Metodološki zvezki, Vol. 8, No. 1, 2011, 57-77

Statistical Methods for Social Networks: A Focus on Parallel Computing

Marina Marino[1] and Agnieszka Stawinoga[2]

Abstract

Network analysis and modeling have received considerable attention in recent times and require the solution of intricate mathematical problems, e.g. the problem of enumeration graphs under specified conditions, finding the largest complete graph and so on. Even though a lot of well-known algorithms have been proposed, some problems are still challenges from a computational point of view and their fast solutions are thus of great practical interest. This paper focuses on some parallel algorithms for social network analysis. In particular, a review of some existing parallel algorithms is carried out and a new parallel algorithm is proposed for parameters estimation in Exponential Random Graph Models.

1 Introduction

During the last few years statistical methods for analysing social networks have been developed. From a descriptive point of view there are many well-known techniques that measure properties of a network, of nodes, or of subsets of nodes (e.g., density, centrality, cohesive subsets). These techniques serve valuable purposes in describing and understanding network features that might bear on particular research questions. However to understand the uncertainty associated with the observed outcomes, it is necessary to associate a statistical model to these measures. Statistical models, also give the possibility to derive estimation for underlying parameters and to test whether certain network substructures are more commonly observed in the network than might be expected by chance. Then, it is possible to develop hypotheses about the social processes that might produce these structural properties. In particular, in recent years, there has been growing interest in Exponential Random Graph Models (ERGMs) for networks. These probability models provide a flexible way to model social network dependence structure. In this model, the parameters are unknown coefficients and must be estimated.

[1] Department of Mathematics and Statistics, University of Naples Federico II, Naples, Italy; mari@unina.it

[2] Department of Mathematics and Statistics, University of Naples Federico II, Naples, Italy; agnieszka.stawinoga@unina.it

Parameter estimation as well as computation of some characteristic network indices can require too much computation time to make them unfeasible for very large networks.

In this paper, section 2 introduces some notations and basic definitions. Then we provide a brief introduction to challenges in parallel computing for networks (section 3). After an overview of parallel algorithms developed to analyze social network structures (section 4), we introduce ERGMs and the Robbins-Monro algorithm for estimating the model parameters (section 5). In section 6 we propose a way to reduce the Robbins-Monro computation time by exploiting parallelism of the algorithm; computational results, in terms of efficiency, obtained when implementing our algorithm on a parallel computer are shown.

2 Definitions

One of the most useful ways to represent networks is by means of graphs. Indeed, a graph, consisting of vertices (nodes) joined by edges (links), provide a very flexible abstraction for describing relationships between discrete objects.
In the following analysis we will consider a graph $G = (V,E)$, where V is the set of vertices representing actors in the social network, and E, the set of edges representing the relationships between the actors, called also ties. We denote the number of nodes and edges by n and m, respectively.

The *degree* $\deg(v)$ of a vertex $v \in V$ is the number of edges incident to v, with loops being counted twice. A vertex of degree 0 is an isolated vertex.

A graph is a *weighted graph* if a positive number (*weight*) $\omega(e)$ is assigned to each edge. Such weights might represent, for example, costs, lengths or capacities, etc. depending on the problem. The weight of the graph is the sum of the weights given to all edges. For *unweighted graphs*, it is assumed that $\omega(e)=1 \ \forall e \in E$.

A graph can be *directed*, if edges have direction, or *undirected*. In a directed graph the edges are usually called arcs and vertices have both *in-degrees* and *out-degrees*: the in-degree of vertex v is the number of arcs ended with v and the out-degree is the number of arcs which originate with v.

The *density* of a graph is a proportion of the number of edges present in a graph to the maximum possible number of edges in a graph with n vertices.

A *path* from $v \in V$ to $s \in V$ is an alternating sequence of vertices and edges, beginning with v and ending with s, such that each edge connects its preceding vertex with its succeeding one. The *length* of a path is the sum of the weights of its edges.

Two vertices u and v are called *adjacent* if an edge exists between them.

A *clique* in a graph is a subset of nodes, all of which are adjacent to each other, and there are no other nodes that are also adjacent to all of the members of the clique. A *maximal clique* is defined as a clique that cannot be contained in

other cliques. The maximal clique with the largest size is called *maximum clique*. A 3-vertex clique is called *triangle*.

The *degree distribution* is the probability distribution of degrees of vertices over the whole graph; so, the degree distribution $P(k)$ of a graph is the fraction of vertices in the graph with degree k.

A *scale-free network* is a network whose degree distribution follows a power law, at least asymptotically.

3 Parallel computing and its challenges for network

It is clear that the availability of efficient algorithms for solving problems related to graph theory is essential in order to analyze networks. Moreover, real-world networks are often very large in size, ranging from several hundreds of thousands to billions of vertices and edges so large-scale network analysis is a very interesting area of research that has found applications in social networks (friendship networks), the internet (the world-wide web), transportation networks and biological networks (protein-interaction networks). This implies the solution of graph-related problems on large-scale data. In particular, the interest is devoted to study two characteristics of network: *centrality* (which nodes in the graph are best connected to others, or have the most influence) and *connectivity* (how nodes are connected to one another). However, as these problems grow larger in scale, computation and memory capacities of a single processor computer may not be enough (space-efficient memory representation of large graphs is itself a big challenge). The development of parallel computer systems has made tractable large-size problems that would otherwise take an exceedingly long time.

In the following analysis, with the term "parallel computer", we will refer to a set of processors that are able to work cooperatively to solve a computational problem. This definition is broad enough to include parallel supercomputers that have hundreds or thousands of processors, networks of workstations, multiple-processor workstations, and embedded systems. Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").

However, in the case of graph-related problems, solving algorithms have some inherent characteristics that make them poorly adapted to an implementation on current parallel computer. This is due to the fact that massive graphs that occur in real-world applications are often not amenable to a balanced partitioning among processors of a parallel system. Several properties of graph problems which create important challenges for efficient parallelism for solving large-scale graph problems are extensively discussed by Lumsdaine et al. (2007).

First of all, the computations performed by a graph algorithm are dictated by the vertex and edge (node and link) structure of the graph on which it is operating.

As a result, parallelism based on partitioning of computation (task parallelism) can be difficult to exploit because the structure of computations in the algorithm is not known a priori.

