr>
<tr>
<td>p2p-Gnutella08</td>
<td>6301</td>
<td>20777</td>
<td>2383</td>
</tr>
<tr>
<td>p2p-Gnutella30</td>
<td>36682</td>
<td>88328</td>
<td>1590</td>
</tr>
<tr>
<td>CA-CondMat</td>
<td>23133</td>
<td>186936</td>
<td>173361</td>
</tr>
<tr>
<td>CA-HepTh</td>
<td>9877</td>
<td>51971</td>
<td>28339</td>
</tr>
</tbody>
</table>

**Table 6.** The calculation time multiplicators (CTM) of the ISMA algorithm compared to other algorithms for a number of non-biological networks.

<table>
<thead>
<tr>
<th>network</th>
<th>CTM(Ullmann)</th>
<th>CTM(VF)</th>
<th>CTM(VF2)</th>
<th>CTM(RSMA)</th>
<th>STRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wiki-Vote</td>
<td>126.8</td>
<td>211.9</td>
<td>142.8</td>
<td>11.1</td>
<td>41.2</td>
</tr>
<tr>
<td>p2p-Gnutella08</td>
<td>131.9</td>
<td>27.3</td>
<td>12.9</td>
<td>1.6</td>
<td>26.5</td>
</tr>
<tr>
<td>p2p-Gnutella30</td>
<td>706.4</td>
<td>35.7</td>
<td>6.4</td>
<td>1.1</td>
<td>17.7</td>
</tr>
<tr>
<td>CA-CondMat</td>
<td>870.1</td>
<td>565.2</td>
<td>241.3</td>
<td>2.3</td>
<td>15.0</td>
</tr>
<tr>
<td>CA-HepTh</td>
<td>293.4</td>
<td>195.0</td>
<td>83.5</td>
<td>1.5</td>
<td>11.1</td>
</tr>
</tbody>
</table>

The last column shows the search tree reduction factors (STRF), i.e. the ratio of the number of nodes in the search tree of the RSMA algorithm to the search tree of the ISMA algorithm.

doi:10.1371/journal.pone.0061183.t005

doi:10.1371/journal.pone.0061183.t006

doi:10.1371/journal.pone.0061183.t007