Parallel Computing 36 (2010) 469–485

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Parallelism in simulation and modeling of scale-free complex networks

Tomas Hruz *, Stefan Geisseler, Marcel Schöngens

Institute of Theoretical Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland

ARTICLE INFO

Article history:
Received 1 September 2009
Received in revised form 10 April 2010
Accepted 30 April 2010
Available online 11 May 2010

Keywords:
Scale-free networks
Stochastic processes
Non-growing complex networks

ABSTRACT

Evolution and structure of very large networks has attracted considerable attention in recent years. In this paper we study a possibility to simulate stochastic processes which move edges in a network leading to a scale-free structure. Scale-free networks are characterized by a "fat-tail" degree distribution with considerably higher presence of so called hubs – nodes with very high degree. To understand and predict very large networks it is important to study the possibility of parallel simulation. We consider a class of stochastic processes which keeps the number of edges in the network constant called equilibrium networks. This class is characterized by a preferential selection where the edge destinations are chosen according to a preferential function $f(k)$ which depends on the node degree k. For this class of stochastic processes we prove that it is difficult if not impossible to design an exact parallel algorithm if the function $f(k)$ is monotonous with an injective derivative. However, in the important case where $f(k)$ is linear we present a fully scalable algorithm with almost linear speedup. The experimental results confirm the linear scalability on a large processor cluster.

1. Introduction

The study of complex networks has attracted considerable attention in recent years. The research community as well as the public has became sensitive to the fact that various sorts of networks have a profound effect on our lives. We understand today that our brains, our communication lines, our highways, our flight connections and our social contacts (just to mention a few examples) exhibit a structure of a complex network with nodes representing the entities and edges representing some sort of interactions among them. Proliferation of networks has led to a deeper research showing that the behavior and the growth of such networks are far from being purely random. It often follows certain topological and structural patterns discovered in the theory of scale-free networks and small-worlds [1,10,18,21,7]. Scale-free networks are characterized by a "fat-tail" degree distribution with considerably higher presence of so called hubs – nodes with very high degree. Exactly speaking only the degree distribution $P(k) = \alpha k^{-\gamma}$, where $P(k)$ denotes the probability of finding a vertex with degree k in the network, is invariant under scaling. However, all networks having a log–log near linear degree distribution for higher degree k are studied under the term scale-free networks.

Stochastic processes are often used to model the evolution of complex networks. Such processes consist of simple phenomenological rules describing how the edges and nodes appear and change in the network. A master equation is a difference (or differential) equation describing the behavior of a certain network quantity under the given stochastic process. The basic quantity characterizing the complex network is the degree distribution. Degree distribution $P(k,t)$ is the probability that at any given time t a vertex chosen uniformly at random would have degree equal to k.

* Corresponding author.
E-mail address: tomas.hruz@inf.ethz.ch (T. Hruz).

0167-8191/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2010.04.004

However, even the simplest edge and vertex change rules can lead to a very complicated master equation [13]. In this situation a simulation of the network stochastic process can bring valuable insight into the behavior and possible solutions of the master equation. Moreover, some real networks are so large and complicated that the only chance to setup a framework where they could be understood is to simulate them on a large-scale.

For some critical networks like internet it can be necessary in future to have an online prediction model which would follow and predict the development of the network conditions. Similarly as the weather forecast service observes, simulates and predicts the behavior of the atmosphere on earth it will be important to know the state and the future development of the network. In such applications fast parallel algorithms capable of simulating the network processes on a very large scale are needed. The network simulation systems can also play a central role in understanding and preventing security problems in large networks. Fast simulations of very large networks can show how the evolution of the underlying network structure influences the speed and other factors of a security problem spreading over the network.

To understand and predict very large networks it is important to study the possibility of parallel simulation. We concentrate on a basic building block of network evolution called preferential selection because it creates a major obstacle to parallelization of scale-free models. Preferential selection denotes a step where a network node is selected with a probability dependent on its degree. This dependency is expressed through a preference function which we discuss in detail in the next sections. It is a part of a larger step called preferential attachment, where after the node is preferentially selected an edge is attached to it which was selected during the earlier stages of the stochastic process. We prove a theorem indicating that it will be very hard (if not impossible) to find efficient and exact parallel algorithms for the general case. However, a very important special case of linear preferential attachment (for the discussion on importance of the linear case see e.g. [11]) can be simulated in parallel and we provide an algorithm which achieves almost linear speedup in the number of processors.

The principal problem, under which conditions the stochastic processes generating scale-free networks can be parallelized, was not studied in the literature. The authors in [22] consider a problem how to generate in parallel a large, scale-free network which can be used for testing purposes of simulation frameworks. The proposed method serves perfectly well for the purposes the authors in [22] consider, however the conditions on the preference function which would allow (resp. not allow) efficient parallelization are not studied. Moreover, in comparison to the method presented here, the PBA method from [22] can be used only for one particular preference function $f(k) = k$, because selecting an edge uniformly at random and choosing its vertex at random is equivalent to the preference function $f(k) = k$. Our parallel algorithm allows to consider any linear preference function. The authors in [22] also develop a method called PK which provides a deterministic construction of scale-free graphs, however the stochastic process generating the graph is not provided. We think that to identify this process a new research into equivalence between certain classes of stochastic processes and prescribed scale-free graphs would be needed. It is known that any prescribed degree distribution in a graph can be achieved under the configuration model [18].

The class of complex networks similar to our scenario which was considered from the parallelism point of view are small-world networks. Parallel algorithms for generation of small-world networks has been studied in [2,15]. Small-world networks have similar features as scale-free networks (for example small diameter) but the generating stochastic processes do not use preferential selection which is the main obstacle to the parallelization of scale-free networks as we argue below. Another field which can be related to our research is the study of parallelism for Monte Carlo methods [23]. Intuitively, a stochastic process acting on a network can be transformed to a Monte Carlo setting, because the behavior of very large networks can be continuously approximated with a partial differential equation [10], where the degree plays a role of spatial coordinate and discrete time is approximated with continuous time coordinate. However, to understand this relation and its consequences to parallel simulation of scale-free networks, a new branch of research would be needed.