Moreover, the data in graph problems are typically unstructured and highly irregular. This means that data-access patterns and workload are only known at run-time, as well as data dependency, which is dynamic. So, similar to the difficulties encountered in parallelizing a graph problem based on its computational structure, the irregular structure of graph data makes it difficult to introduce parallelism by partitioning the problem data (data parallelism). In particular, scalability can be quite limited by unbalanced computational loads resulting from poorly partitioned data.

Finally, algorithms to solve graph-related problems are typically memory intensive, and the memory accesses are fine-grained and highly irregular. Indeed, relatively small amounts of computational work are done between memory accesses, so there is a higher ratio of data access to computation than for other scientific computing applications. Moreover, because graphs represent the relationships between entities and because these relationships may be irregular and unstructured, the data access patterns tend not to have very much locality. So, runtime can be dominated by the waiting time of memory. Thus, the execution time of a graph computation strongly correlates with the memory subsystem performance, rather than the processor clock frequency or the floating-point processing capabilities of the system. This leads to poor performance on parallel computing systems. On distributed memory clusters, few parallel graph algorithms outperform their best sequential implementations due to long memory latencies and high synchronization costs. Parallel shared memory systems are a more supportive platform. They offer higher memory bandwidth and lower latency than clusters, as the global shared memory avoids the overhead of message passing. However, parallelism is dependent on the cache performance of the algorithm and scalability is limited in most cases. While it may be possible to improve the cache performance to a certain degree for some classes of graphs, there are no known general techniques for cache optimization because the memory access pattern is largely dependent on the structure of the graph.

4 Social network structural properties and parallel computing: state of art

The main topic of Social Network Analysis (SNA), as stated in Carrington et al. (2005), is to “measure the structural properties of networks and/or relational properties of particular objects/actors within them”. In this section we present a quick overview of parallel algorithms developed for analyzing several structural properties of social networks. In particular, we will focus on parallel solutions

proposed for solving problems such as centrality indices calculation, maximal cliques enumeration and triangles counting.

4.1 Centrality indices

Centrality is one of the most important and widely used measurements in SNA (Carrington et al., 2005). It is a descriptive characteristic for actors/ networks with various structural properties, and a crucial parameter in analyzing and understanding the actor roles in social network. SNA researchers usually use centrality to identify the most powerful, influential or critical actors. The most fundamental and popular definitions of centrality were proposed by Freeman in the late 1970s (Freeman, 1979). The author developed three measures of centrality, one relative and one absolute measure of the centrality of positions in a network, and one reflecting the degree of centralization of the entire network. These measures are based on degree, closeness and betweenness concepts, respectively.

Bader and Madduri (2006) presented parallel algorithms for computing those centrality indices, optimized for scale-free sparse graphs. These algorithms have been optimized to exploit properties typically observed in large scale real-world networks, such as the low average distance, high local density, and heavy-tailed power law degree distributions. The authors presented implementation details of the centrality metrics on two classes of shared memory systems: symmetric multi-processors and multi-threaded architectures. Both of them have an high memory bandwidth and an uniform memory access. Moreover, a cache-friendly adjacency array representation (Park et al., 2002) for internally storing the graph is used.

4.1.1 Degree centrality

The degree centrality is designed to detect nodes with a higher number of adjacent edges (higher degree) and reflects the popularity and relational activity of an actor (Freeman, 1979). Degree centrality is measured as the degree of the given node v:

$$C_D(v) = \frac{\deg(v)}{n-1},$$

where deg(v) denotes the degree of a vertex v.

For directed graph, two measures of degree centrality are usually defined: in-degree and out-degree centrality.

Bader and Madduri (2006) proposed to take advantage of memory characteristics of the high performance computer they employed, storing both the in-degree and out-degree of each node in contiguous arrays during construction of

the graph. This makes the computation of degree centrality straightforward, with a constant time look-up on both the computing systems they referred to.

4.1.2 Closeness centrality

Closeness centrality measurement is based on the geodesic distance $d(v,u)$, that is, the minimum length of the path from v to u (Freeman, 1979). Closeness centrality for a node v is defined as:

$$C_C(v) = \frac{1}{\sum_{u \in V} d(v,u)}.$$

It indicates the actor's availability, so it reveals the capacity of a node to be reached.

An approximation of this index can be obtained choosing randomly k sample vertices and computing Breadth-First Search (BFS)[3], for unweighted graphs, or Single-Source Shortest Paths (SSSP)[4], for weighted graphs, from each sample vertex to all other vertices.

The estimated centrality of a vertex is defined in terms of the average distance to the sample vertices as:

$$CC_a(v) = \frac{k}{n\sum_{i=1}^{k} d(v,u)}.$$

If the number k of sample vertices is set to $\Theta(\log n/\varepsilon^2)$, the approximate closeness centrality value can be calculated in $O(\log n/\varepsilon^2(n\log n+m))$ time within an additive error of $\varepsilon\Delta$ with high probability (Eppstein and Wang, 2004).

Bader and Madduri (2006) proposed a parallel algorithm based on this approach. The parallelization is achieved by exploiting data parallelism. Each of p processors runs SSSP (or BFS) computations for k/p vertices and store the evaluated distance values. The approximate closeness centrality value of each vertex can be calculated in $O(k)=O(\log n/\varepsilon^2)$ time, and the summation for all n vertices would require $O(n\log n/p\varepsilon^2)$.

[3] BFS is a graph search algorithm that systematically explores the edges of graph to discover every vertex that is reachable from vertex v. It computes the distance (smallest number of edges) from v to each reachable vertex.

[4] SSSP find shortest paths from a source vertex v to all other vertices in the graph

4.1.3 Betweenness centrality

The betweenness centrality of a vertex *v* is defined to be the fraction of shortest paths between pairs of vertices in a network that pass through *v* (Anthonisse, 1971; Freeman, 1979). It is defined as follows:

$$BC(v) = \sum_{s,t \in V, s \neq v, t \neq v} \delta_{st}(v) = \sum_{s,t \in V, s \neq v, t \neq v} g_{st}(v) / g_{st} ,$$

where $\delta_{st}(v)$ is the pairwise dependency, that is the ratio between the number $g_{st}(v)$ of shortest paths between *s* and *t* that contain *v*, and the number g_{st} of shortest paths between nodes *s* and *t*.

This measurement represents the actor's capability to influence or control interaction between actors it links. It measures the control a vertex has over communication in the network, and can be used to identify critical nodes in the network. High centrality index means that a vertex reaches other vertices on relatively short paths, or that a vertex lies on a considerable fraction of shortest paths connecting pairs of other vertices.

