APPROXIMATE CYCLES COUNT IN UNDIRECTED GRAPHS

Maytham Safar
Computer Eng. Dept., College of Eng. & Petroleum, Kuwait University
P.O. Box 5969 Safat 13060, Kuwait,
Email: maytham.safar@ku.edu.kw

Khaled Mahdi
Chemical Eng. Dept., College of Eng. & Petroleum, Kuwait University
P.O. Box 5969 Safat 13060, Kuwait
Email: khaled.mahdi@ku.edu.kw

Hisham Farahat
Computer Eng. Dept., College of Eng. & Petroleum, Kuwait University
P.O. Box 5969 Safat 13060, Kuwait
Email: hishamfarahat@gmail.com

Saud Albehairy
Computer Eng. Dept., College of Eng. & Petroleum, Kuwait University
P.O. Box 5969 Safat 13060, Kuwait
Email: saud331@gmail.com

Ali Kassem
Computer Eng. Dept., College of Eng. & Petroleum, Kuwait University
P.O. Box 5969 Safat 13060, Kuwait
Email: ali.qassim@gmail.com

Khalid Alenzi
Computer Eng. Dept., College of Eng. & Petroleum, Kuwait University
P.O. Box 5969 Safat 13060, Kuwait
Email: engineer.khalid@gmail.com

Received 13 February 2012

Accepted 14 February 2012

Abstract

In social networks, counting the number of different cycle sizes can be used to measure the entropy of the network that represents its robustness. The exact algorithms to compute cycles in a graph can generate exact results but they are not guaranteed to run in a polynomial time. We present an approximation algorithm for counting the number of cycles in an undirected graph. The algorithm is regression-based and guaranteed to run in a polynomial time. A set of experiments are conducted to compare the results of our approximate algorithm with the results of an exact algorithm based on the Donald-Johnson backtracking algorithm.

Keywords: Complex networks, social networks, cyclic entropy, cycles

1. Introduction

Social networks are becoming a trend recently in the online community. A social network consists of people (i.e. online subscribers or users) and the relations between them. Examples of famous social network applications include Paltalk, Hi5, Facebook and mySpace. Such dynamic and complex networks are used to share information and media [1, 2] and many applications and systems are developed to mine those complex networks for similar information or media of interest to the users [3, 4].

Social networks are further subject to the analysis of researchers by studying the flow of communication and information exchange within the social network [5, 6] and their evolution from different perspectives and applications [7]. Such analysis is required in order to measure the social network vulnerability to the spreading of rumors, diseases, news, viruses etc. In addition to the above, an important measure when analyzing a social network would be its robustness. The robustness measures the strength of the network to resist failures and attacks.

Robustness of a network is measured by finding its entropy [8]. Entropy of a network is calculated by the evaluation of the cycles' degree distribution, which requires counting the number of cycles existing in the network. The problem of finding and counting circuits and cycles in large graphs has been of interest to researchers lately due to its challenging complexity. Exhaustive enumeration, even by smart algorithms proposed in earlier research, is restricted to small graphs as the number of cycles grows exponentially with the size of the graph. Therefore, it is believed that it is unlikely to find a precise and an efficient algorithm for counting circuits. Finding a method of approximation to this problem is the alternative.

Counting cycles in a graph is an *NP-Complete* problem. Hence, in our previous work [9], we proposed a *DFS-XOR-based* approximation algorithm to count the cycles in un-directed graph. The algorithm starts by running depth first search (*DFS*) on a randomly selected vertex of the network. Next, it starts counting the cycles starting with the ones that contain that vertex. The approximation is achieved from the number of *XOR* levels performed in the exact solution. In order to count all the cycles we have to perform *n XOR* levels, equal to the number of unique cycles. Our solution was to perform *z XOR* levels where $z < n$. z is a value decided by the user of the algorithm. By doing so, we can adjust the calculation time of the algorithm by sacrificing some of the accuracy. As z increases the generated results become more accurate and the algorithm becomes slower and vice versa. If $z = n$ then the generated result will be the exact solution, however, it will lead to an exponential cost to compute all the n *XOR* levels.