The stochastic process which we study in this paper provides a simplification of the processes occurring in real complex networks. The complex network community has proposed more complex processes to model finer phenomena (apart from scale-free degree distribution) occurring in real networks [5,8]. However, for such stochastic processes a very limited body of theory exists which would provide a sufficient theoretical analysis (mean-field model) of their behavior not to say an analysis of their parallelization possibilities. Most studies provide simulation results of proposed models which often illustrate that the model can capture the features of real complex networks. Moreover, in many proposed processes, preferential attachment is used as a building block. This is also true for emergent studies of features like network conductivity [17] where the preferential attachment plays a role of an important building block which is combined with other stochastic rules to obtain a conductivity modeling for internet graphs. Therefore, a deeper study of the preferential attachment provided in our paper can serve as a starting point to consider more complex scenarios.

Generic simulation systems like [9] can be successfully built on an idea that in many situations even if the preferential selection is used, approximate solutions are possible. This is for example the case of sparse networks where our theory does not provide a good prediction. Another such case is discussed in the concluding section where we propose to use the linear approximation of weak non-linearities of preferential function. However, our analysis in Section 3 shows that in dense graphs with nonlinear preferential selection if exact solution is needed the preferential selection is an obstacle for an efficient parallelism. In this case the focus should be on parallelly efficient and theoretically well understood approximations to the given stochastic process.

The present paper is organized as follows. In the next section we define the basic network generating processes and the related notation. In Section 3 we study the depth of the dependency tree for general preferential attachment, and in Section 4

we provide a parallel algorithm for the linear preferential selection. We conclude with experimental results and possible directions of further research.

2. Stochastic models of complex network evolution

To model the nodes and the relations of a complex network we consider a multigraph $G(V,E)$ without an orientation, where V is a set of nodes (vertices) and E is a set of edges. The number of nodes is denoted with N, $|V| = N$ and the number of edges with L, $|E| = L$. The basic quantity describing the network evolution is the degree distribution defined as $P(k) = N(k)/N$, where $N(k)$ denotes the number of nodes having degree k. The averaging of a quantity X is denoted with $\langle X\rangle$ or with $\overline{X}$. Specifically, the average degree of the network is denoted with $\bar{k}$, and it equals $\bar{k} = 2L/N$. In the cases where we consider changes of the quantities like $P(k)$ or $N(k)$ over time, we add the parameter t as in $N(k,t)$ or $P(k,t)$.

The complex network theory recognizes two principal cases with respect to the evolution of the number of edges. First, there are non-growing or equilibrium networks where the number of edges L is constant or bounded within some interval. Second, growing or non-equilibrium networks are studied, where during the network evolution the number of nodes and the number of edges grows substantially and it is not bounded by any constant.

In Process 1 we define a widely studied basic equilibrium process [10,11,13]. We call the process illustrated in Fig. 1 "Simple Edge Selection Process" (SESP). One repetition of the process steps called a process loop models one discrete time unit. As an initial condition we generally suppose a multigraph $G(V,E)$ with L edges and N vertices.

Process 1: SESP

Require: a multigraph $G(V,E)$ with L edges and N vertices.
1: $N_s \Leftarrow$ *StepLimit* {Initialize the number of process loops.}
2: **while** number of process loops smaller than N_s **do**
3: An edge, denoted E_i, is selected uniformly at random.
4: An end vertex, denoted v_i, of E_i is selected uniformly at random. The other end vertex will be denoted as v_j.
5: A vertex, denoted v_l, is selected with a probability proportional to $f(k)$ i.e. with probability $f(k)/(N\langle f\rangle)$ where k is the degree of v_l.
6: The edge E_i is rewired from v_i to v_l i.e. the edge E_i between v_i and v_j is deleted and a new edge between v_l and v_j is created.
7: **end while**

Non-growing complex networks have been observed in many situations. A typical example is represented by metabolic networks in living organisms which consist of complex chains of biochemical reactions in the living cell [20,14]. The number of reactions and substances flowing through the network is constant, however the network is stochastically changing its topology as a reaction to the changes of the organism state. Another example are neural networks in the brain where the activity of the connections between neurons are stochastically reconfigured on the fly to reflect the current processing needs. An overview of further examples together with modeling theory can be found in [19]. Generally, most of the growing networks (like for example internet and WWW) will reach a saturation phase because of constraints on resources and energy which are present in any physical system. Then the stochastic reconfiguration processes will prevail over the growth processes.

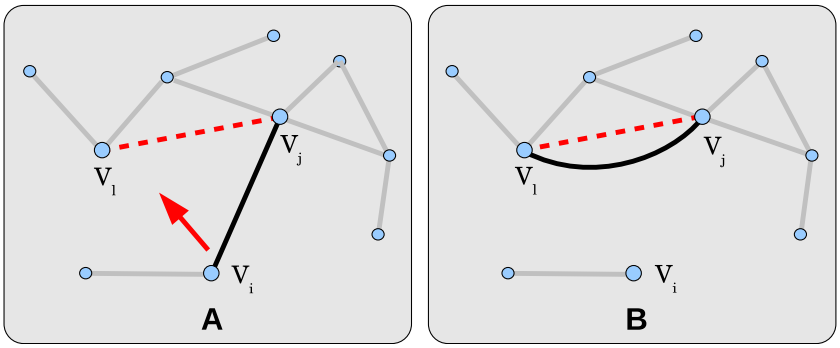

Fig. 1. Illustration of the "Simple Edge Selection Process" (SESP - see Process 1). The process selects an edge uniformly at random (Picture A) and rewires it to a node which was preferentially selected (Picture B). A detailed explanation of preferential selection and the process definition is provided in Section 2. The rewiring is unconditional, this means that there is no test of the connectivity state between v_l and v_j. During the rewiring operation the following three cases can occur: (i) there is no edge between the node v_l and the node v_j, therefore the process does not create a multiple edge or a self loop, (ii) there are already one or more edges between v_l and v_j, the process adds another edge between the nodes, and (iii) if $v_l = v_j$, the process creates a self-loop.