Bader and Maddurri (2006) proposed a parallel algorithm for computing a betweenness centrality index based on a sequential algorithm presented by Brandes (2001). Brandes' procedure requires the definition of a set of predecessors of a vertex v on shortest paths from s as:

$$P_s(v) = \left\{ u \in V : \{u, v\} \in E, d(s,v) = d(s,u) + \omega(u,v) \right\}.$$

The sequential algorithm performs *n* BFS from each $s \in V$; these searches make it possible to create the predecessor set and to record the number of shortest paths through a vertex *w* such that $v \in P_s(w)$.

Next, for every $s \in V$, both the number of shortest paths through a vertex *w* stored in *g(w)* and predecessor sets along the paths are used to compute

$$\delta_s(v) = \sum_{t \in V} \delta_{st}(v) = \sum_{w: v \in pred(s,w)} \frac{g_{sv}}{g_{sw}} (1 + \delta_s(w)) \quad \forall v \in V .$$

In the end, betweenness centrality is obtained as:

$$BC(v) = \sum_{s \in V, s \neq v} \delta_s(v) .$$

This algorithm computes the betweenness centrality index for all vertices in the graph in $O(mn+n^2\log n)$ time for weighted graphs, and $O(mn)$ time for unweighted graphs.

The parallelism of this algorithm can be exploited at three levels of granularity[5]:

- Coarse-grained: the computation starting from each source vertex can be considered as a task; the algorithm needs n tasks to compute partial values, which can proceed in parallel. If p processors are used, p copies of data structures are required. In a real world, this space usage for a large scale graph easily exceeds the available physical memory on conventional parallel computers.
- Medium-grained: the BFS explores all neighbors of each vertex. One exploration of a vertex can be considered as one task, thus, all tasks could proceed totally in parallel if there was no shared neighbor between any two vertices. Otherwise, memory access conflicts occur and a synchronization mechanism is required to exploit this granularity of parallelism.
- Fine-grained: the task of exploring the neighbors of a vertex itself can also be parallelized. The amount of available parallelism depends on the degree of a vertex.

In the work of Bader and Maddurri (2006), the parallel algorithms exploit either medium-grained or fine-grained parallelism. For the symmetric multi-processors implementation, they assigned a fraction of the vertices from which to initiate BFS (or SSSP) computations to each processor. The authors noted that the approach used for symmetric multi-processors system is not efficient for multi-threaded system so they proposed a finer partition of the work in order to saturate all the hardware threads. They parallelized the actual BFS (or SSSP) computation, and also have a medium-grained partition at the outer level. The computational complexity is at least of $O(nm/p)$ for unweighted and weighted graphs and $O[(nm+n2\log n)/p]$ for weighted graphs. These bounds can be reached only if the degree of each vertex is 1.

The parallel algorithm for betweenness centrality can be used also for calculating the stress centrality, a centrality metric based on shortest paths counts, first presented by Shimbel (1953).

[5] Applications are often classified according to how often their subtasks need to synchronize or communicate with each other. An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; it exhibits coarse-grained parallelism if they do not communicate many times per second

4.2 Maximal clique

Many practical problems such as detection of social hierarchies, genome mapping, gene expression analysis, require the enumeration of all maximal cliques in a graph (i.e. to solve the problem of maximal clique enumeration (MCE)). By a result of Moon and Moser (1965), any *n*-vertex graph can have at most $3^{n/3}$ maximal cliques. Enumerating all maximal cliques in networks is an NP-hard problem (Lawier, 1980), that is, its computational cost is, at least, of exponential order with respect to the number of vertices.

Zhang et al. (2005) proposed a parallel algorithm (pClique) to enumerate all maximal cliques in a graph. This algorithm is based on the work of Kose et al. (2001) where a serial algorithm (KOSE) that uses BFS strategy is proposed and makes it possible to find all maximal cliques in an order with non-decreasing size *k*. Thereby, maximal cliques with size *k* are generated from cliques of size (*k-1*). However, KOSE algorithm requires storing all k-cliques and (k+1)-cliques, and this needs an enormous amount of memory.

The parallel technique is based on the fact that the generation of (k+1)-cliques from a k-clique sub-list is independent of any other k-clique sub-lists. So, starting from different k-clique sub-lists, each processor can enumerate its own (k+1)-cliques independently. Since the number of cliques generated could vary from one sub-list to another, this algorithm needs load balancing to pass some work from heavy loaded processor to light loaded ones. The pClique algorithm proceeds as follows: at each step a list of k-cliques is divided and distributed among processors which enumerate (k+1)-cliques. When all processors finish their work, results from each processors are collected, the updated list of cliques is again divided and redistributed among processors. The procedure ends when no candidate (k+1)-cliques are generated by any processors. Since all processors work independently, this algorithm does not require any communication when processors perform clique enumeration operation but the BFS strategy used makes the algorithm memory-intensive as well as the sequential one. This affects both size of graphs that can be handled by the algorithm and the parallel algorithm performance (see experimental results in the work of Zhang et al. (2005)).

Another parallel MCE algorithm is the "Parallel Enumeration of All Maximal Cliques" (Peamc) algorithm presented by Du et al. (2006). This algorithm is based on a serial MCE procedure described by the same authors. This procedure, based on Depth-first search (DFS)6, generates a disjoint set of maximal cliques for every

[6] DFS is an algorithm for traversing a graph starting from a root item (a selected node of the graph) and traveling through the edges. Algorithm starts at a specific vertex, which becomes current vertex. Then algorithm traverse graph choosing a vertex adjacent to the current vertex to visit next. If all adjacent vertices have already been discovered, or there are no adjacent vertices, then the algorithm backtracks to the last vertex that had undiscovered neighbors. Once all reachable vertices have been visited, the algorithm selects from any remaining undiscovered vertices and continues the traversal. The algorithm finishes when all vertices have been visited.

vertex in a given graph. Peamc uses a simple parallelization scheme that simply assigns each parallel process a disjoint set of vertices and allows each process to enumerate every maximal clique in the set of maximal cliques assigned to it by the serial algorithm. The Peamc algorithm does not have the same memory issues of pClique algorithm but since there is not a load balancing step after the initial work distribution, certain processes would be allowed to terminate early.

