Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 901321, 11 pages
http://dx.doi.org/10.1155/2015/901321

Research Article

Fast Parallel All-Subgraph Enumeration Using Multicore Machines

Saeed Shahrivari and Saeed Jalili

Computer Engineering Department, Tarbiat Modares University (TMU), Tehran 14115-111, Iran

Correspondence should be addressed to Saeed Jalili; sjalili@modares.ac.ir

Received 28 January 2014; Revised 21 November 2014; Accepted 21 November 2014

Academic Editor: Przemyslaw Kazienko

Enumerating all subgraphs of an input graph is an important task for analyzing complex networks. Valuable information can be extracted about the characteristics of the input graph using all-subgraph enumeration. Notwithstanding, the number of subgraphs grows exponentially with growth of the input graph or by increasing the size of the subgraphs to be enumerated. Hence, all-subgraph enumeration is very time consuming when the size of the subgraphs or the input graph is big. We propose a parallel solution named *Subenum* which in contrast to available solutions can perform much faster. Subenum enumerates subgraphs using edges instead of vertices, and this approach leads to a parallel and load-balanced enumeration algorithm that can have efficient execution on current multicore and multiprocessor machines. Also, Subenum uses a fast heuristic which can effectively accelerate non-isomorphism subgraph enumeration. Subenum can efficiently use external memory, and unlike other subgraph enumeration methods, it is not associated with the main memory limits of the used machine. Hence, Subenum can handle large input graphs and subgraph sizes that other solutions cannot handle. Several experiments are done using real-world input graphs. Compared to the available solutions, Subenum can enumerate subgraphs several orders of magnitude faster and the experimental results show that the performance of Subenum scales almost linearly by using additional processor cores.

1. Introduction

Enumerating subgraphs of a given size has been shown to be a very useful task in the area of complex network analysis. Subgraphs can be used to identify building blocks and functional and nonfunctional characteristics in social, biological, chemical, and technological graphs [1]. An interesting application is subgraph mining which can be used to extract functional properties. A good example is finding *network motifs*, which are defined as connected subgraphs that occur significantly more frequently than expected [2]. One of the best known approaches for finding network motifs is to enumerate all subgraphs and then extract significant motifs after omitting frequent subgraphs that occur in random networks [3]. There are also many other applications in areas like data mining, statistics, systems biology, chemoinformatics, social networks, telecommunications, and web mining.

Although subgraph enumeration is a useful task, it is a computational challenging problem [4]. Enumeration can be classified into two distinct problems: enumerating all labeled subgraphs and enumerating nonisomorphic subgraphs, that is, subgraphs that have identical structure but different vertex labels. In the first problem, all of the subgraphs of a given size should be enumerated. On the other hand, in the second problem which is much more important, all of the nonisomorphic subgraphs of a given size must be enumerated. Both problems are very time consuming because the number of both labeled and nonisomorphic subgraphs increases exponentially by giving a bigger subgraph size or a larger input graph for subgraph enumeration.

As the size of the input graph increases, the number of subgraphs of size k increases exponentially (in the worst case $C(n,k)$ for a complete graph) [5]. The number of nonisomorphic subgraphs, which can be calculated using the Polya enumeration theorem [6], also increases exponentially as k increases. Therefore, by increasing the subgraphs size or the input graph's size, subgraph enumeration will take more time. When nonisomorphic subgraphs are enumerated, the problem becomes more complicated because an additional mechanism must be used to identify isomorphic subgraphs.

There is no known polynomial algorithm for subgraph isomorphism problem yet, and this overcomplicates the subgraph enumeration problem [7].

Due to the complex nature of subgraph enumeration problem, it is a very challenging and time-consuming problem. Available sequential algorithms tend to take a lot of time to do the job [3]. Hence, a good solution is to use parallel and distributed systems to accelerate subgraph enumeration [8]. Several other recent works targeting parallel subgraph enumeration have been proposed recently [8]. However, most of the related works are based on message passing interface (MPI) and hence are designed to work on cluster computing systems [8, 9]. In contrast, our goal is to provide a fast and easy to use tool for subgraph enumeration on commodity multicore and multiprocessor machines and to the best of our knowledge it has not yet been done. For this reason, we present a parallel solution, named *Subenum,* which is designed for faster and more scalable subgraph enumeration on multicore and multiprocessor machines. Subenum provides fast and efficient methods for counting and dumping both all and just nonisomorphic subgraphs.

Subenum's strength compared to other similar works can be classified into three categories. First, we have presented a new edge-based parallel subgraph enumeration algorithm named *PSE*, which is an improved version of the well-known sequential ESU algorithm. PSE provides a parallel and load-balanced approach for subgraph enumeration. The second strength is using a custom polynomial-time heuristic for detecting isomorphic subgraphs. The last strength is using a combination of external sorting and the nauty canonical labeling algorithm which enables Subenum to enumerate nonisomorphic subgraphs even when the number of subgraphs is so big that they cannot be stored in the main memory.

For evaluating the performance of Subenum we have performed several experiments on real-world graphs from different areas like social network, biological networks, software engineering, and electrical circuits. During the experiments, we compared Subenum's performance to state-of-the-art algorithms and implementations. Experimental results show that Subenum provides a parallel, load-balanced, and effective solution for all-subgraph enumeration problem. Compared to the fastest available tools for nonisomorphic subgraph enumeration, Subenum enumerates subgraphs several times faster and is able to reduce execution time from days to hours. In addition, Subenum is able to handle large graphs and also large subgraph sizes while other solutions fail to handle them.

2. Related Work

Related works for subgraph enumeration can be categorized into three main classes [8]: all-subgraph enumeration, single-subgraph enumeration, and subgraph-set enumeration. In all-subgraph enumeration (our problem), all of the subgraphs of size k of the original graph must be enumerated [1, 3, 4, 10]. Nevertheless, other conditions can also be defined for subgraphs for example, subgraphs of size k that have an Eulerian path. In single-subgraph enumeration, all of the isomorphic subgraphs of a predefined individual subgraph of size k must be enumerated [5]. Finally, in the subgraph-set enumeration, isomorphic subgraphs of a given set of subgraphs of size k must be enumerated [2]. As stated before, our solution is for the first kind of enumeration, that is, all-subgraph enumeration. Hence, we concentrate on related works that enumerate all subgraphs of size k of a given input graph. Interested reader can find deeper discussions in [8, 11–15].