Step 5 in Process 1 is called preferential selection, and $f(k)$, $k \geqslant 0$ is a preference function. This is a basic building block of processes generating scale-free distributions, which are the subjects of study in the theory of complex networks. During the preferential selection a node is chosen in dependence of its degree. If the function $f(k)$ is increasing, the nodes having more edges are chosen with higher probability. For example in the case of Internet Web pages the authors are linking pages of their web sites to other pages already well known to them and these are exactly the web pages having already a lot of links. This sort of preferential selection is often modeled with the linear preference function $f(k) = \beta k + k_0$. On the other hand, to consider non-linearities in $f(k)$ will be necessary if we want to have more accurate models of real networks, because there are natural constrains (on energy and other resources) which must lead to saturation effects. We denote as $\langle f \rangle$ the mean value of $f(k)$ which is generally dependent on time $\langle f \rangle(t) = \sum_{s=0}^{2L} f(s)P(s,t)$.

The preferential selection is difficult to parallelize because it depends on global information, namely on the development of the degree distribution. Closer inspection of the preferential selection suggests that a processing unit needs the whole information about the degree distribution whenever it wants to compute the selection probability $s(k) = f(k)/(N\langle f \rangle)$. Indeed, we can prove in Section 3 that this is the case for a considerable range of scale-free networks with nonlinear preference function. Our idea of a parallel algorithm for linear preference is based on the observation that we can replace the step dependent on the degree distribution with steps where objects are chosen uniformly at random. This makes the process less dependent on the information that can not be distributed cheaply across the processing units. On the other hand, Steps 3 and 4 of Process 1 have a different character, independent of the degree distribution, therefore they can be better parallelized. For example, the edges can be distributed between the processors, and a randomly chosen processor can uniformly at random choose an edge from the set of its local edges.

The master equation describes the time evolution of the degree distribution in the mean-field approximation [4]. It is a deterministic finite difference equation (or in some cases partial differential equation) which describes the development of the degree distribution mean value. For Process 1 the master equation [13,10] can be formulated as

$$P(k,t+1) = P(k,t) - \frac{f(k)}{N\langle f \rangle(t)}P(k,t) + \frac{f(k-1)}{N\langle f \rangle(t)}P(k-1,t) - \frac{k}{N\bar{k}}P(k,t) + \frac{k+1}{N\bar{k}}P(k+1,t) + O(1/N^2), \tag{1}$$

where $O(1/N^2)$ denotes the terms dependent on $1/N^2$ which disappear very quickly for larger networks.

As we illustrated in Fig. 1 the SESP process can create multiple edges and self-loops, therefore we need multigraphs as an underlying model. Because in the large graph limit the frequency of such artifacts tends to zero [10], the multigraph setting is standardly used. On the other hand, it would be desirable to have a model allowing only simple edges and no self-loops. In [13] we investigated the constraints preserving the simple graph structure, and we have shown that the understanding of these constraints involves a whole hierarchy of object distributions, which makes the exact modeling very difficult.

The preferential selection constitutes an important building block for modeling of complex networks. This can be illustrated on another much investigated class of complex networks that describes non-equilibrium resp. growing networks [3,4]. The basic model of complex network growth is illustrated in Fig. 2. The stochastic process called "Barabasi–Albert Model" (BM) can be defined with the following steps, where the initial condition is again a multigraph $G(V,E)$ with L edges and N vertices. Marginally, it can be noted that this process does not create any new multiple edges and self-loops i.e. it preserves the simple graph property if the initial configuration is a simple graph.

Process 2: BM

Require: a multigraph $G(V,E)$ with L edges and N vertices.
1: $N_s \Leftarrow StepLimit$ {Initialize the number of process loops.}
2: **while** number of process loops smaller than N_s **do**
3: A vertex, denoted v_i, added to the existing graph G.
4: A vertex, denoted v_j, is selected with a probability proportional to $f(k)$ i.e. with probability $f(k)/(N\langle f \rangle)$ where k is the degree of v_j.
5: A new edge E_i is created from v_i to v_j.
6: **end while**

The master equation for BM can be formulated [10] as

$$N(k,t+1) = N(k,t) + \frac{k-1}{t\bar{k}}N(k-1,t) - \frac{k}{t\bar{k}}N(k,t) + \delta_{k,1}, \tag{2}$$

where t means discrete time starting at $t = 2$ and $\delta_{k,1}$ is the Kronecker delta. In the basic setting the initial condition is a graph having 2 vertices and one edge between them, i.e. $N(k,2) = 2\delta_{k,1}$.

In the rest of the paper we concentrate on the preferential selection step in the context of Process 1 (SESP), however the analysis and the parallelization we propose are also applicable for other cases where the preferential selection is used.

3. Parallelism for general preferential selection

To show that parallel simulation is difficult in certain situations we proceed in three steps. In the first part we show that if the preference function $f(k)$ is monotonous and has an injective derivative then the selection probability $s(k) = f(k)/(N\langle f \rangle)$

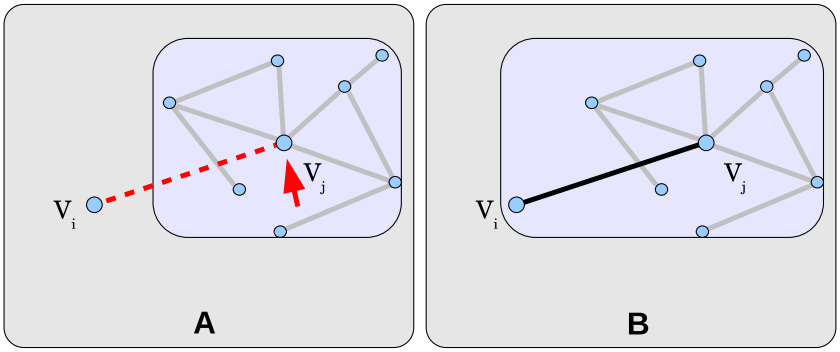

Fig. 2. Illustration of Process 2 – growing scale-free network. The process adds a new node to the existing network (Picture A) and preferentially selects a node from the existing network, which is then connected to the new node (Picture B).

changes for every node in the network if the degree distribution $P(k,t)$ changes during a discrete time unit (process loop) of Process 1. In other words, to know the selection probability the information about all nodes in the network is necessary. To obtain this fact one has to follow the $\Rightarrow$ direction of Lemma 3 and $\Rightarrow$ direction of Lemma 4. Lemmas 1 and 2 are technical tools which are valid for any preference function $f(k)$. On the other hand to obtain Lemma 3 we have to suppose that the difference (derivative) of the function $f(k)$ is injective. For Lemma 4 to be valid a monotonous function $f(k)$ is necessary.