To overcome this limitation, Schmidt et al. (2009) presented a parallel, scalable, and memory-efficient MCE algorithm for distributed and/or shared memory high performance computing architectures. The proposed algorithm represents a parallelization of the MCE method of Bron and Kerbosch (1973) (BK) that enumerates maximal cliques by using a backtracking search when visiting all vertices in a maximal clique. The search paths in BK are expanded into a search tree structure. A search path can be extended in BK by visiting an unexplored vertex that is connected to every vertex already explored by the search path. Thus, the explored vertices form a clique in the graph. If a search path cannot be expanded, then the representative clique is a maximal clique. Schmidt et al. (2009) proposed an effective decomposition of the BK search tree into independent search sub-trees whose leaf nodes represent maximal cliques. The algorithm is structured as follows. Initially, it reads in the input graph data. Since the subtask of generating a sub-tree in the BK search tree requires that a processor is able to check the adjacency of nodes in the input graph, the algorithm distributes the vertex adjacency list to every processor. Once the graph data has been distributed, all candidate path structures that represent cliques of size 1 are generated and distributed to the computing elements which can begin to independently generate maximal cliques according to the BK method of generating maximal cliques. Processors continue to generate maximal cliques independently until one of them finishes exploring all of the sub-trees of the BK search for which it had been assigned. At this point the idle processor requests for more sub-trees to explore from another randomly chosen processor. This algorithm makes it possible to solve the problem MCE also for large graphs in a time that scale linearly with the number of processors used.

4.2 Problem of triangles

The triangle is one of the most basic structures in complex network analysis, and the problems of deciding if a given graph contains a triangle as well as counting the number of triangles in the graph, and listing all of them recently gained much practical importance.

In particular, counting the total number of triangles is required to check two properties of a network: *clustering* and *transitivity*. Clustering, measured by means of the clustering coefficient proposed by Watts and Strogatz (1998), is a measure of degree to which nodes in a graph tend to cluster together, while transitivity,

quantified by the transitivity ratio (Newman et al., 2002), measures the probability that two neighbors of a vertex are connected.

Both these indices are based on the total number of triangles and are widely used to understand the structure, dynamics and evolution of real-world networks. Owing to the large dimension of this network, straightforward and even approximate counting algorithms can be slow. Moreover, the asymptotically fastest existing methods for counting triangles (lowest time complexity) (Alon et al., 1997; Itai and Rodeh, 1978) suffer from space complexity. Specifically, for a network with n nodes, they present $O(n^2)$ space complexity. Therefore, in practice, instead of counting exactly the triangles, it is preferred to list the triangles (Latapy, 2008) or to use streaming (Bar-Yosseff et al., 2002; Buriol et al. 2006) and semi-streaming models (Becchetti et al., 2008).

Tsourakakis (2008) presented a new method for counting triangles approximately in large, real-world networks. In this work the author presented the Eigen-Triangle algorithm for counting the total number of triangles in a graph, and the EigenTriangleLocal algorithm that returns the count of triangles that contain a desired node. Both algorithms use a link between the number of triangles and the eigenvalues of the adjacency matrix of the graph (the number of triangles is proportional to the sum of the cubes of eigenvalues) and the observation that just the top eigenvalues contribute significantly to the total number of the triangles. The algorithms are based on Lanczos method (Golub and Van Loan, 1989) method which is a well studied projection method for solving the symmetric eigenvalue problem using Krylov subspaces. These algorithms are not only fast, but, most of them have been parallelized, or can be easily parallelized.

5 Stochastic model for social networks

Graph models used in social network analysis may describe different issues, the temporal dynamics of the social processes or the probabilistic structure of the social ties, but each of them represents the idealization and simplification of the actual processes. The exponential random graph models for social networks are the stochastic models for graphs and attempt to represent the stochastic mechanisms that produce ties, and the complex dependencies this induces. This class of models forms a statistical exponential family and it has been referred to as the p* class of models (Holland and Leinhardt, 1981; Wasserman and Pattison, 1996). The Markov random graph models (Frank and Strauss, 1986) are a particular sub-class of exponential random graph models in which a possible tie from i to j is assumed conditionally dependent only on other possible ties involving i and/or j.

A social network can be represented also by $n \times n$ adjacency matrix, Y, which with its elements is assumed for following analysis as random variables. We denote by Y_{ij} a network tie variable where $Y_{ij} = 1$ if there is a network tie from actor i to actor j, and where $Y_{ij} = 0$ if there is no tie. We specify y_{ij} as the observed

value of the variable Y_{ij}. We let Y be the matrix of all variables Y_{ij} and y the observed matrix with elements y_{ij}.

The Exponential Random Graph Model (ERGM) is defined by

$$P_{\theta,\gamma}(\mathrm{Y} = y) = \frac{\exp\left\{\theta^t u(y)\right\}}{\kappa(\theta,\gamma)}, \tag{5.1}$$

where:

- Y is the adjacency matrix of random network on n nodes,
- γ is the support of Y, the set of all possible networks with n nodes,
- θ is a vector of parameters,
- $u(y)$ is a known q-vector of graph statistics on y,
- $\kappa(\theta,\gamma)$ is the normalizing factor.

In this model, the θ parameters are unknown parameters that we want to estimate. The goal in defining the vector of statistics $u(y)$ is to choose statistics that summarize the social structure of the network (e.g. number of edges, number of triangles). These statistics should match the purpose for which networks are being simulated and model parameters are being estimated. The denominator $\kappa(\theta,\gamma)$ depends on both θ and the support γ and it is defined as:

$$\kappa(\theta,\gamma) = \sum_{z\in\gamma} \exp\left\{\theta^t u(z)\right\}. \tag{5.2}$$

The dependence of $\kappa(\theta,\gamma)$ on the unknown parameter vector carries the primary barrier to inference when using this model. It makes difficult to obtain the maximum likelihood estimation for an exponential family random graph model.

Until recently, inference for ERG models has been almost exclusively based on an alternative local approximation to the likelihood function referred to as the pseudo-likelihood (Strauss and Ikeda, 1990). The computational tractability of the pseudo-likelihood function makes it an attractive alternative to the full likelihood function. Since this approach assumes conditional independence of the random variables representing the relational ties (that is, Y_{ij} and Y_{kl} are independent whenever $i \neq k$ and $j \neq l$), it gives reasonable results only for dyadic independence models which for non-directed networks are defined as those in which $P(Y_{ij}=y_{ij})$ is independent of $P(Y_{kl}=y_{kl})$ $\forall\,(i,j)\neq(k,l)$ conditional on the actor attributes. Such models typically consist of an edge term and set of terms counting the number of instances of edges among actors with different attribute combinations. In this case the Maximum Pseudo-Likelihood (MPLE) estimators correspond to the exact solution and the true maximum likelihood estimator may be found via an MPLE computation. For dyadic dependence models statistical properties for MPLE

estimators are not well understood and in practice MPLE does not provide a good performance (van Duijn et al., 2009).