The focus of this research paper will be counting the number of cycles in a graph representing a social network, which is considered a preliminary step for measuring the robustness. We present a new regression-based approximation algorithm that counts the number of cycles in a guaranteed polynomial time.

2. Related Work

This section of the research presents several algorithms addressing the same problem for counting the number of cycles in a given graph.

The algorithm in [10] is an approximation algorithm that has proven its efficiency in estimating large number of cycles in polynomial time when applied to real world networks. It is based on transferring the cycle count problem into statistical mechanics model to perform the required calculation. The algorithm counts the number of cycles in random, sparse graphs as a function of their length. Although the algorithm has proven its efficiency when it comes to real world networks, the result is not guaranteed for generic graphs.

The algorithm in [11] is based on backtracking with the look-ahead technique. It assumes that all vertices are numbered and start with vertex s. The algorithm finds the elementary paths, which start at s and contain vertices greater than s. The algorithm repeats this operation for all vertices in the graph. The algorithm uses a stack in order to track visited vertices. The advantage of this algorithm is that it guarantees finding an exact solution for the problem. The complexity of the algorithm is $O((V+E)(C+1))$. Where, V: number of vertices. E: number of edges. C: number of cycles. The time bound of this algorithm depends on the number of cycles, which grows exponentially in real life networks.

Hangbo Liu and Jiaxin Wang [12] presented an algorithm based on cycle vector space methods. A vector space that contains all cycles and union of disjoint cycles is formed using the spanning of the graph. Then, vector math operations are applied to find all cycles. This algorithm is computationally complex, since it investigates all vectors and only a small portion of them could be cycles.

We propose an approximation algorithm named Regression-based approximation algorithm that have an advantage over the algorithm proposed in [5] and [9]. It can guarantee the correctness of the results for all graph types. Moreover, our approximation algorithm is more time efficient when it comes to real life problems of counting cycles in graphs, because their complexity is not dependant on the factor of number of cycles, and can guarantee a polynomial time execution.

3. Cycles Counting Algorithms

Counting cycles in graphs is an NP-Complete problem. NP-Complete problems are a challenging area for computer scientists since until now there is no algorithm discovered to solve them in polynomial time.
The idea of updating an existing algorithm in order to enhance its performance is tricky. Usually, there is a tradeoff between the accuracy of the algorithm and its cost [9]. Designers will have to sacrifice a certain accuracy percentage for the sake of gaining an enhancement in the time or space complexities of the algorithm.
The challenge faced is to develop approximation algorithms that give an approximate solution for the problem of counting the number of cycles in a guaranteed polynomial time.

3.1. *Exact Algorithm*

With this algorithm we are interested in finding the number of cycles for each possible cycle length. We developed an algorithm to find all the simple cycles in a graph that is based on an algorithm created by Johnson [13]. The algorithm is based on backtracking technique. It starts with node *s*, which is the vertex with the least ID, and begins to enumerate all cycles that passes through *s*. This is done by building a simple path starting from s using a stack to save the vertices. Whenever *s* is encountered again, a cycle is created and printed. Addition to that any vertex is currently in a path (stored in the stack) is being blocked so it cannot be added again to the stack. When a node is finished (the algorithm passes through all of its edges), it is being popped from the stack and unblocked for future use. After enumerating all the cycles with *s* is a common node, the algorithm removes *s* from the graph *G* and starts the process with the second least vertex. These steps are repeated until *G* has two nodes. Since our graph is undirected, the equation

$$C_{un} = \frac{C_{d\sigma} - e}{2} \qquad (1)$$

is used to convert from directed cycles to undirected cycles. Where $C_{d\sigma}$ is the total number of cycles in the directed version of the original undirected graph (i.e. replace each undirected edge with two directed edges in opposite directions). C_{un} is the total number of cycles in the undirected graph and e is the total number of edges in the graph.