In the second part we additionally suppose that the network distribution is bounded from above by a scale-free distribution. Under this condition we show in Theorem 6 that the degree distribution $P(k,t)$ changes with high probability in every time unit of the Process 1. To prove Theorem 6 we need Lemma 5 which is independent on any concrete degree distribution class. Now combining Lemmas 1–5 and Theorem 6 we obtain a class of networks, namely the scale-free networks with monotonous nonlinear function $f(k)$, for which the selection probability $s(k)$ changes with high probability in every time unit for all nodes in the networks.

In the third part we develop a model of parallel computing which considers dependencies between computation states based on changes in degree distribution. The idea (captured in the definition of the dependency tree) is to organize the computational states in a tree where the edges model the dependencies of the selection probability on degree distribution. We summarize our analysis in Theorem 7 about the depth of the dependency tree during the simulation of Process 1.

Additionally we use the following notation. Let $[a,b]$ denote the interval of integers n such that $a \leqslant n \leqslant b$. We denote the degree of node i in discrete time t of Process 1 by $k_i(t)$. Observe, that in each step of the process the connectivity of at most two vertices v_i and v_l is changed. By $K = \{k_i(t), k_i(t), k_i(t+1), k_l(t+1)\}$ we denote the degrees that get changed in step t and $t+1$. The following Lemma reveals an important connection between these degrees and the development in the degree distribution.

Lemma 1. *For every discrete time t of Process 1 the following equivalence holds:*

$$v_i = v_l \vee k_i(t) = k_l(t) + 1 \iff \quad \forall k \in [0, 2L] : P(k, t+1) = P(k, t).$$

Proof. First we consider $\Rightarrow$: If $v_i = v_l$, then $k_i(t) \neq k_l(t) + 1$. Since from $v_i = v_l$ it follows directly that $\forall\ k \in [0,2L]$: $P(k,t+1) = P(k,t)$, we only need to consider the case when $k_i(t) = k_l(t) + 1$. We decompose the process into two phases. Phase one is the deletion of the edge (v_j, v_i) and phase two is the adding of edge (v_j, v_l). Moreover, $N(k,t)$, $N(k,t+1)$ only differ in $k \in K$. We do not consider vertex v_j since no change is caused in any of the two phases. In the first phase v_i moves from the set of vertices with degree $k_i(t)$ to the set of vertices with degree $k_i(t) - 1 = k_l(t)$. We denote the state after phase one and before phase two by t'. We obtain

$$N(k_i(t), t') = N(k_i(t), t) - 1,$$
$$N(k_l(t), t') = N(k_l(t), t) + 1.$$

For phase two the only thing that happens is that vertex v_l moves from the set of vertices with degree $k_l(t)$ to the set of vertices with degree $k_l(t) + 1 = k_i(t)$. Then,

$$N(k_i(t), t+1) = N(k_i(t), t') + 1,$$
$$N(k_l(t), t+1) = N(k_l(t), t') - 1.$$

Therewith, we get that $N(k_i(t), t+1) = N(k_i(t), t)$ and $N(k_l(t), t+1) = N(k_l(t), t)$. From the fact that $k_i(t+1) = k_i(t) - 1 = k_l(t)$ and $k_l(t+1) = k_l(t) + 1 = k_i(t)$ we also obtain that $N(k_i(t+1), t+1) = N(k_i(t+1), t)$ and $N(k_l(t+1), t+1) = N(k_l(t+1), t)$. Since $P(k,t) = N(k,t)/N$ we have $P(k,t+1) = P(k,t)$.

Now, we consider $\Leftarrow$: Assume for the sake of contradiction, that $v_i \neq v_l \wedge k_l(t)+1 \neq k_i(t)$. Consider degree $k_i(t)$. In Phase 1 the edge (v_j, v_i) is removed, s.t. $N(k_i(t),t') = N(k_i(t),t) - 1$. In Phase 2 the edge (v_j, v_l) is added. If $v_i \neq v_l$ we know that $k_l(t+1) = k_l(t)+1$. Since $k_l(t)+1 \neq k_i(t)$ it follows that $k_l(t+1) \neq k_i(t)$. That means that v_l will not get a degree of $k_i(t)$ in discrete time step $t+1$. Further, v_l does not contribute to $N(k_i(t),t+1)$. Hence, $N(k_i(t))$ does not change in Phase 2 and we have $N(k_i(t),t+1) = N(k_i(t),t') = N(k_i(t),t) - 1$. This is a contradiction, because we can derive that $P(k_i(t),t) \neq P(k_i(t),t+1)$. □

The next lemma studies the time development of the preference function mean during the process.