The most notable efforts for all-subgraph enumeration problem are done in the network motif finding problem. As stated before, one of the best known exact approaches for finding network motifs is via all-subgraph enumeration and then counting nonisomorphic subgraphs [3]. The most notable works in this sector are mfinder [1], Kavosh [3], ESU aka FANMOD [4, 14], FPF [10], gtriesScanner [16], FaSe [17], NetMODE [18], and QuateXelero [19]. Note that gtriesScanner and FaSe use the ESU algorithm for subgraph enumeration, but in conjunction with ESU, they use the G-Tries data structure to accelerate subgraph isomorphism detection. Also note that FANMOD is limited to subgraphs smaller than 9 and NetMODE is limited to subgraphs smaller than 7. Compared to this group of related works, our solution has three strengths: parallel execution, using a heuristic (ordered labeling) for subgraph isomorphism, and external memory based isomorphic subgraphs counting.

Since subgraph enumeration is a time-consuming task, some recent works have used cluster computing to tackle the problem. Most of the available works for parallel all-subgraph enumeration are based on MPI. The most notable MPI-based solutions are discussed in [8, 20]. More works are done for parallel single-subgraph enumeration [2, 9, 21, 22]. Some recent works have used the MapReduce programming model [23] and Hadoop [24] for efficient single-subgraph enumeration on cloud and cluster computing systems. The most mentionable works are [25–29]. However, these works are also based on cluster and cloud computing systems. In contrast to available related work, Subenum presents a parallel solution that can boost the speed of all-subgraph enumeration problem using parallel processing capabilities of current commodity multicore and multiprocessor systems which are more accessible than expensive and complex solutions like cluster and parallel computing. There are some other similar but more complex problems like colored subgraph enumeration and motif finding [30, 31], but in order to keep this section short, we skip them. The interested reader can refer to [32] for more information.

3. Preliminaries

In mathematics, a graph is a collection of points that are connected by some links. The points of a graph are called vertices and the links are called edges. In this paper, if we use G to denote a graph, then $V(G)$ is used to present the vertices of G and $E(G)$ is used to present the edges of G. Vertices and edges of a graph can be assigned labels, weights, or colors. However, we assume graphs to be directed, simple, and unweighted. In other words, we assume that just the vertices take labels and the edges are directed and do not have weights and also there is at most one edge between two vertices.

For a vertex set $V' \subseteq V$ its open neighborhood $N(V')$ is the set of all vertices, $V - V'$, which are adjacent to at least one vertex of V'. For a vertex $v \in V - V'$ its exclusive neighborhood with respect to V' denoted by $N_{\text{excl}}(v, V')$ is the set of all vertices neighboring v that do not belong to $V' \cup N(V')$.

The graph H is a subgraph of G, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. An induced subgraph of G on the vertices set N denoted by $G[N]$ is a subgraph of G with N as the vertex set containing all edges between vertices of N that are in $E(G)$. When we say that we are enumerating subgraphs of size k of a graph like G we mean that we are enumerating induced subgraphs of G. Two subgraphs G_1 and G_2 are isomorphic if and only if there is a one to one correspondence between their vertices, and there is an edge between two vertices of G_1 if and only if there is an edge between the corresponding vertices in G_2. Actually, there is no polynomial time algorithm for graph isomorphism problem yet [7].

ESU Enumeration Algorithm. The most well-known algorithm for subgraph enumeration is the ESU algorithm [14]. ESU assumes that vertices are labeled by unique integer values. The basic idea of ESU algorithm is to start from each vertex v and enumerate all subgraphs of size k that contain v and vertices that have a bigger label than v, that is, the subgraphs that are v-rooted. ESU enumerates each subgraph just once. Details of the ESU algorithm are given in Algorithm 1.

4. Subenum: A Solution for All-Subgraph Enumeration

The easiest approach for parallel subgraph enumeration is to enumerate subgraphs rooted from each vertex in parallel using the ESU algorithm. However, this approach results unbalanced parallel tasks. Usually, there is a great variance in the number of subgraphs rooted from each vertex because vertices with higher degrees tend to participate in more subgraphs. For example in one of our experiments, more than 20% of subgraphs were enumerated from an identical vertex. Hence, this naïve approach causes unbalanced parallel loads and sometime this unbalanced load can cause inefficient parallelism.

Our idea for parallel enumeration is to enumerate subgraphs containing each edge in parallel. Enumerating subgraphs using edges causes more fine-grained parallel tasks because each vertex will be decomposed into several edges. Hence, the whole process is more load-balanced than vertex-based enumeration. For this purpose we need an algorithm for enumerating all subgraphs of size k that contain a specific edge $e(v, w)$. For this purpose, we have designed *Edge-based Subgraph Enumeration* (*ESE*) algorithm. ESE itself is an extended version of the ESU algorithm. However, in contrast to ESU, ESE is nonrecursive. Details of ESE algorithm are given in Algorithm 2.

Having defined the edge-based enumeration algorithm, we can explain *Parallel Subgraph Enumeration* (*PSE*) algorithm which uses the ESE algorithm as a building block. The procedure of PSE is simple. First, we put all of the edges of the input graph into a shared queue (for the case of bidirectional edges we put just one of them). Then, we use p concurrent threads to pick edges from the shared queue and enumerate subgraphs of each edge using p instances of ESE algorithm in parallel. The shared queue of edges between threads causes a more load-balanced parallelism. A more formal description of PSE is given in Algorithm 3.

Algorithm 3 enumerates all subgraphs of size k. However, if we want to enumerate nonisomorphic subgraphs, we need a mechanism to detect isomorphic subgraphs. One of the most efficient methods for graph isomorphism detection is *graph canonization*. Graph canonization produces a *canonical label* for every graph. Canonical labeling is completely graph invariant. Graph G is isomorphic to H if and only if canonical label of G equals canonical label of H [33]. There are some practical algorithms for canonical labeling like *nauty* [34], *bliss* [35], and *traces* [36]. However, there is no known polynomial-time algorithm for canonical labeling [36].

Most of the competing solutions use nauty for canonical labeling. When they find a new subgraph, first they find its canonical labeling using the nauty algorithm. Then, the new canonical label is looked up against a set of visited canonical labels. If the new canonical label is present in the set, then this subgraph is omitted else and the new canonical label is added to the label set. Some solutions, like FaSe, use a more sophisticated solution like G-tries instead of a lookup table for storing subgraphs, but the whole process is the same. This approach has two shortcomings. First, for each found subgraph, we need to generate its canonical labeling which can take exponential time in the worst case [34]. The second problem is the obligation to keep all of the unique canonical labels that are seen before in the main memory. These two shortcomings lead to unnecessary usage of processor and memory resources. Specially, when the size of subgraphs is big, for example, when k is more than 8, it would be impractical to keep all canonical labels in the main memory of a commodity workstation because there are millions of canonical labels.