The pseudo code for this algorithm is shown in Algorithm 1. We summarize the algorithm with these steps:

(i) Assign IDs for all the nodes in the graph.
(ii) Choose node *S* as the node with least ID.
(iii) Initialize a path by making *S* is the root of the path.
(iv) Start a depth first traversal using *S* as the root, for each new unblocked node add it to the current path.
(v) If *S* is found again, then a new cycle is found and displayed.
(vi) When a node is finished, it set to unblocked so that it can be used in other circuits.
(vii) After finishing all the nodes, remove *S* from the graph and start again from step 2.
(viii) If *S* is the last node in the graph, end the algorithm.

Before finding the cycles, a simplification process is applied to the graph that removes nodes and edges that cannot be a part of any cycle. This speeds up the Johnson algorithm. This process can be summarized with these steps:

(i) Multi edges between the same nodes are considered as one edge.
(ii) Remove nodes with degree less than 2 because these nodes are leaves that will never form a cycle.
(iii) Remove cut-edges, which are edges that if removed from a graph, the graph will be divided into two or more subgraphs.

We noticed that steps from 2 to 7 are similar and repeated for each node, the only difference is that the graph has one less node than its previous. Using this we can parallelize the algorithm by creating *N-1* threads, each thread is responsible of finding cycles originating from its root.

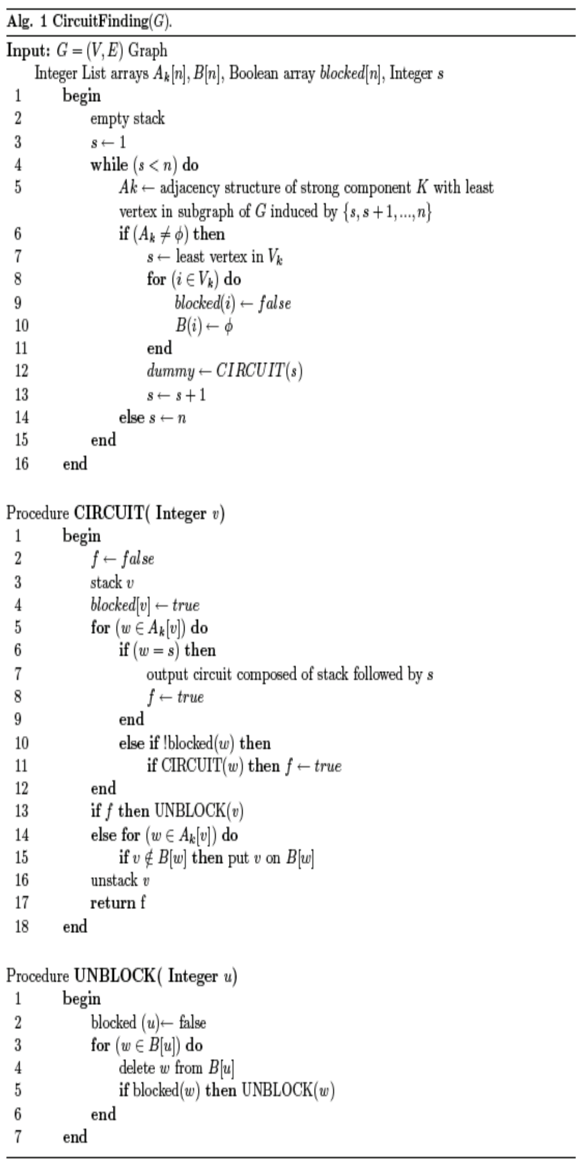
The complexity of Johnson's algorithm is *O((n + e)c)* where *n* is the number of nodes, *e* is the number of edges and *c* is the number of circuits in the graph. To have a sense of how the cycles' computation problem is an *NP-complete*, Table 1 summarizes the results of the algorithm running on complete graphs which has the worst case running times. See Figure 1 for the details of the exact algorithm.

Table 1. Exact algorithm execution time.