Lemma 2. *Let $f(k)$, $k \geqslant 0$ be a preference function in Process 1. For any discrete time t it holds that*

$$\langle f\rangle(t+1) - \langle f\rangle(t) = \frac{1}{N}\left(f(k_i(t)-1) - f(k_i(t)) + f(k_l(t)+1) - f(k_l(t))\right).$$

Proof. By definition we obtain

$$\langle f\rangle(t+1) - \langle f\rangle(t) = \sum_{k=0}^{2L} f(k)P(k,t+1) - \sum_{k=0}^{2L} f(k)P(k,t).$$

As mentioned above, only the connectivity of v_i and v_l is changed and corresponding degrees are $K = \{k_i(t), k_l(t), k_i(t+1), k_l(t+1)\}$. Hence,

$$\langle f\rangle(t+1) - \langle f\rangle(t) = \frac{1}{N}\left(\sum_{k\in K} f(k)N(k,t+1) - \sum_{k\in K} f(k)N(k,t)\right).$$

For $N(k,t+1)$ of the first sum, we get the following identities:

$$\begin{aligned}
N(k_i(t),t+1) &= N(k_i(t),t) - 1,\\
N(k_i(t+1),t+1) &= N(k_i(t+1),t) + 1,\\
N(k_l(t),t+1) &= N(k_l(t),t) - 1,\\
N(k_l(t+1),t+1) &= N(k_l(t+1),t) + 1.
\end{aligned}$$

Therewith, the second sum can be subtracted directly, such that we obtain

$$\begin{aligned}
\langle f\rangle(t+1) - \langle f\rangle(t) &= \frac{1}{N}\left(f(k_i(t+1)) - f(k_i(t)) + f(k_l(t+1)) - f(k_l(t))\right)\\
&= \frac{1}{N}\left(f(k_i(t)-1) - f(k_i(t)) + f(k_l(t)+1) - f(k_l(t))\right),
\end{aligned}$$

which completes our proof. □

The following lemma is establishing the relation between changes in the degree distribution and the preference function f.

Lemma 3. *Let $f(k)$, $k \geqslant 0$ be a preference function in Process 1 for which the first difference $\Delta f(k) = f(k+1) - f(k)$ is injective. Then the degree distribution does not change in one discrete time unit of Process 1 iff the mean $\langle f\rangle$ of $f(k)$ does not change, i.e.:*

$$\forall k \in [0, 2L] : P(k,t+1) = P(k,t) \Longleftrightarrow \langle f\rangle(t+1) = \langle f\rangle(t). \tag{3}$$

Proof. First, we show the direction $\Rightarrow$ in (3). Given $P(k,t+1) = P(k,t)$ we have

$$\langle f\rangle(t) = \sum_{k=0}^{2L} f(k)P(k,t) = \sum_{k=0}^{2L} f(k)P(k,t+1) = \langle f\rangle(t+1).$$

For the direction $\Leftarrow$ in (3), let us suppose that $\langle f\rangle(t+1) = \langle f\rangle(t)$. If $v_i = v_l$ we can conclude that $P(k,t+1) = P(k,t)$ because the selected edge is detached from the vertex v_i and immediately reattached to the same vertex. For $v_i \neq v_l$ we have $0 = \langle f\rangle(t+1) - \langle f\rangle(t)$. By Lemma 2 the following equalities hold

$$0 = \frac{1}{N}\left(f(k_i(t)-1) - f(k_i(t)) + f(k_l(t)+1) - f(k_l(t))\right) = \frac{1}{N}\left(\Delta f(k_l(t)) - \Delta f(k_i(t)-1)\right).$$

Therefore, by injectivity of $\Delta f(k)$ we obtain:

$$0 = \Delta f(k_l(t)) - \Delta f(k_i(t)-1) \Rightarrow \Delta f(k_l(t)) = \Delta f(k_i(t)-1) \Rightarrow k_i(t) = k_l(t)+1.$$

Finally, we can apply Lemma 1 to obtain the desired result. □

The next step in understanding the obstacles in parallelization of the preferential selection is how changes in the mean $\langle f\rangle$ of the preference function influence the changes in the selection probability.

Lemma 4. *Let $f(k) \geqslant 0$, $k \geqslant 0$ be a strictly monotonic (decreasing or increasing) positive preference function in Process 1. If $\langle f\rangle$ changes in one discrete time unit of Process 1, then the selection probability $s(k) = f(k)/(N\langle f\rangle)$ changes for all vertices i.e.*

$$\forall v_i \in V : \frac{f(k_i(t+1))}{N\langle f\rangle(t+1)} \neq \frac{f(k_i(t))}{N\langle f\rangle(t)}.$$

Proof. Because we rewire only one edge from a vertex v_i to a vertex v_l in one discrete time unit of Process 1, the degree k and therefore $f(k)$ can change for at most 2 vertices v_i and v_l. For all other vertices v_m we have $k_m(t) = k_m(t+1)$ and only $\langle f\rangle$ changes. Thus, the selection probability changes and we obtain

$$\frac{f(k_m(t))}{N\langle f\rangle(t)} \neq \frac{f(k_m(t+1))}{N\langle f\rangle(t+1)}.$$

For the vertices v_i and v_l it remains to show that

$$\frac{f(k_i(t+1))}{\langle f\rangle(t+1)} \neq \frac{f(k_i(t))}{\langle f\rangle(t)} \quad \text{and} \quad \frac{f(k_l(t+1))}{\langle f\rangle(t+1)} \neq \frac{f(k_l(t))}{\langle f\rangle(t)}.$$

According to Lemma 3 the mean value $\langle f\rangle(t) = \sum_{k=0}^{2L} f(k)P(k,t)$ changes if and only if there exists a k for which $P(k,t)$ changes, therefore $v_i \neq v_l$ (in Process 1 if $v_i = v_l$ the degree distribution $P(k,t)$ does not change because the edge is reattached to the same vertex) and the graph contains at least 2 vertices.

For all vertices $v_m \in V$

$$N\langle f\rangle \geqslant f(k_m). \tag{4}$$

This can be seen as follows. Since f and $P(k,t)$ are non-negative functions it follows that

$$N\langle f\rangle = N\sum_{k=0}^{2L} f(k)P(k,t) \geqslant Nf(k_m)P(k_m,t) = f(k_m)N(k_m,t) \geqslant f(k_m),$$

because $v_m \in N(k_m,t)$.