To overcome these shortcomings, we propose a two-phase subgraph isomorphism solution that works with external storage. In the first phase, we use a fast $O(v^2)$ heuristic called *ordered labeling* to eliminate a considerable portion of isomorphic subgraphs. Then in the second phase, we use the nauty algorithm to eliminate all remaining isomorphic subgraphs. Advantages of our two-phase solution are as follows: (i) faster execution time and (ii) the ability to handle situations where the number of nonisomorphism subgraphs exceeds the main memory limits. A schematic of our proposed two-phase subgraph isomorphism detection solution is given in Figure 1. A flowchart for the ordered labeling step (first step of the first phase) is also given in Figure 2.

As shown in Figure 2, we use an intermediate set residing in the main memory for early duplicate ordered label

```
Input: A graph $G$ and an integer $k$: $1 < k \leq |V(G)|$
Output: All subgraphs of size $k$
  (1) for each vertex $v \in V(G)$ do:
      (a) $V_{Extension} \leftarrow \{u \in N(\{v\}) : u > v\}$
      (b) ExtendSubGraph($\{v\}, V_{Extension}, v$)
ExtendSubGraph($V_{Subgraph}, V_{Extension}, v$)
  (1) if $|V_{Subgraph}| = k$ then output $G[V_{Subgraph}]$ and return
  (2) while $V_{Extension} \neq \emptyset$ do
      (a) remove a vertex $w$ from $V_{Extension}$
      (b) $V'_{Extension} \leftarrow V_{Extension} \cup \{u \in N_{excl}(w, V_{Subgraph}) : u > v\}$
      (c) ExtendSubGraph($V_{Subgraph} \cup \{w\}, V'_{Extension}, v$)
```

Algorithm 1: ESU subgraph enumeration algorithm.

```
Input: A graph $G$, and an integer $k$: $1 < k \leq |V(G)|$, and an edge $e(v, w)$
Output: All subgraphs of size $k$ that contain $e(v, w)$
  (1) let Stack be a stack of tuples
  (2) if $v > w$ then swap $v$ and $w$                    //to guarantee that $v$ is smaller than $w$
  (3) $V_{Extension} \leftarrow \{u \in N(\{v\}) : u > w\} \cup \{u \in N_{excl}(w, \{v\}) : u > v\} - \{v, w\}$
  (4) push new tuple($\{v, w\}, V_{Extension}, v$) into Stack
  (5) while Stack is not empty do:
      (a) top ← pop the tuple on top of the stack
      (b) if $|top[0]| = k$ then output $G[top[0]]$ and return    //top[0] is the first item of the tuple
      (c) while $top[1] \neq \emptyset$ do:                       //top[1] is the extension set
          (i) remove a vertex $x$ from $top[1]$
          (ii) $V'_{Extension} \leftarrow top[1] \cup \{u \in N_{excl}(x, V_{Subgraph}) : u > top[2]\}$    //top[2] is the root
          (iii) push new tuple($top[0] \cup \{x\}, V'_{Extension}, top[2]$) into Stack
```

Algorithm 2: ESE enumeration algorithm.

```
Input: A graph $G$, an integer $k$: $1 < k \leq |V(G)|$ as the size of subgraphs, and an integer $p$ as the number of concurrent threads
Output: All subgraphs of size $k$
  (1) let $Q$ be an empty list.
  (2) for each edge $e(v, w) \in E(G)$ do:
      (a) if $e(w, v)$ is not in $Q$ then insert $e(v, w)$into $Q$
  (3) spawn $p$ threads
  (4) for each thread do in parallel:
      (a) while $Q$ is not empty do:
          (i) pick an edge $e(v, w)$ from $Q$
          (ii) enumerate all subgraphs of size $k$ containing $e$ using ESE algorithm
  (5) wait until all threads are done
```

Algorithm 3: PSE subgraph enumeration algorithm.

detection and when the size of the set exceeds the memory limit, we spill ordered labels set to external memory, that is, a file on disk. To generate an ordered labeling for a subgraph, we reorder the adjacency matrix considering degree of each vertex. Then, we concatenate rows of the reordered adjacency matrix to generate the ordered label for that subgraph. Algorithm 4 gives a more formal explanation of ordered labeling algorithm. Actually, ordered labeling algorithm just changes the labels of vertices and the graph structure is not changed. Hence, ordered labeling algorithm preserves graph isomorphism class and canonical labeling. Figure 3 shows an example of ordered labeling.

According to Figure 1, after generating ordered labeling for subgraphs and dumping them to a file on external storage, we have a file containing pairs of ordered labels and their frequencies. Afterwards, we use the nauty algorithm to generate a canonical label of each ordered label. Having a file containing canonical labels and frequencies for each subgraph, first we sort the file by canonical labels using parallel external merge sort algorithm and then, we traverse

```
Input: A subgraph $G$ represented with its adjacency matrix $M$
Output: A binary string of length $|V(G)|^2$ as the ordered labeling for $G$
  (1) let $L$ be a list of vertices, and initially $L = \varnothing$
  (2) for each vertex $v \in V(G)$ do:
      (a) insert $v$ to $L$
  (3) sort $L$ by degree of each vertex
  (4) let $Lookup$ be a lookup table and $Lookup[x]$ as the value associated to $x$.
  (5) for each vertex $v \in L$ do:
      (a) set $Lookup[v]$ equal to rank of $v$ in $L$
  (6) let $N$ be a binary matrix of size $M$ filled with zeros
  (7) for each $M_{i,j}$ in $M$ do:   //$A_{i,j}$ denotes the element of matrix $A$ in row $i$ and column $j$
      (a) if $M_{i,j} = 1$ then set $N_{Lookup[i],Lookup[j]}$ to 1
  (8) return concatenation of rows of $N$
```

Algorithm 4: Ordered labeling algorithm.

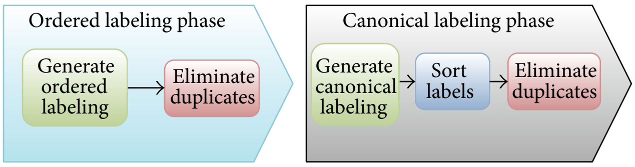


Figure 1: Two-phase isomorphism detection.

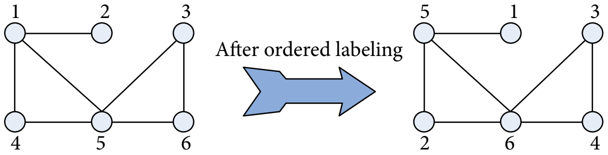


Figure 3: An example of ordered labeling.