Number of nodes	Number of cycles	Time (msec)
5	37	313
8	8018	906
11	5488059	31671
13	710771275	3188079

3.2. ***Regression-based Approximation Solution***

Here, we propose an efficient approximation algorithm that is based on the work in [14]. It is based on the algorithm that has been introduced in [10]. Some modifications on the algorithm has been introduced to give more accurate results. The new algorithm uses curve fitting (regression) to estimate the number of circuits (cycles) in an undirected graph and provides analytical results for the typical entropy of circuits in sparse random graphs. The approximation in this algorithm is based on a statistical mechanics approach. It uses a Bethe approximation technique [15], and iterations of the Belief Propagation equations and an approximation method to approximate the statistical mechanics model and find the cycles distribution. Two methods can be used as an approximation algorithms Monte Carlo simulation or Bethe approximation. Bethe is used here because of the well-known correspondence between both Bethe and Belief Propagation. First of all, the graph is reduced; all leaf nodes (nodes with degree 1 or 0) are removed from the graph. Each edge of the graph is initialized with a random positive value $y^{(0)}$. Each edge is iterated from its initial value until convergence reaching to a fixed value of y^{*}. Convergence is determined according to some accuracy level. In this algorithm, the convergence is considered



```
Alg. 1 CircuitFinding($G$).
Input: $G = (V, E)$ Graph
    Integer List arrays $A_k[n]$, $B[n]$, Boolean array $blocked[n]$, Integer $s$
begin
    empty stack
    $s \leftarrow 1$
    while $(s < n)$ do
        $Ak \leftarrow$ adjacency structure of strong component $K$ with least
        vertex in subgraph of $G$ induced by $\{s, s+1, ..., n\}$
        if $(A_k \neq \phi)$ then
            $s \leftarrow$ least vertex in $V_k$
            for $(i \in V_k)$ do
                $blocked(i) \leftarrow false$
                $B(i) \leftarrow \phi$
            end
            $dummy \leftarrow CIRCUIT(s)$
            $s \leftarrow s + 1$
        else $s \leftarrow n$
    end
end

Procedure CIRCUIT( Integer $v$)
begin
    $f \leftarrow false$
    stack $v$
    $blocked[v] \leftarrow true$
    for $(w \in A_k[v])$ do
        if $(w = s)$ then
            output circuit composed of stack followed by $s$
            $f \leftarrow true$
        end
        else if !blocked$(w)$ then
            if CIRCUIT$(w)$ then $f \leftarrow true$
    end
    if $f$ then UNBLOCK$(v)$
    else for $(w \in A_k[v])$ do
        if $v \notin B[w]$ then put $v$ on $B[w]$
    unstack $v$
    return f
end

Procedure UNBLOCK( Integer $u$)
begin
    blocked $(u) \leftarrow$ false
    for $(w \in B[u])$ do
        delete $w$ from $B[u]$
        if blocked$(w)$ then UNBLOCK$(w)$
    end
end
```

Figure 1. Pseudo code for the Exact Algorithm

to be satisfied when $\left|y^{T+1} - y^{T}\right| \leq 0.001$. The value y represents the probability that the edge is present in a cycle c. The y value can be calculated using the following equation:

$$y_{i\to j}^{T+1} = \frac{u\Sigma_{m\in\beta_{i-j}} y_{m\to i}^{T}}{1 + 0.5u^{2}\Sigma_{m,n\in\beta_{i-j} \quad m\neq n} y_{m\to i}^{T} y_{n\to i}^{T}} \qquad (2)$$

where u is a positive real value. Then from all y's two values are calculated; C_L and

$$L = \sum_{(i,j)\in E} \frac{u y^*_{i\to j} y^*_{j\to i}}{1 + u y^*_{i\to j} y^*_{j\to i}} \tag{3}$$

$$\begin{aligned} R = {} & \frac{1}{N}\sum_{i\in V} \ln\left(1 + 0.5u^2 \sum_{m,n\in\beta_{i-j} \quad m\neq n} y^*_{m\to i} y^*_{n\to i}\right) \\ & - \frac{1}{N}\sum_{(i,j)\in E} \ln(1 + u y^*_{i\to j} y^*_{j\to i}) \\ & - \frac{L\ln(u)}{N} \end{aligned} \tag{4}$$

$$C_L = e^{RN} \tag{5}$$

where

- β_i is the set of neighbors of node i.
- β_{i-j} is the set of neighbors of i except j.
- N is the number of nodes in the graph.
- C_L is the number of cycles of size L.