Suppose that $f(k)$ is increasing. From the proof of Lemma 1 we know that $k_l(t+1) = k_l(t) + 1$. For vertex v_l using Lemma 2 we get

$$\frac{f(k_l(t+1))}{\langle f\rangle(t+1)} = \frac{f(k_l(t)+1)}{\langle f\rangle(t+1)} = \frac{f(k_l(t)+1)}{\langle f\rangle(t) + \frac{f(k_l(t)+1) - f(k_l(t)) + f(k_i(t)-1) - f(k_i(t))}{N}}.$$

It holds $f(k_i - 1) - f(k_i) < 0$ since f is strictly monotone increasing. Hence, we derive that

$$\frac{f(k_l(t+1))}{\langle f\rangle(t+1)} > N\frac{f(k_l(t)) + f(k_l(t)+1) - f(k_l(t))}{N\langle f\rangle(t) + f(k_l(t)+1) - f(k_l(t))} = N\frac{f(k_l(t)) + c}{N\langle f\rangle(t) + c},$$

where $c = f(k_l(t)+1) - f(k_l(t))$. Since inequality (4) $N\langle f\rangle \geqslant f(k_l)$ allows to reduce the term removing c, it follows that

$$\frac{f(k_l(t+1))}{\langle f\rangle(t+1)} > N\frac{f(k_l(t))}{N\langle f\rangle(t)} = \frac{f(k_l(t))}{\langle f\rangle(t)}.$$

Now we consider vertex v_i. Analogically as above, using Lemma 2 we obtain

$$\frac{f(k_i(t+1))}{\langle f\rangle(t+1)} = \frac{f(k_i(t)-1)}{\langle f\rangle(t+1)} = \frac{f(k_i(t)-1)}{\langle f\rangle(t) + \frac{f(k_l(t)+1) - f(k_l(t)) + f(k_i(t)-1) - f(k_i(t))}{N}}.$$

Due to strict monotonicity of f it holds that $f(k_l+1) - f(k_l) > 0$ and we get

$$\frac{f(k_i(t+1))}{\langle f\rangle(t+1)} < N\frac{f(k_i(t)) + f(k_i(t)-1) - f(k_i(t))}{N\langle f\rangle(t) + f(k_i(t)-1) - f(k_i(t))} = N\frac{f(k_i(t)) + c}{N\langle f\rangle(t) + c},$$

where $c = f(k_i(t)-1) - f(k_i(t))$. As a result of inequality (4) we can finally derive

$$\frac{f(k_i(t+1))}{\langle f\rangle(t+1)} < N\frac{f(k_i(t))}{N\langle f\rangle(t)} = \frac{f(k_i(t))}{\langle f\rangle(t)}.$$

Analogically for a decreasing positive function $f(k)$ we obtain that

$$\frac{f(k_l(t+1))}{\langle f\rangle(t+1)} < \frac{f(k_l(t))}{\langle f\rangle(t)} \quad \text{and} \quad \frac{f(k_i(t+1))}{\langle f\rangle(t+1)} > \frac{f(k_i(t))}{\langle f\rangle(t)}.$$

Therefore the selection probability $s(k) = \frac{f(k)}{N\langle f\rangle}$ changes for all N vertices in G, when $\langle f\rangle$ changes. □

Before we can prove Theorem 6 below, which describes how often (probabilistically) the degree distribution changes for a scale-free network during the process, we need the following technical Lemma 5.

Lemma 5. *Let $G(t) = (V, E(t))$ be a network. The probability that the degree distribution does not change in a single discrete time step of Process 1 is*

$$\Pr[\forall k \in [0, 2L] : P(k,t) = P(k,t+1)] = \sum_{k=1}^{2L} \frac{f(k)P(k)k}{\langle f\rangle 2L} + \sum_{k=1}^{2L} \frac{f(k-1)P(k-1)P(k)k}{\langle f\rangle\langle k\rangle}.$$

Proof. By Lemma 1 we get

$$\Pr[\forall k \in [0, 2L] : P(k,t) = P(k,t+1)] \Longleftrightarrow \Pr[v_i = v_l \vee k_i(t) = k_l(t) + 1],$$

and since $\Pr[v_i \neq v_j \wedge k_l(t) + 1 \neq k_i(t)] = 0$ (see the proof of Lemma 1) we obtain

$$\Pr[\forall k \in [0, 2L] : P(k,t) = P(k,t+1)] = \Pr[v_i = v_l] + \Pr[k_i(t) = k_l(t) + 1].$$

For the first probability we have

$$\Pr[v_i = v_l] = \sum_{k=0}^{2L} \Pr[v_i = v_l | k_i(t) = k] \Pr[k_i(t) = k] = \sum_{k=1}^{2L} \frac{f(k)}{N \cdot \langle f\rangle} \frac{N(k)k}{2L},$$

since there are $N(k)$ vertices each connected to k half-edges, the probability of selecting a half-edge connected to a vertex with degree k is $\Pr[k_i(t) = k] = (N(k)k)/(2L)$. By Process 1 the probability of selecting vertex v_l with degree k is $(f(k)N(k))/(N\langle f\rangle)$, but out of all $N(k)$ vertices with degree k only one is v_i, thus $\Pr[v_i = v_l | k_i(t) = k] = f(k)/(N\langle f\rangle)$. $\Pr[k_i(t) = 0] = 0$ because v_i is selected through an edge selection, therefore the summation can start from 1. By the same argument the second probability is

$$\Pr[k_l(t) + 1 = k_i(t)] = \sum_{k=0}^{2L} \Pr[k_l(t) + 1 = k_i(t) | k_i(t) = k] \Pr[k_i(t) = k] = \sum_{k=1}^{2L} \frac{f(k-1)N(k-1)}{N \cdot \langle f\rangle} \frac{N(k)k}{2L}.$$

Therefore, the overall probability is

$$\Pr[\forall k \in [0, 2L] : P(k,t) = P(k,t+1)] = \sum_{k=1}^{2L} \frac{f(k)}{N \cdot \langle f\rangle} \frac{N(k)k}{2L} + \sum_{k=1}^{2L} \frac{f(k-1)N(k-1)}{N \cdot \langle f\rangle} \frac{N(k)k}{2L}.$$