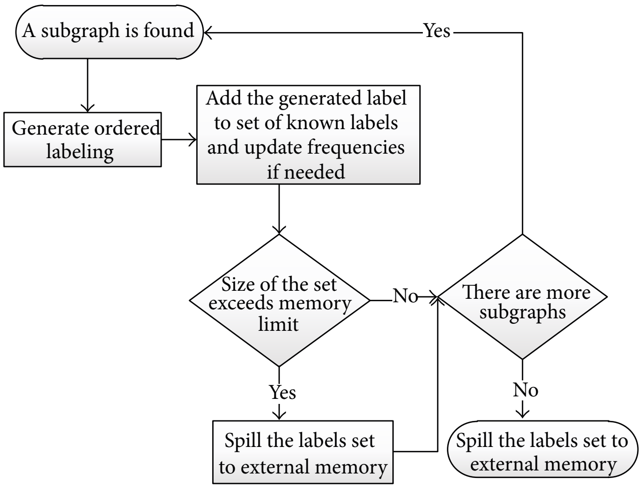

A subgraph is found
Yes
Generate ordered labeling
Add the generated label to set of known labels and update frequencies if needed
Size of the set exceeds memory limit
No
There are more subgraphs
Yes
No
Spill the labels set to external memory
Spill the labels set to external memory

Figure 2: Generating ordered labeling for subgraphs.

the sorted file and detect duplicate canonical labels and merge frequencies of duplicate labels. At last, we have unique canonical labels and their frequencies, that is, nonisomorphic subgraphs and their frequencies.

4.1. Complexity Analysis. During the ordered labeling heuristic we perform $O\ (k^2)$ lookups, assuming k as the size of the subgraph. Hence, if we use a data structure like *hash table* that provides $O(1)$ expected lookups, then the time complexity of the ordered labeling algorithm would be $O\ (k^2)$ which is far better than traditional canonical labeling algorithms that have time complexity of $O(k!)$.

For enumeration of subgraphs of size k, whether isomorphism detection is done or not, we need to enumerate all subgraphs of size k. In the worst case, for a complete input graph of size n, the number of induced subgraphs of size k is $C(n, k)$. On average, for a general input graph of size n, the number of subgraphs of size k should be exponential [4]. Hence, if we assume α as the number of subgraphs of size k, β as the number of isomorphic classes for subgraphs, and p as the number of processors, generating ordered labeling of all subgraphs needs $O(\alpha{\cdot}k^2/p)$ operations, eliminating duplicate ordered labels needs execution of standard parallel merge sort algorithm which has complexity of $O(\beta \cdot \log\beta/p)$, and applying nauty on unique ordered labels needs $O(\beta \cdot k!/p)$. Hence, the overall time complexity of Subenum is $O(\alpha{\cdot}k^2/p + \beta \cdot \log\beta/p + \beta.k!/p)$ which is far more better than other tools that use nauty directly which have complexity of $O(\alpha \cdot k!)$, because β is smaller than α [3, 4].

5. Experimental Results

In order to evaluate the performance and effectiveness of Subenum, we performed various experiments on different real-world graphs. For this purpose, we selected some well-known graphs from various fields like social networks, biology, communication, web graphs, and peer-to-peer networks. We have used eight different graphs: Elegans (neuronal network of Caenorhabditis elegans [37]), Jazz (network of jazz musicians [38]), School (face to face contact patterns in a primary school [39]), Vidal (proteome-scale map of human binary protein-protein interactions [40]), Gnutella (structure of Gnutella p2p network from August 31, 2002 [41]), Slash (slashdot social network from February 2009

Table 1: The properties of graphs used in experiments.

	Elegans	Jazz	School	Vidal	Gnutella	Slash	Tweet	Notre
Number of vertices	297	198	238	3,133	62,586	82,168	81,306	325,729
Number of edges	2,345	2,742	5,539	6,726	147,892	948,464	1,768,149	1,497,134
avg (deg.)	14.46	27.69	46.54	4.10	4.72	48.77	38.21	6.77
σ (deg.)	12.94	17.41	19.85	6.79	5.70	19.81	67.93	42.87

Table 2: The effectiveness of ordered labeling heuristic for subgraph isomorphism detection.

		Subgraph Size				
		3	4	5	6	7
Elegans	Number of subgraphs	47,322	1,394,259	43,256,069	1,309,307,357	37,818,052,163
	Number of ordered labels	20	552	24,745	961,476	31,104,089
	Number of nonisomorphic subgraphs	13	197	7,072	286,376	9,584,962
Jazz	Number of subgraphs	67,414	1,833,618	49,500,654	1,266,953,062	30,166,157,456
	Number of ordered labels	5	45	862	32,493	2,291,205
	Number of nonisomorphic subgraphs	4	24	267	5,647	237,008
School	Number of subgraphs	205,796	8,581,352	348,596,925	13,140,615,595	451,141,199,919
	Number of ordered labels	5	45	862	32,515	2,409,520
	Number of nonisomorphic subgraphs	4	24	267	5,647	237,319
Vidal	Number of subgraphs	86,715	2,161,170	62,607,036	1,901,854,904	58,919,388,890
	Number of ordered labels	42	766	18,201	411,148	8,637,628
	Number of nonisomorphic subgraphs	3	24	267	4,909	97,094
Gnutella	Number of subgraphs	1,564,126	23,646,400	449,446,489	9,806,726,769	234,415,296,091
	Number of ordered labels	7	70	933	12,787	170,594
	Number of nonisomorphic subgraphs	5	32	291	2,714	25,230

[42]), Tweet (social circles from Twitter [43]), and Notre (web graph of Notre Dame [44]). Main properties of these graphs are tabulated in Table 1. The first four graphs are small; for example, they have less than 10,000 vertices and edges. On the other hand, the latter four graphs are larger, for example, more than tens of thousands of vertices and up to one million edges.

The main goal of our experiments is to evaluate the overall speed of Subenum compared to available state-of-the-art algorithms. We divide the experiments into three sections. First, we evaluate effectiveness of ordered labeling heuristic for subgraph isomorphism detection. Then, we evaluate scalability and parallelism performance of Subenum on multicore and multiprocessor machines. Finally, we compare ultimate speed of Subenum to some of the available tools for subgraph enumeration (FANMOD, Kavosh, G-Tries, and FaSe) considering different input graphs and subgraph sizes.

We used two machines during our experiments. The first machine was a four-core Intel i7-2600 CPU having 8 GB of RAM and running Windows 7 64-bit edition. The second machine had two 6-core Intel Xeon-E5620 CPUs and 32 GB of RAM running Ubuntu 12.04. The 4-core i7 machine is mainly used for comparison with other tools, while the 12-core Xeon machine is mainly used for parallelism and scalability experiments. For better scalability, we used Azul Zing JVM on the Xeon machine. Subenum is coded in the Java programming language and its source code is available via GitHub at https://github.com/shahrivari/subenum.

5.1. Effectiveness of Ordered Labeling Heuristic. For testing the effectiveness of ordered labeling heuristic, we applied ordered labeling on some of the input graphs considering subgraphs of various sizes. The details about the effectiveness of ordered labeling heuristic are given in Table 2.