Currently the favored methods for statistical inference are Markov Chain Monte Carlo (MCMC) Maximum Likelihood Estimate (MLE) (Geyer and Thompson, 1992) and an MCMC implementation of the Robbins-Monro algorithm (Snijders, 2001; Snijders, 2002), both of which rely on the properties of the method of moments for exponential family distributions. The first method, proposed for approximating MLEs in exponential families, was used for parameter estimation in exponential random graph models by Corander et al.(1998).

In the following analysis we will focus on the Robbins-Monro algorithm. This method is a stochastic iterative algorithm and can be used to compute moment estimates, and therefore also maximum likelihood estimates in the exponential random graph model. The algorithm proposed by Snijders (2002) distinguishes three phases. In phases 1 and 3 a generation of networks is required by simulating random draws from the exponential random graph distribution with parameters that depend on the algorithm's phase. In phase 1 generated networks are used to determine a diagonal matrix $D_0 = diag(\mathrm{cov}_{\theta_0}(u(Y)))$ to be used in the successive phase. Its diagonal elements are estimates of the derivatives $d_{kk} = \partial E_\theta u_k(Y) / \partial \theta_k$, k = 1,2,..., q evaluated in the initial value θ_0 of the estimation algorithm, the method usually used to choose θ_0 is pseudolikelihood estimation method. In phase 3 the estimate covariance matrix $\Sigma(\theta) = \mathrm{cov}(u(Y))$ of $u(Y)$ is used to estimate the standard error of the model parameters estimates. The phase 2 is the most important one which consists of several sub-phases. The main goal of this phase is to determine iteratively the estimates parameter, according to the updating step:

$$\hat{\theta}^{(t+1)} = \hat{\theta}^{(t)} - a_t D_0^{-1} Z(t), \tag{5.3}$$

where a_t is the step size, D_0 is the diagonal matrix computed in phase 1, and $Z(t)$ for $t=1,2,...$ are random variables so that the conditional distribution of $Z(t)$ given $Z(1),...,Z(t-1)$ is the distribution of Z_θ (a random variable with probability distribution governed by parameter θ) obtained for $\theta = \hat{\theta}^{(t)}$. Z_θ is given by $Z_\theta = u(Y) - u_0$ where $u_0=u(y)$ is the observed value of statistics and Y has probability distribution (5.1) with parameter θ.

6. A new parallel algorithm for estimation ERGM parameter

In June 2009 we presented at the Workshop ARS'09 the state of art of parallel computing for ERGM (Marino and Stawinoga, 2009). We showed how parallel

computing could be useful for ERGM. We distinguished two kinds of problems where parallel computing could be used:

- for parameter estimation
- for network simulation.

During our studies of different algorithms for parameter estimation we realized that in the Robbins-Monro algorithm there are two moments (phase 1 and phase 3) where the generation of number of independent networks is required and so, parallelism could be effective. The parallelization of phase 2 has to be further investigated owing to the presence of MCMC simulations and inherently sequential nature of the updating step (5.3).

Recently, Ripley and Snijders (2010) introduced in the SIENA v.4 package the possibility to perform all three phases of parameters estimation procedure in parallel. However, it has to be noted that since Siena v.4 carries out the statistical estimation of models for the evolution of social networks according to the dynamic actor-oriented model of Snijders (2001), the Robbins-Monro algorithm solves a slightly different problem which does not require MCMC simulations. So, parallelization of all three phases can be achieved using straightforward simple multiple simulations.

In the following analysis we present our parallel approach and some results obtained by implementing a parallel algorithm.

The natural strategy for parallelizing phases 1 and 3 of the Robbins-Monro algorithm is to distribute the generation of networks among processors; processors work concurrently to generate different networks under the same parameters configuration and to compute a partial estimates of the diagonal matrix D_0 in phase 1 and of the covariance matrix $\Sigma(\theta)$ of statistics $u(Y)$ in phase 3 that are afterwards combined to obtain the estimate of overall matrices. Communication among processors is limited to the initialization phase, where basic common information is exchanged, and the final phase, when partial results are combined to compute overall results of both phases. Therefore, this algorithm can be generally considered as "naturally parallel". However, it is worth noting that phase 1 requires the generation of a small number of networks (usually, $7+3q$, where q is the number of statistics used in the model) while in phase 3 a greater number of generated networks is required; in the work of Snijders (2002) it is suggested to generate 1000 of them.

6.1 Computational results

In order to evaluate the effectiveness of such a parallel approach we implemented our algorithm on an IBM Bladecenter. It consists of 6 Blade LS 21, each one equipped with 2 AMD Opteron 2210 HE, with 4 GB DDR2 RAM. Each Blade is a Non-Uniform Memory Access (NUMA) architecture. The Blades are connected via

a Gigabit Ethernet Nortel layer 2/3 switch, integrated into the Bladecenter. The operating system installed on the system is Linux Red Hat Enterprise ES, release 4. Our algorithm is written in R language using functions of R environment and of the statnet package (Handcock et al., 2003); algorithm communication (message passing) relies on RMPI (Yu, 2010).

To evaluate the performance of the proposed parallel algorithm we developed, we use parallel efficiency (Kumar et al., 1994) defined as:

$$E(N,P) = \frac{1}{P}\frac{T_{seq}(N)}{T(N,P)}, \tag{6.1}$$

where $T(N,P)$ is the runtime of the parallel algorithm, and $T_{\text{seq}}(N)$ is the sequential runtime of the Robbins-Monro algorithm.

The data used for our work are the network data from package statnet for R. We chose 4 networks different in size and density. The data "flo" represents a data set of Padgett (1994), consisting of weddings among leading Renaissance Florentine families; it is a directed network with 16 nodes (families) and 20 edges (marital ties); the network's density is 0.167. The data "coleman" is a directed, unvalued 73-node network with 202 edges. This data derives from Coleman reports research (Coleman, 1964) on self-reported friendship ties among 73 boys in a small high school in Illinois over spring time in the 1957-1958 academic year; the density of network is 0.077. The networks "faux.mesa.high" and "faux.magnolia.high" (Resnick et al., 1997) represent simulations of an in-school friendship network with 205 nodes (students), 203 undirected edges (mutual friendships) and 1461 nodes (students), 974 undirected edges (mutual friendship), respectively. The densities of these two networks are 0.0097 and 0.0009.
We decided to test our algorithm only on undirected networks so we transformed "flo" and "coleman" to meet this characteristic using R function for this purpose.