Since $P(k) = (N(k))/N$ and the expected degree per vertex is $\langle k\rangle = (2L)/N$, we get

$$\Pr[\forall k \in [0, 2L] : P(k,t) = P(k,t+1)] = \sum_{k=1}^{2L} \frac{f(k)P(k)k}{\langle f\rangle 2L} + \sum_{k=1}^{2L} \frac{f(k-1)P(k-1)P(k)k}{\langle f\rangle\langle k\rangle}. \qquad \square$$

From Lemmas 3 and 4 we know that if the distribution $P(k,t)$ changes then the selection probability $s(k)$ changes for all vertices. Now we want to know what is the probability that $P(k,t)$ is changed. To estimate this probability we consider a class of networks for which the degree distribution is bounded from above by a scale-free function. We call a network $G = (V,E)$ *scale-free* for the parameters $\alpha \in \mathbb{R}^+$ and $\gamma > 1$, if the degree distribution is bounded by

$$P(k) \leqslant \begin{cases} \alpha k^{-\gamma} & \text{for } k \geqslant 1, \\ 1 & \text{for } k = 0. \end{cases}$$

This class of networks is in fact larger than the one usually understood under the term scale-scale. Moreover, we bound also the (nonlinear) preference function on the interval $[0,2L]$ to be inside a region illustrated in Fig. 3. The region is bounded from above by linear function $\beta_1 k + \beta_0$ and from the bottom by $\beta_3 k - \beta_3$. It is possible to consider a larger class of functions $f(k)$, but this will make the proof more complex and we do not believe it would bring substantially new information. Under these assumptions we can bound from below the probability of a change in $P(k,t)$ as follows.

Theorem 6. *Let $G = (V,E)$ be a scale-free network with parameters $\gamma \geqslant 2$ and let f be a preference function that is bounded by a region defined as (see Fig. 3)*

$$f(k) \leqslant \beta_1 k + \beta_0, f(k) \geqslant \begin{cases} 0 & \text{for } k \in [0,1), \\ \beta_3 k - \beta_3 & \text{for } k \geqslant 1, \end{cases}$$

where $\beta_1, \beta_3 > 0$ and $\beta_0 \geqslant 0$. Then the probability that the degree distribution stays the same in one discrete time step is bounded by

$$\Pr[\forall k \in [0, 2L] : P(k,t) = P(k,t+1)] \leqslant \frac{\alpha\beta_1}{\beta_3\langle k\rangle - \beta_3} + \frac{4\alpha\beta_0 \ln(2L)}{2L[\beta_3\langle k\rangle - \beta_3]} + \frac{\beta_0}{\langle k\rangle[\beta_3\langle k\rangle - \beta_3]} + \frac{2\alpha^2(\beta_1 + \beta_0)}{\langle k\rangle[\beta_3\langle k\rangle - \beta_3]}.$$

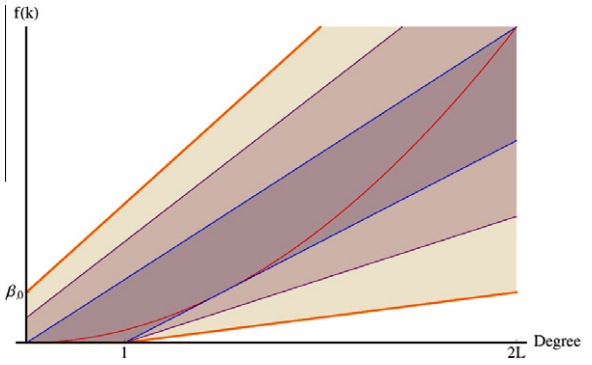

Fig. 3. Illustration of bounding area for the preference function $f(k)$ in Theorem 6.

Proof. Lemma 5 states that

$$\Pr[\forall k \in [0,2L] : P(k,t) = P(k,t+1)] = \sum_{k=1}^{2L} \frac{f(k)P(k)k}{2L\langle f\rangle} + \sum_{k=1}^{2L} \frac{f(k-1)P(k-1)P(k)k}{\langle f\rangle\langle k\rangle}. \tag{5}$$

Under the assumption that the network is scale-free and that $f(k)$ is bounded linearly, for the first summand we get

$$\frac{\sum_{k=1}^{2L} f(k)P(k)k}{2L\langle f\rangle} \leqslant \frac{\sum_{k=1}^{2L}(\beta_1 k+\beta_0)\alpha k^{-\gamma}k}{2L\sum_{k=1}^{2L}(\beta_k-\beta_3)P(k)} = \frac{\alpha\beta_1\sum_{k=1}^{2L}k^{2-\gamma}+\alpha\beta_0\sum_{k=1}^{2L}k^{1-\gamma}}{2L\left[\beta_3\sum_{k=1}^{2L}kP(k)-\beta_3\sum_{k=1}^{2L}P(k)\right]}.$$

By definition of $\sum_{k=1}^{2L} kP(k) = \langle k\rangle$. We bound $\sum_{k=1}^{2L} P(k)$ from above by 1 and by the fact that $\gamma \geqslant 2$, we obtain

$$\frac{\sum_{k=1}^{2L} f(k)P(k)k}{2L\langle f\rangle} \leqslant \frac{\alpha\beta_1 2L+\alpha\beta_0\sum_{k=1}^{2L}k^{-1}}{2L[\beta_3\langle k\rangle-\beta_3]}.$$

The harmonic series $\sum_{k=1}^{2L} k^{-1}$ converges against $\ln(2L)+\bar{\gamma}$ where $\bar{\gamma}$ is the Euler–Mascheroni constant. Thus, for all $L \geqslant 1$ it holds that $\ln(2L)+\bar{\gamma} < \ln(2L)+1$. Consequently,

$$\frac{\sum_{k=1}^{2L} f(k)P(k)k}{2L\langle f\rangle} \leqslant \frac{\alpha\beta_1 2L}{2L[\beta_3\langle k\rangle-\beta_3]} + \frac{\alpha\beta_0(\ln(2L)+1)}{2L[\beta_3\langle k\rangle-\beta_3]} \leqslant \frac{\alpha\beta_1}{\beta_3\langle k\rangle-\beta_3} + \frac{4\alpha\beta_0\ln(2L)}{2L[\beta_3\langle k\rangle-\beta_3]}. \tag{6}$$