Refer to [5,14] for further details on the above equations.

The procedure explained above is repeated starting from an initial value of $u = u_0$ to $u = u_{\max}$. Where u_0 and $u_{\max}$ are greater than 0. At each iteration step, a new distribution point (L, C_L) is produced. The iteration step for u is 0.0001 at the early stages of the algorithm. This value is not fixed. It will be changed when $L_{new} - L_{old} < 0.001$ (i.e. the progress in L is slow). If this condition is satisfied, u is increased by 10%. As noticed from the equations above, the output at each step (L, C_L) depends on u. At specific stages of the iteration (when u gets large), much iteration is wasted giving nearly the same point. To avoid this condition, a jump in u is made. This algorithm yields a plot of (L, C_L) points. To extract the needed distribution points (3 to n), we use regression. Based on a work done by [16], the guassian equation models the cycles' distributions of a graph.

$$y = a.e^{-(\frac{x-b}{c})^2} \tag{6}$$

We used Equation 6 to fit the curve and find the function that represents the distribution. The pseudo code for this algorithm is shown in Figure 2. This algorithm has a running time that has a polynomial growing trend with the graph size and logarithmically with the required accuracy. Since the algorithm uses the adjacency matrix to represent the graph, the space complexity is $O(2n^2)$.

4. Experiments

The scope of the experiments is to compare the results of running an algorithm (based on Donald Johnson backtracking algorithm) that finds an exact solution in super power hardware with results of running the approximate solutions introduced in the paper in a limited power hardware. We have used a real social network extracted from the Paltalk with 26 nodes.

4.1. *Exact Algorithm*

In this experiment [8] a grid of 30 Mac Pro machines watch with two 2.66 dual core Intel processors, which gives a total of 319.2 GHZ processor power. One extra machine is used to control the grid and distribute the tasks to the machines (called clients). The controller follows these steps: 1- read the network from the database, 2-Create the graph based on the chosen model, 3- divide the algorithm into threads and 4- submit each thread as a task to the clients. XGrid has been used as the distributed computing protocol that is developed by Apple and preinstalled on Mac OS X Leopard. The code was written to be compiled and run under Java 5. With this setup, each thread (mentioned above) can be executed on a single core, which gives us a parallelism of 120 threads at a time.

The application was running for 11 hours to generate the obtained results. The actual entropy was computed as 2.21, and the actual probability distribution is shown in Figure 3. The same experiment was re-run on a regular Intel Core Duo based laptop with 2GB of RAM and it took 22 hours to calculate the same entropy and probability distribution. This proves that throwing extra hardware and resources to solve an NP problem would only yield a marginal improvement in the performance. However, advising better algorithms would yield a better performance as we will show in the next set of experiments.

4.2. *Regression-based Approximation Solution*

For this experiment, we have used a regular Intel Core Duo based laptop with 2GB of RAM. Figure 3 shows a comparison between the probability distribution of the cycles (from which we compute the entropy) of the three algorithms. It is very obvious that Regression-based approximation algorithm had a better matching distribution to the actual distribution related to the DFS-XOR-based approximation algorithm. The approximation entropy was computed as 2.19 with an error of less than 1%. The new Regression-based approximation solution reached this accuracy in only 140 seconds (using a slower machine). Hence, the performance of Regression-based approximation algorithm is three orders of magnitude less than the running time of DFS-XOR-based approximation solution, and five orders magnitude less than the exact algorithm.