Three numbers are reported per subgraph size and input graph in Table 2: the number of subgraphs, the number of ordered labels, and the number of nonisomorphic subgraphs. As the numbers show, the numbers of ordered labels are much smaller than the numbers of subgraphs and close to the number of nonisomorphic subgraphs. This shows that Subenum calls the expensive nauty algorithm significantly fewer times (in orders of the number of ordered labels) while other solutions call nauty per each found subgraph. For example, considering the subgraphs of size 6 for Gnutella graph, Subenum calls nauty 12,787 times, while other tools call nauty more than 9 billion times.

5.2. Parallelism and Scalability. Subenum is inherently designed for running on multicore and multiprocessor machines. Hence, an important performance factor is the scalability of Subenum. That is to say, we want to know how much speedup is gained when additional processors are available to Subenum. For this purpose, we calculated the speedup of Subenum running with different counts of threads. We performed both all-subgraph enumeration and nonisomorphic subgraph enumeration. For calculating the speedup value, we divided the execution time of multithreaded

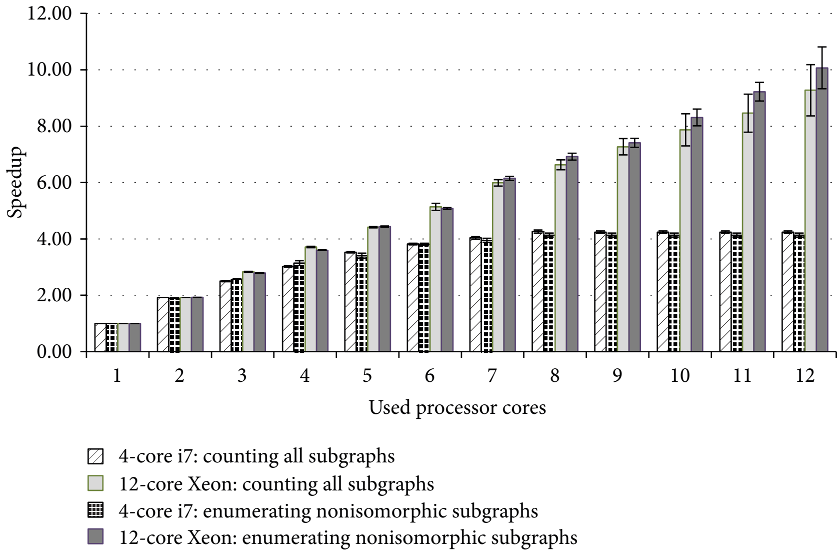


FIGURE 4: Overall speedup values considering different graphs.

version to the execution time of the single threaded version. Figure 4 shows an overall view of Subenum's scalability using multithreads. For this experiment, we executed Subenum using different number of threads on all of the input graphs and enumerated subgraphs of sizes 5 and 6 for the first four graphs and subgraphs of size 3 and size 4 for the latter four graphs. As Figure 4 shows, Subenum can reach a near-linear speedup when additional threads of execution are used until threads count reaches the number of available processor cores. Note that the small improvements after increasing the number of threads to values greater than number of cores are due to *HyperThreading* feature of Intel CPUs which allows each core to run two logical threads simultaneously. More details of speedup values for each input graph are given in Figure 5 which shows the increase of speedup values for each input graph by using more threads.

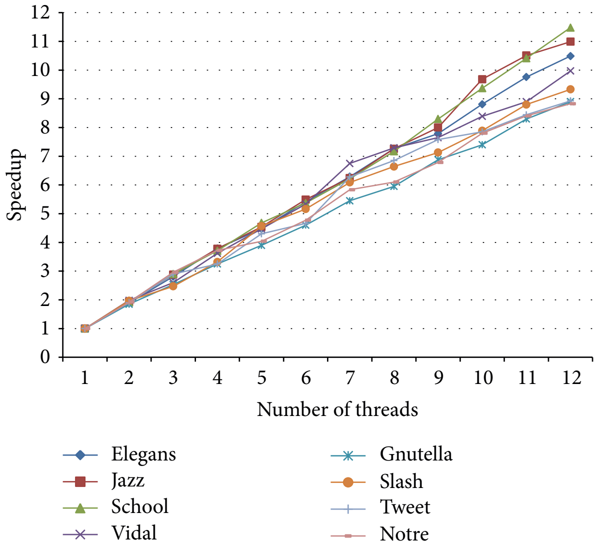


FIGURE 5: The speedup for each graph using different number of threads on 12-core Xeon machine.

5.3. Comparison to Other Solutions. The main goal of Subenum is to provide a faster solution for all-subgraph enumeration and nonisomorphic subgraph enumeration problems compared to available solutions. In this part of the paper, we compare the performance of Subenum to the best known available software for all-subgraph enumeration problem. The comparison is made to Kavosh, FANMOD, gtrieScanner, and FaSe. During the experiments of this section, we used the 4-core i7 machine.

All of the other solutions are sequential and comparing Subenum which is a parallel solution to sequential solutions is not very fair because Subenum can use all of the available cores, while others just use a single core. For this reason, in this experiment we used the 4-core i7 machine that has fewer cores compared to the 12-core Xeon machine. For better comparison, for every graph and subgraph size, we reported the performance of Subenum when using a single core, too. Note that Subenum is programmed in Java, while all of the other solutions are programmed in C/C++ which has proven to produce faster executable programs because of producing native machine code in contrast to Java that compiles to byte code which executes in the *Java Virtual Machine* (*JVM*).

For more clarity, we divided the input graphs into two groups. The first group consists of smaller graphs: Elegans, Jazz, School, and Vidal. The second group consists of larger graphs: Gnutella, Slash, Tweet, and Notre. Since the graphs of first group are smaller, larger subgraph sizes can be enumerated, while for the second group, enumerating large subgraphs like 8 can take months and even years.