Moreover, for our analysis we chose three models:

- model that includes four statistics: number of edges, the geometrically weighted edgewise shared partner (GWESP)
- model that includes four statistics: number of edges, the geometrically weighted edgewise shared partner (GWESP) and the geometrically weighted degree (GWD)
- model that includes four statistics: number of edges, the geometrically weighted edgewise shared partner (GWESP), the geometrically weighted degree (GWD) and the geometrically weighted dyadwise shared partner (GWDSP).

The geometrically weighted edgewise shared partner (GWESP), the geometrically weighted degree (GWD) and the geometrically weighted dyadwise shared partner (GWDSP) represent statistics that are useful for avoiding

degeneracy and for capturing high-order of dependency structures in networks (Hunter, 2007). The terms GWESP, GWDSP are equivalent to the alternating k-triangle and the alternating k-star. The GWD shares a similar relationship to the alternating k-two path, proposed by Snijders et al. (2006).

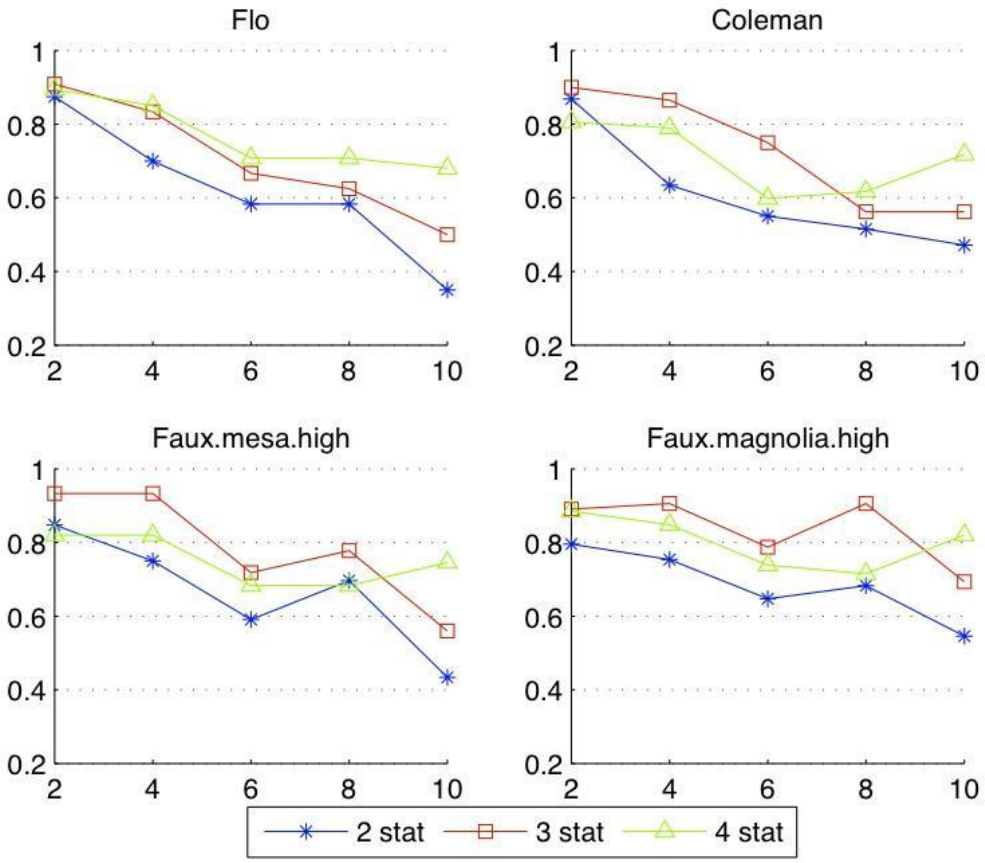

Figure 1: Efficiency (on vertical axis) of phase 1 for the four networks considered for a varying number of processors (on horizontal axis) and number of statistics.

In Figure 1 we report the efficiency of phase 1 when varying the number of processors from 2 to 10 and the number of statistics for the four different networks described above. We note that efficiency generally tends to decrease as the number of processors increase especially for lower dimension networks. This is because this phase requires generation of a small number of networks, so, as the number of processors grows (and number of networks that each processor have to generate decreases), the communication phase increasingly affects the total execution time.

In Figure 2 we report the efficiency of phase 3 when varying the number of processors from 2 to 10 and the number of statistics for the four different networks. For this phase, efficiency is always greater than 0.7 and shows an increasing tendency with processors number. We observed that time required for

phase 3, for the networks we considered, is almost 50% of the total computing time.