5. Conclusion and Future Work

Counting the number of cycles in a social network is usually used to calculate the entropy of that network. This entropy is used to measure the robustness and stability of the network. Finding the exact number of cycles is an NP problem, hence we have proposed an approximation algorithm that is regression-based. Although the approximation algorithm discovers only a small percentage of the cycles in the network, the calculated entropy is very accurate. This is due to the fact that those algorithms are able to find a similar probability distribution of the cycles to the actual network. However, the approximate algorithms are using limited resources and a few orders of magnitude less time to execute. The developed algorithm can be further enhanced to be applied to directed graphs, and other networks.

```
Regression-based ApproximationAlgo($G$)
Input: Undirected graph $G$.
Output: Array $A[1..n]$ of size $n$ which contains the cycle count of each length.
1:  Begin
2:    $A[1] \leftarrow A[2] \leftarrow 0$
3:    Reduce $G$ by removing all leaves nodes
4:    $U \leftarrow 0$
5:    Points $\leftarrow$ null set of points
6:    while($L < n$)
7:    begin
8:        $y$ = random number between 0 and 10 ( do for all edges)
9:        while($|y_{new} - y|<0.0001$ for all edges)
10:           Calculate $y_{new}$ using equation 2 for all edges twice
11:           $y \leftarrow y_{new}$
12:       $L \leftarrow$ equation 3
13:       $C_L \leftarrow$ equation 5
14:       Add ($L$,$CL$) to points
15:       $u \leftarrow u + 0.0001$
16:       if( $|L_{new}-L_{old}|<0.001$)
17:           $u \leftarrow u * 1.1$
18:   end
19:   for($i$ = 3 to $n$)
20:   begin
21:       Find two points that surround $i( L_p< i < L_{p+1})$
22:       Calculate $C_i$ using interpolation
23:       $A[i] \leftarrow C_i$
24:   end
```

Figure 2. Pseudo code for Regression-based Approximation Algorithm

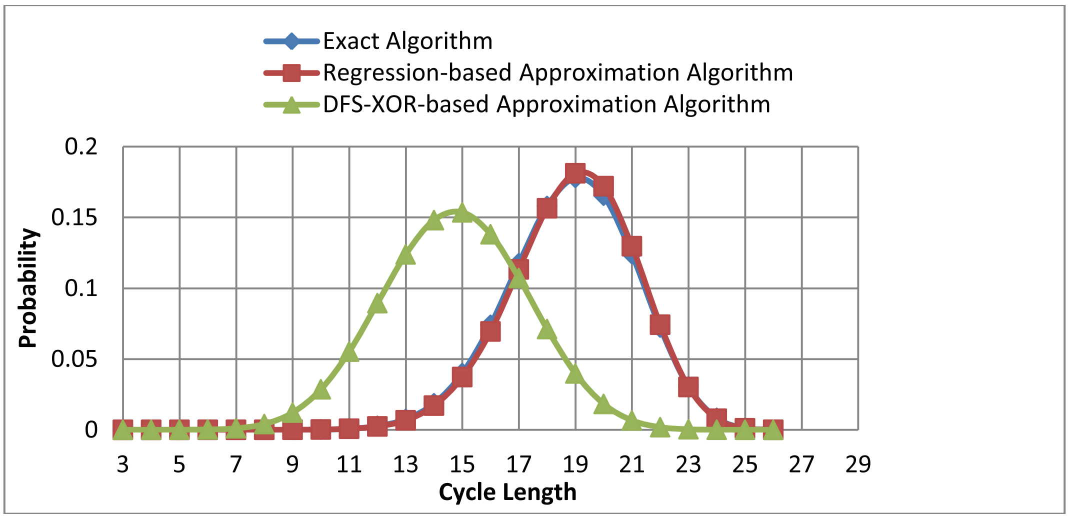


Figure 3. Comparison of the probability distribution of the three different algorithms