Figure 6 gives an overall performance comparison of different solutions for the first group of graphs. For this experiment, we enumerated nonisomorphic subgraphs of sizes 5 and 6 for each input graph and reported the average normalized times. As Figure 6 shows, for all of the input graphs Subenum is the fastest solution. When Subenum is

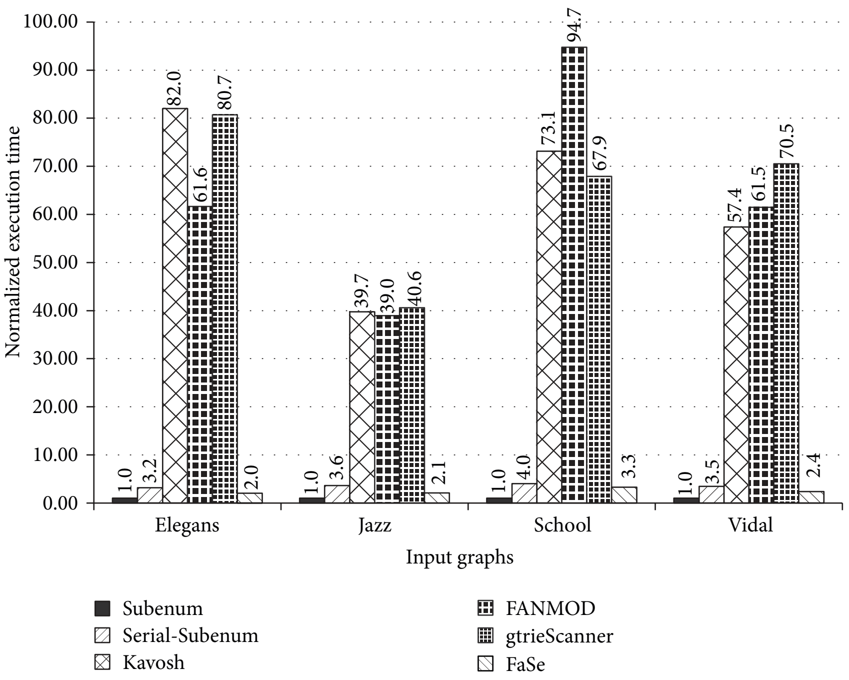


Figure 6: Average normalized execution times for smaller input graph (subgraphs of sizes 5 and 6).

executed in sequential mode, FaSe is faster, but it does not reach performance of Subenum when all of the 4 cores of the i7 CPU are used. We believe that the better performance of FaSe in the sequential mode is due to better performance of C++ compared to Java. Using Figure 6, we can also conclude that there is a great performance gap between Subenum and FaSe compared to other solutions. The main reason behind this issue is the better methods that Subenum and FaSe use to deal with subgraph isomorphism detection.

More details are given in Table 3. The execution times for each input graph and different subgraph sizes are given for each solution. For all input graphs and subgraph sizes, Subenum delivers the fastest execution time. An interesting point is the failure of other tools when larger subgraphs are enumerated. When large subgraphs (e.g., 8) are enumerated, other tools crash due to memory issues. These cases are denoted by *"Out of Mem."* in Table 3. The main reason behind this is the large count of nonisomorphic subgraphs. Since other tools keep all nonisomorphic subgraphs in the main memory, the main memory fills up and the tools crash. For the cases in which the execution of a solution took more than a week, we did not proceed and reported these cases by an estimation like ">1 *week*".

We performed the same experiments for the larger input graphs, too. Kavosh and gtrieScanner failed to load all of the graphs. Inspecting their code shows that they use a Boolean matrix for storing edges. Hence, they need $O(v^2)$ space, considering v as the number of vertices. FaSe can just handle the Gnutella and Slash graphs and fails due to insufficient memory for the rest of the input graphs. FANMOD performs better in handling large graphs. However, it is much slower than Subenum and FaSe. The details of execution times are given in Table 4.

6. Conclusion and Further Work

The number of both isomorphic and nonisomorphic subgraphs of a given graph grows exponentially when the size of input graph or the subgraphs to be enumerated is increased. Hence, the only available solution for accelerating all-subgraph enumeration problem is to use parallel and distributed systems. We presented a new parallel solution, named Subenum, for the all-subgraph enumeration problem on multicore and multiprocessor systems. In contrast to available parallel solutions that are designed for execution on cluster computing systems, Subenum is designed for faster execution on commodity multicore and multiprocessor desktop and workstation systems.

The novelties of Subenum can be summarized in three points. First, we designed a new parallel subgraph enumeration algorithm named PSE that provides a load-balanced and parallel procedure suited for subgraph enumeration on multicore and multiprocessor systems. Second, we offered a new, simple, and polynomial heuristic for subgraph isomorphism detection problem and we showed that it is very effective for pruning candidate subgraphs. And lastly, using a parallel external sorting solution we enabled Subenum to enumerate nonisomorphic subgraphs even when they are so large that they do not fit in the main memory. Our practical experiments on different real-world input graphs showed that Subenum is a scalable parallel solution and can easily outperform the fastest available tools like FANMOD and Kavosh on commodity multicore machines.

For further work, we plan to develop a distributed subgraph enumeration solution using the MapReduce programming model and the Hadoop framework. We have done most of the work and the preliminary results are encouraging.

Table 3: The execution times of different tools for small graphs in seconds.

	Tool	Subgraph size			
		5	6	7	8
Elegans	Subenum	**1.7**	**39**	**1,175**	**37,993**
	Serial Subenum	4.2	130	4,553	147,806
	FANMOD	55.2	2,453	85,465	Out of Mem.
	Kavosh	53.1	3,285	119,286	Out of Mem.
	gtrieScanner	52.7	3,233	Out of Mem.	Out of Mem.
	FaSe	2.2	81	Out of Mem.	Out of Mem.
Jazz	Subenum	**1.7**	**40**	**1,051**	**29,216**
	Serial Subenum	5.3	145	4,059	111,876
	FANMOD	46.3	1,578	49,660	Out of Mem.
	Kavosh	45.6	1,611	50,310	Out of Mem.
	gtrieScanner	43.7	1,649	Out of Mem.	Out of Mem.
	FaSe	2.7	84	2,663	Out of Mem.
School	Subenum	**8.8**	**285**	**17,062**	>1 month
	Serial Subenum	32.5	1,157	68,326	>1 month
	FANMOD	604	27,230	>1 week	Out of Mem.
	Kavosh	405	21,085	>1 week	Out of Mem.
	gtrieScanner	399	19,554	>1 week	Out of Mem.
	FaSe	20	954	42,156	Out of Mem.
Vidal	Subenum	**1.9**	**52**	**1,756**	**61,374**
	Serial Subenum	5.7	181	**6,851**	238,449
	FANMOD	76	3,239	147,369	>1 month
	Kavosh	76	3,016	143,159	>1 month
	gtrieScanner	82	3,720	197,432	>1 month
	FaSe	3.2	124	3,780	Out of Mem.

Table 4: The execution times of different tools for large graphs in seconds.