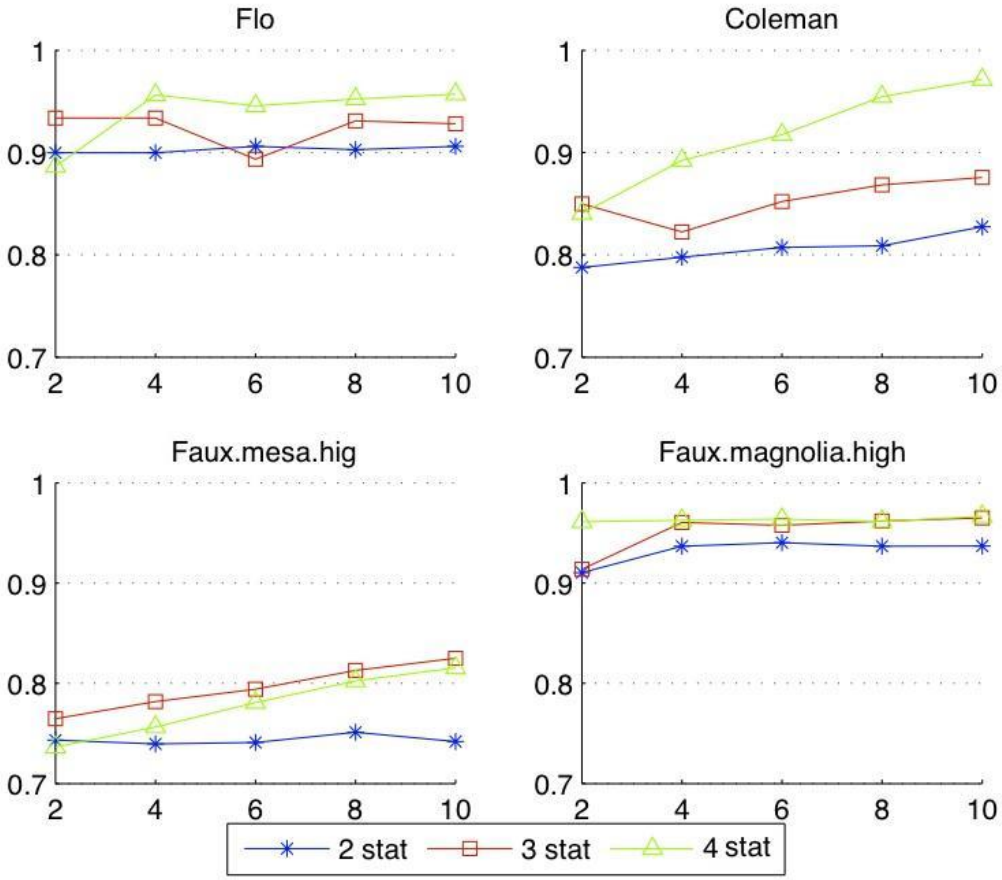

Figure 2: Efficiency (on vertical axis) of phase 3 for the four networks considered for a varying number of processors (on horizontal axis) and number of statistics.

Finally, in Figure 3 we show comparisons of phase 3 efficiency values among the 4 different networks for a different number of statistics. From these comparisons we observe that, even if phase 3 perform well on parallel computer for all the considered networks, networks for which reduction of computational time is maximum are the larger ones and the ones with greater density.

7 Conclusions and future work

The use of parallel computing in social network analysis becomes essential as networks size (and density) increase. In this paper, after a review of some parallel algorithms developed to address computational problems in evaluating structural properties of networks, we focus our attention on exploiting parallelism when estimating parameters of an exponential random graph model. In particular we introduce parallelism in phases 1 and 3 of the Robbins-Monro algorithm. Results

obtained in terms of parallel efficiency show that parallelization is not really effective in phase 1 except for larger networks, while in phase 3 it makes it possible to reduce computational time by a factor almost equal to the number of processors used. Parallelism of the Robbins-Monro algorithm has to be further investigated in order to reduce also the computation time of phase 2; this phase is rather more difficult to parallelize because of the presence of MCMC and inherently sequential nature of the updating step.

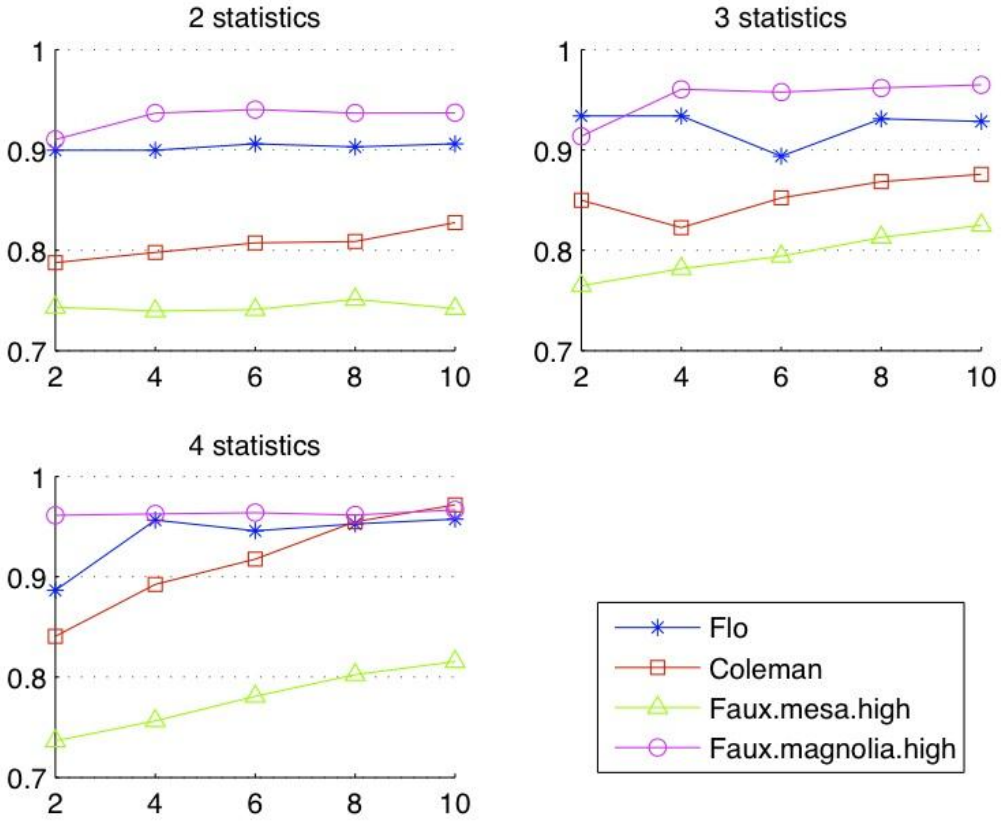

Figure 3: Efficiency (on vertical axis) of phase 3 for the three different numbers of statistics considered for a varying number of processors (on horizontal axis) and networks.

References

[1] Alon, N., Yuster, R., and Zwick, U. (1997): Finding and counting given length cycles. *Algorithmica*, **17**(3), 209-223.

[2] Anthonisse, J.M. (1971): The rush in a directed graph. *Technical Report BN 9/71-Stichting Mathematisch Centrum-Amsterdam*.

[3] Bader, D.A. and Madduri, K. (2006): Parallel algorithms for evaluating centrality indices in real-world networks. In *Proceedings of the 35th*

International Conference on Parallel Processing (ICPP). Columbus: IEEE Computer Society.

[4] Bar-Yosseff, Z., Kumar, R., and Sivakumar, D. (2002): Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA 02: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 623-632.

[5] Becchetti, L., Boldi, P., Castillo, C., and Gionis, A. (2008): Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of ACM KDD*, Las Vegas, NV, USA.

[6] Besag, J. (1974): Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society Series B*, **36**, 192-236.

[7] Brandes, U. (2001): A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, **25**(2), 163-177.

[8] Bron, C. and Kerbosch, J. (1973): Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, **16**(9), 575-577.

[9] Buriol, L.S., Frahling, G., Leonardi, S., Marchetti-Spaccamela, A., and Sohler, C. (2006); Counting triangles in data streams. In *PODS 06: Proceedings of the Twentyfifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, New York:ACM, 253-262.