References

1. R. Angelova, M. Lipczak, E. Milios, P. Pralat: Investigating the Properties of a Social Bookmarking and Tagging Network. International Journal of Data Warehousing and Mining, (2010), 6(1): 1-19.
2. S. Papadopoulos, A. Vakali, I. Kompatsiaris: The Dynamics of Content Popularity in Social Media. International Journal of Data Warehousing and Mining, (2010), 6(1): 20-37.
3. R. Wetzker, C. Zimmermann, C. Bauckhage: Detecting Trends in Social Bookmarking Systems: A del.icio.us Endeavor. International Journal of Data Warehousing and Mining, (2010), 6(1): 38-57.
4. V. Torra, Y. Narukawa: Word similarity from dictionaries: Inferring fuzzy measures from fuzzy graphs. International Journal of Computational Intelligence Systems (IJCIS), (2008), 1: 19-24.
5. E. Marinari, and G. Semerjian, *On the number of circuits in random graphs*, Journal of Statistical Mechanics: Theory and Experiment (2006).
6. T. Takashita, Y. Abe, T. Itokawa, T. Kitasuka, M. Aritsugi: Design and implementation of a system for finding appropriate tags to photos in Flickr from Web browsing behaviour. International Journal of Web and Grid Services (2011), 7(1): 75-90.
7. C. Grilo, L. Correia: The influence of the update dynamics on the evolution of cooperation. International Journal of Computational Intelligence Systems (IJCIS), (2009), 2(2): 104-114.
8. K. Mahdi, H. Farahat, and M. Safar, *Temporal Evolution of Social Networks in Paltalk*, in Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services (2008).
9. M. Safar, K. Alenzi, and S. Albehairy, *Counting Cycles in an Undirected Graph using DFS-XOR Algorithm*, In the Proceedings of the First International Conference on 'Networked Digital Technologies' (NDT), (2009).
10. E. Marinari, R. Monasson and G. Semerjian, *An algorithm for counting circuits: application to real world and random graphs*, Europhysics Letter, (2006).
11. R. Tarjan, *Enumaration of the Elementary Circuits of a Directed Graph*, Cornell University Ithaca,(NY, USA, 1972), Technical Report: TR72-145.
12. H. Liu, and J. Wang, *A new way to enumerate cycles in a graph*, International Conference on Internet and Web Applications and Services.
13. D. Johnson, *Finding All the Elementary Circuits of a Directed Graph*. SIAM Journal on Computing(1975) , pp. 77-84.
14. K. Mahdi, M. Safar, and H. Farahat *Analysis of Temporal Evolution of Social Networks*, In the Journal of Mobile Multimedia (JMM), Rinton Press (Princeton, New Jersey, 2009), 5(4) 333-350.
15. Rios, P. D. L., S. Lise and A. Pelizzola, *Bethe approximation for self-interacting lattice trees*, (2001) Europhysics Letters 53 176-182.
16. M. Safar, K. Mahdi and A. Qassim, *Universal Cycles Distribution Function of Social Networks*, In the Proceedings of the First International Conference on 'Networked Digital Technologies' (NDT), (2009).
17. The university of Texas at Austin, *Cycles in an undirected graph,* http://www.me.utexas.edu/~bard/IP/Handouts/cycles.pdf .
18. J. Yedidia, W.T. Freeman and Y. Weiss, *Understanding Belief Propagation and its Generalizations*, Mitsubishi Electric Research Laboratorie,(2002).
19. Erdos Renyi model: http://en.wikipedia.org/wiki/Erdős–Rényi_model.
20. T. Takashita, Y. Abe, T. Itokawa, T. Kitasuka, M. Aritsugi: *Design and implementation of a system for finding appropriate tags to photos in Flickr from Web browsing behaviour*, International Journal of Web and Grid Services(2011), 7(1): 75-90.
21. R. Angelova, M. Lipczak, E. Milios, P. Pralat: *Investigating the Properties of a Social Bookmarking and Tagging Network*, International Journal of Data Warehousing and Mining(2010), 6(1): 1-19.
22. S. Papadopoulos, A. Vakali, I. Kompatsiaris: *The Dynamics of Content Popularity in Social Media*, International Journal of Data Warehousing and Mining(2010), 6(1): 20-37.
23. R. Wetzker, C. Zimmermann, C. Bauckhage: *Detecting Trends in Social Bookmarking Systems*: A del.icio.us Endeavor, International Journal of Data Warehousing and Mining (2010), 6(1): 38-57.