	Tool	Subgraph size			
		4	5	6	7
Gnutella	Subenum	**2.3**	**30**	**687**	**19,288**
	Serial Subenum	11.9	153	3,237	87,796
	FANMOD	15	442	11,715	324,152
	Kavosh	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	gtrieScanner	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	FaSe	3.0	49	1,108	29,423
Slash	Subenum	**1,040**	**460,928**	>1 year	>1 year
	Serial Subenum	3,963	>1 week	>1 year	>1 year
	FANMOD	67,047	>1 month	>1 year	>1 year
	Kavosh	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	gtrieScanner	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	FaSe	2,373	>1 week	>1 year	>1 year
Tweet	Subenum	**3,791**	>1 month	>1 year	>1 year
	Serial Subenum	**14,671**	>1 month	>1 year	>1 year
	FANMOD	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	Kavosh	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	gtrieScanner	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	FaSe	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
Notre	Subenum	**23,273**	>1 month	>1 year	>1 year
	Serial Subenum	**82,833**	>1 month	>1 year	>1 year
	FANMOD	451,156	>1 year	>1 year	>1 year
	Kavosh	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	gtrieScanner	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.
	FaSe	Out of Mem.	Out of Mem.	Out of Mem.	Out of Mem.

Another opportunity is using available powerful and low cost Graphical Processing Units (GPU). Due to the complexity of the problem, using parallel GPU based solutions like CUDA may also bring a huge performance boost.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[2] P. Ribeiro and F. Silva, "g-tries: an efficient data structure for discovering network motifs," in *Proceedings of the ACM Symposium on Applied Computing (SAC '10)*, pp. 1559–1566, 2010.

[3] Z. R. M. Kashani, H. Ahrabian, E. Elahi et al., "Kavosh: a new algorithm for finding network motifs," *BMC Bioinformatics*, vol. 10, no. 1, article 318, 2009.

[4] S. Wernicke and F. Rasche, "FANMOD: a tool for fast network motif detection," *Bioinformatics*, vol. 22, no. 9, pp. 1152–1153, 2006.

[5] J. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," in *Research in Computational Molecular Biology*, T. Speed and H. Huang, Eds., vol. 4453 of *Lecture Notes in Computer Science*, pp. 92–106, Springer, Berlin, Germany, 2007.

[6] F. Harary and E. Palmer, "The enumeration methods of Redfield," *American Journal of Mathematics*, vol. 89, no. 2, pp. 373–384, 1967.

[7] D. S. Johnson, "The NP-completeness column," *ACM Transactions on Algorithms*, vol. 1, no. 1, pp. 160–176, 2005.

[8] P. Ribeiro, F. Silva, and L. Lopes, "Parallel discovery of network motifs," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 144–154, 2012.

[9] Z. Zhao, M. Khan, V. S. A. Kumar, and M. V. Marathe, "Subgraph enumeration in large social contact networks using parallel color coding and streaming," in *Proceedings of the 39th International Conference on Parallel Processing (ICPP '10)*, vol. 10, pp. 594–603, September 2010.

[10] F. Schreiber and H. Schwbbermeyer, "Towards motif detection in networks: frequency concepts and flexible search," in *Proceedings of the International Workshop on Network Tools and Applications in Biology*, pp. 91–102, 2004.

[11] M. Kiyomi, *Studies on subgraph and supergraph enumeration algorithms [Ph.D. thesis]*, The Graduate University for Advanced Studies, 2006.

[12] P. Ribeiro, *Efficient and scalable algorithms for network motifs discovery [Ph.D. thesis]*, Porto University, 2011.

[13] A. Masoudi-Nejad, F. Schreiber, and Z. R. M. Kashani, "Building blocks of biological networks: a review on major network motif discovery algorithms," *IET Systems Biology*, vol. 6, no. 5, pp. 164–174, 2012.

[14] S. Wernicke, "Efficient detection of network motifs," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 4, pp. 347–359, 2006.

[15] E. Wong, B. Baur, S. Quader, and C.-H. Huang, "Biological network motif detection: principles and practice," *Briefings in Bioinformatics*, vol. 13, no. 2, pp. 202–215, 2012.

[16] P. Ribeiro and F. Silva, "G-tries: a data structure for storing and finding subgraphs," *Data Mining and Knowledge Discovery*, vol. 28, no. 2, pp. 337–377, 2014.

[17] P. Paredes and P. Ribeiro, "Towards a faster network-centric subgraph census," in *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM '13)*, pp. 264–271, IEEE, August 2013.

[18] X. Li, D. S. Stones, H. Wang, H. Deng, X. Liu, and G. Wang, "NetMODE: network motif detection without Nauty," *PLoS ONE*, vol. 7, no. 12, Article ID e50093, 2012.

[19] S. Khakabimamaghani, I. Sharafuddin, N. Dichter, I. Koch, and A. Masoudi-Nejad, "QuateXelero: an accelerated exact network motif detection algorithm," *PLoS ONE*, vol. 8, no. 7, Article ID e68073, 2013.

[20] W. Tie, J. W. Touchman, Z. Weiyi, E. B. Suh, and X. Guoliang, "A parallel algorithm for extracting transcriptional regulatory network motifs," in *Proceedings of the 5th IEEE Symposium on Bioinformatics and Bioengineering (BIBE '05)*, pp. 193–200, October 2005.

[21] M. Schatz, E. Cooper-Balis, and A. Bazinet, *Parallel Network Motif Finding*, 2008.

[22] P. Ribeiro, F. Silva, and L. Lopes, "Efficient parallel subgraph counting using G-tries," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 217–226, 2010.

[23] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[24] T. White, *Hadoop: The Definitive Guide*, Yahoo Press, 2012.

[25] Z. Zhao, G. Wang, A. R. Butt, M. Khan, V. S. A. Kumar, and M. V. Marathe, "SAHAD: subgraph analysis in massive networks using Hadoop," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS '12)*, pp. 390–401, Shanghai, China, May 2012.

[26] Z. Zhao, "Subgraph querying in relational networks: a mapreduce approach," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops (IPDPSW '12)*, pp. 2502–2505, May 2012.

[27] J. Cohen, "Graph twiddling in a MapReduce world," *Computing in Science and Engineering*, vol. 11, no. 4, Article ID 5076317, pp. 29–41, 2009.

[28] B. Wu and Y. Bai, "An efficient distributed subgraph mining algorithm in extreme large graphs," in *Artificial Intelligence and Computational Intelligence*, vol. 6319 of *Lecture Notes in Computer Science*, pp. 107–115, Springer, Berlin, Germany, 2010.

[29] F. N. Afrati, D. Fotakis, and J. D. Ullman, "Enumerating subgraph instances using map-reduce," in *Proceedings of the 29th International Conference on Data Engineering (ICDE '13)*, pp. 62–73, April 2013.

[30] A. G. Rudi, S. Shahrivari, S. Jalili, and Z. R. M. Kashani, "RANGI: a fast list-colored graph motif finding algorithm," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 2, pp. 504–513, 2013.

[31] G. Blin, F. Sikora, and S. Vialette, "GraMoFoNe: a cytoscape plugin for querying motifs without topology in protein-protein interactions networks," in *Proceedings of the 2nd International Conference on Bioinformatics and Computational Biology (BICoB '10)*, pp. 38–43, March 2010.