[10] Carrington, P.J., Scott, J., and Wasserman, S. (2005): *Models and Methods in Social Network Analysis*, New York: Cambridge University Press.

[11] Coleman, J.S. (1964): *Introduction to Mathematical Sociology*. New York: Free Pres.

[12] Corander, J., Dahmström, K., and Dahmström, P. (1998): Maximum likelihood estimation for Markov graphs. *Research Report 8*, Department of Statistics, University of Stockholm.

[13] Du, N., Bin, W., Liutong, X., Bai, W., and Xin, P. (2006): A parallel algorithm for enumerating all maximal cliques in complex network. In: *Proc. of ICDM Workshops*, 320-324.

[14] Eppstein, D. and Wang, J. (2004): Fast approximation of centrality. *Journal of Graph Algorithms and Applications*, **8**(1), 39-45.

[15] Frank, O. and Strauss, D. (1986): Markov graphs. *Journal of the American Statistical Association*, **81**, 832-842.

[16] Freeman, L.C. (1979): Centrality in social networks. Conceptual clarification. *Social Networks*, **1**, 215-239.

[17] Geyer, C.J. and Thompson, E.A. (1992): Constrained Monte Carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society Series B*, **54**, 657-699.

[18] Golub, G. and Van Loan, C. (1989): *Matrix Computations*. Baltimore: JohnsHopkinsPress, MD, second edition.

[19] Handcock, M.S., Hunter, D.R., Butts, C.T., Goodreau, S.M., and Morris, M. (2003): statnet: Software tools for the Statistical Modeling of Network Data. URL: http://statnetproject.org.

[20] Holland, P.W. and Leinhardt, S. (1981): An exponential family of probability distributions for directed graphs. *Journal of the American Statistical Association*, **76**(373), 33-65.

[21] Hunter, D.R. (2007): Curved exponential family models for social networks. *Social Networks*, **29**, 216-230.

[22] Itai, A. and Rodeh, M. (1978): Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4), 413-423.

[23] Kose, F., Weckwerth, W., Linke, T., and Fiehn, O. (2001): Visualizing plant metabolomic correlation networks using cliquemetabolite matrices. *Bioinformatics*, **17**(12),1198-1208.

[24] Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994): *Introduction to Parallel Computing - Design and Analysis of Algorithms.* Redwood City, CA: The Benjamin/Cummings Publishing Company.

[25] Latapy, M. (2008): Practical algorithms for triangle computations in very large (sparse (power-law)) graphs. *Theoretical Computer Science*, **407**(1-3), 458-473.

[26] Lawler, E.L., Lenstra, J.K., and Kan, A.H.G.R. (1980): Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, **9**(3), 558-565.

[27] Lumsdaine, A., Gregor, D., Hendrickson, B., and Berry, J. (2007): Challenges in parallel graph processing. *Parallel Processing Letters*, **17**(1), 5-20.

[28] Marino, M. and Stawinoga, A. (2009): Statistical methods for social networks: a focus on parallel computing. In *Book of abstracts. Workshop ARS'09 Social Network Analysis: Models and Methods for Relational Data*, 60-61.

[29] Moon, J. W. and Moser, L. (1965): On cliques in graphs. *Israel Journal of Mathematics*, **3**(1), 23-28.

[30] Newman, M.E.J., Strogatz, S.H., and Watts, D.J. (2002): Random graph models of social networks. *Proceedings of the National Academy of Science of the United States of America*, 2566-2572.

[31] Padgett, J.F. (1994): Marriage and elite structure in renaissance Florence, 1282-1500. Paper delivered to the Social Science History Association.

[32] Park, J., Penner, M., and Prasanna, V. (2002): Optimizing graph algorithms for improved cache performance. In *Proc. International Parallel and Distributed Processing Symp. (IPDPS 2002)*, Fort Lauderdale, FL.

[33] Resnick, M.D., Bearman, P.S., Blum, R.W. et al. (1997): Protecting adolescents from harm. Findings from the National Longitudinal Study on

Adolescent Health. *Journal of the American Medical Association*, **278**, 823-32.

[34] Ripley, R.M. and Snijders, T.A.B. (2010*)*: *Manual for SIENA version 4.0.* URL: http:// stat. gamma. rug. nl/ siena.html.

[35] Schmidt, M.C., Samatovaa, N.F., Thomasc, K., and Park, B-H. (2009): A scalable, parallel algorithm for maximal clique enumeration. *Journal Parallel and Distributed Computing*, **69**, 417-428.

[36] Shimbel, A. (1953): Structural parameters of communication networks. *Bulletin of Mathematical Biophysics*, **15**, 501-507.

[37] Snijders, T.A.B. (2001): The statistical evaluation of social network dynamics. *Sociological Methodology*, **31**(1), 361-395.

[38] Snijders, T.A.B. (2002): Markov chain Monte Carlo estimation of Exponential Random Graph Models. *Journal of Social Structure*, **3**(2).

[39] Snijders, T.A.B., Pattison, P., Robins, G.L., and Handcock, M.S. (2006) New specifications for Exponential Random Graph Models. *Sociological Methodology*, **36**, 99-153.

[40] Strauss, D. and Ikeda, M. (1990) Pseudolikelihood estimation for social networks. *Journal of the American Statistical Association*, **85**, 204-212.

[41] Tsourakakis, C.E. (2008): *Fast Counting of Triangles in Large Real Networks: Algorithms and Laws.* International Conference Data Mining.

[42] van Duijn, M.A.J., Gile, K., and Handcock, M.S. (2009): A framework for the comparison of maximum pseudo-likelihood and maximum likelihood estimation of exponential family random graph models. *Social Networks*, **31**, 52-62.

[43] Wasserman, S. and Pattison, P. (1996): Logit models and logistic regression for social networks: I. An introduction to Markov graphs and p*. *Psychometrika*, **61**, 401-425.

[44] Yu, H. (2010): Interface (Wrapper) to MPI (Message-Passing Interface). URL: http:// www.stats.uwo.ca/ faculty/ yu/ Rmpi.

[45] Watts, D.J. and Strogatz, S.H. (1998): Collective dynamics of smallworld networks. *Nature*, **393**, 440-442.

[46] Zhang, Y., Abu-Khzam, F., Baldwin, N., Chesler, E., Langston, M., and Samatova, N.F. (2005): Genome-scale computational approaches to memory-intensive applications in systems biology. *In Proc. of the 2005 ACM/IEEE Conference on Supercomputing*.