[32] G. Blin, *Combinatorial objects in bio-algorithmics: related problems and complexities [Ph.D. thesis]*, Université Paris-Est, 2012.

[33] L. Babai and E. M. Luks, "Canonical labeling of graphs," in *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pp. 171–183, 1983.

[34] B. D. McKay, "Practical graph isomorphism," *Congressus Numerantium*, vol. 30, pp. 45–87, 1981.

[35] T. Junttila and P. Kaski, "Engineering an efficient canonical labeling tool for large and sparse graphs," in *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments and the 4th Workshop on Analytic Algorithms and Combinatorics*, pp. 135–149, January 2007.

[36] H. Katebi, K. Sakallah, and I. Markov, "Conflict anticipation in the search for graph automorphisms," in *Logic for Programming, Artificial Intelligence, and Reasoning*, N. Bjørner and A. Voronkov, Eds., vol. 7180 of *Lecture Notes in Computer Science*, pp. 243–257, Springer, Berlin, Germany, 2012.

[37] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon, "Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs," *Bioinformatics*, vol. 20, no. 11, pp. 1746–1758, 2004.

[38] M. G. Pablo and L. Danon, "Community structure in jazz," *Advances in Complex Systems*, vol. 6, no. 4, pp. 565–573, 2003.

[39] J. Stehlé, N. Voirin, A. Barrat et al., "High-resolution measurements of face-to-face contact patterns in a primary school," *PLoS ONE*, vol. 6, no. 8, Article ID e23176, 2011.

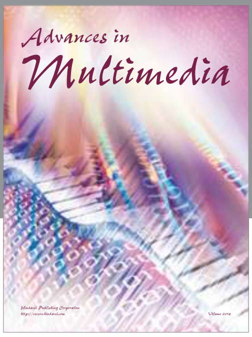
[40] A.-C. Gavin, P. Aloy, P. Grandi et al., "Proteome survey reveals modularity of the yeast cell machinery," *Nature*, vol. 440, no. 7084, pp. 631–636, 2006.

[41] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: densification and shrinking diameters," *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, article 2, 2007.

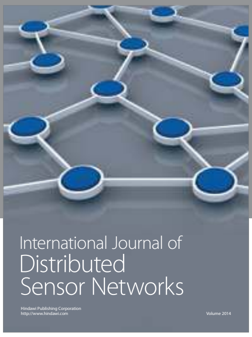
[42] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proceedings of the 19th International World Wide Web Conference (WWW '10)*, pp. 641–650, April 2010.

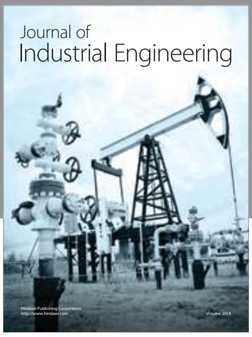
[43] J. Leskovec and J. McAuley, "Learning to discover social circles in ego networks," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS '12)*, pp. 539–547, December 2012.

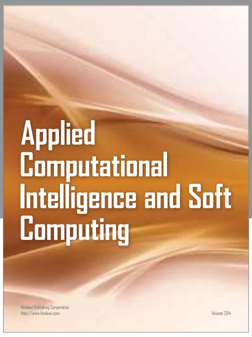
[44] R. Albert, H. Jeong, and A.-L. Barabási, "Internet: diameter of the World-Wide Web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.

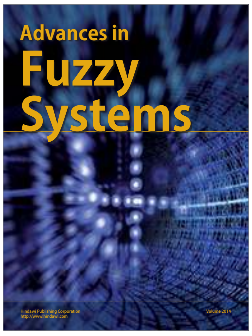

Advances in
Multimedia

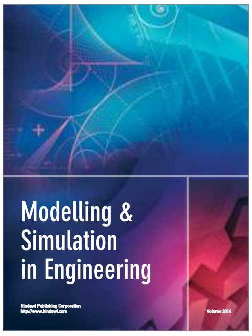
The Scientific
World Journal

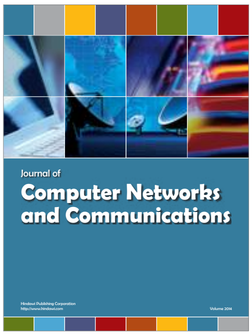

International Journal of
Distributed
Sensor Networks

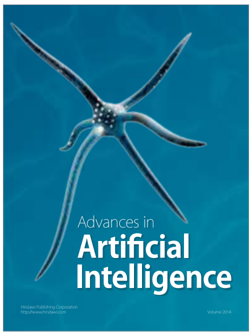

Journal of
Industrial Engineering

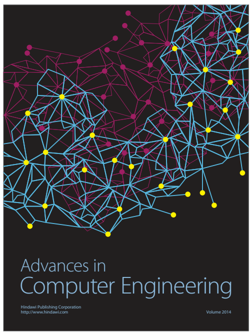

Applied
Computational
Intelligence and Soft
Computing


Advances in
Fuzzy
Systems

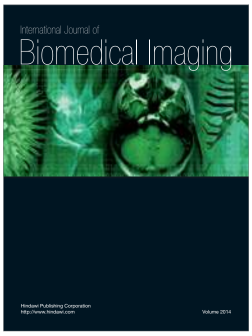

Modelling &
Simulation
in Engineering

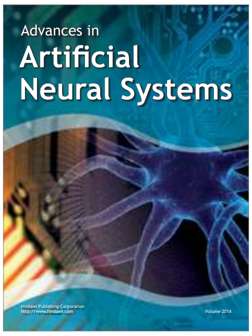

Journal of
Computer Networks
and Communications

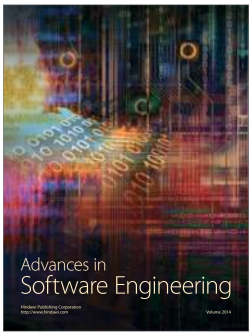

Advances in
Artificial
Intelligence


Advances in
Computer Engineering

International Journal of
Computer Games
Technology

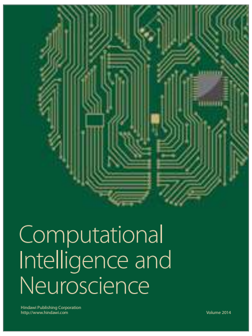

International Journal of
Biomedical Imaging


Advances in
Artificial
Neural Systems


Advances in
Software Engineering

Journal of
Robotics

Advances in
Human-Computer
Interaction


Computational
Intelligence and
Neuroscience

International Journal of
Reconfigurable
Computing

Journal of
Electrical and Computer
Engineering