Analogously, for the second summand we get

$$\frac{\sum_{k=1}^{2L} f(k-1)P(k-1)P(k)k}{\langle f\rangle\langle k\rangle} = \frac{f(0)P(0)P(1)+\sum_{k=2}^{2L} f(k-1)P(k-1)P(k)k}{\langle k\rangle\sum_{k=1}^{2L} f(k)P(k)} \leqslant \frac{\beta_0+\sum_{k=2}^{2L}(\beta_1(k-1)+\beta_0)\alpha(k-1)^{-\gamma}\alpha k^{-\gamma}k}{\langle k\rangle\sum_{k=1}^{2L}(\beta_3 k-\beta_3)P(k)}.$$

For $\gamma \geqslant 2$ we get

$$\frac{\sum_{k=1}^{2L} f(k-1)P(k-1)P(k)k}{\langle f\rangle\langle k\rangle} \leqslant \frac{\beta_0+\alpha^2\sum_{k=2}^{2L}(\beta_1(k-1)+\beta_0)(k-1)^{-2}k^{-1}}{\langle k\rangle\left[\beta_3\sum_{k=1}^{2L}kP(k)-\beta_3\sum_{k=1}^{2L}P(k)\right]}.$$

Since k^{-1} is monotone decreasing, $k^{-1} \leqslant (k-1)^{-1}$. Again, we bound $\sum_{k=1}^{2L} P(k)$ from above by 1. By shifting the index of the sum, we obtain

$$\frac{\sum_{k=1}^{2L} f(k-1)P(k-1)P(k)k}{\langle f\rangle\langle k\rangle} \leqslant \frac{\beta_0+\alpha^2\sum_{k=1}^{2L-1}(\beta_1 k+\beta_0)k^{-2}k^{-1}}{\langle k\rangle\left[\beta_3\sum_{k=1}^{2L}kP(k)-\beta_3\right]} = \frac{\beta_0+\alpha^2\beta_1\sum_{k=1}^{2L-1}k^{-2}+\alpha^2\beta_0\sum_{k=1}^{2L-1}k^{-3}}{\langle k\rangle[\beta_3\langle k\rangle-\beta_3]}.$$

It is $\sum_{k=1}^{\infty} k^{-x} \leqslant 2$ for any $x \geqslant 2$ which yields

$$\frac{\sum_{k=1}^{2L} f(k-1)P(k-1)P(k)k}{\langle f\rangle\langle k\rangle} \leqslant \frac{\beta_0}{\langle k\rangle[\beta_3\langle k\rangle-\beta_3]} + \frac{2\alpha^2(\beta_1+\beta_0)}{\langle k\rangle[\beta_3\langle k\rangle-\beta_3]}. \tag{7}$$

Substituting (6) and (7) into (5) yields

$$\Pr[\forall k \in [1,2L] : P(k,t) = P(k,t+1)] \leqslant \frac{\alpha\beta_1}{\beta_3\langle k\rangle-\beta_3} + \frac{4\alpha\beta_0\ln(2L)}{2L[\beta_3\langle k\rangle-\beta_3]} + \frac{\beta_0}{\langle k\rangle[\beta_3\langle k\rangle-\beta_3]} + \frac{2\alpha^2(\beta_1+\beta_0)}{\langle k\rangle[\beta_3\langle k\rangle-\beta_3]},$$

which completes the proof. □

Let us consider what Theorem 6 means for networks observed in real situations. In [10] p. 80 the authors collected basic characteristics (average degree, γ etc.) from more than 30 real networks. In many important cases like for example the Web, the average degree $\langle k\rangle$ is larger than 10 and L is much larger than 1000. Already for a value of 1000 the second term in Theorem 6 is of order 10^{-2} therefore we can neglect this term for most real networks. In most of the cases $\gamma \geqslant 2$. It is also reasonable to suppose a weak nonlinearity in $f(k)$ therefore we can suppose $\beta_0 = \beta_1 = \beta_3 = 1$. In Fig. 4 we show how the probability in Theorem 6 depends on the average degree if we take the above assumptions. It is also clear from Fig. 4 that our estimate in Theorem 6 is not accurate enough for $k \leqslant 6$ because probability cannot be larger than 1. We can conclude that for real networks with average degree larger than 8 the probability that degree distribution changes is larger than 1/2.

To capture the problems in design of parallel simulation algorithms for network processes we consider dependencies between the computation states. During simulation of the process the network changes its structure and runs through different states. For a parallelization it is important how the current state depends on its predecessor states. If there is global structural change in the predecessor state, the current state depends strongly on it, i.e. the state can only be computed when all information about the predecessor state are available. On the other hand, if the predecessor is subject to smaller changes, the successor state may be computed in parallel with the predecessor, using only information of the predecessor's predecessor. The information needed by Process 1 to compute the next state is the selection probability. That means, a state strongly depends on the predecessor state for which the selection probability changes globally. Now, assume we know any state of a simulation of Process 1 from discrete time step 0 to T and assume that there are no two states with equal selection probability for all vertices. With serial computation the complexity is of $\Omega(T)$. We want to investigate how much we can gain from parallelization methods when repeating exactly the same simulation.

To quantify this gain more formally we introduce a dependency tree. Let $\mathcal{G}(T) = \{G(1), G(2), \ldots, G(T)\}$ be the set of all states $G(t) = (V, E(t))$ of the simulation of Process 1 on some network G. A state $G(t) \in \mathcal{G}$ is *globally changed*, if the selection probability changes for any $v_i \in V$ of $G(t)$, i.e. $\forall v_i \in V$: $s(k(t)) \neq s(k(t-1))$. The starting state $G(0)$ is defined as globally changed. We say $G(t_j)$ *strongly depends* on $G(t_i)$, if $t_i < t_j$ and $G(t_i)$ is globally changed and if for all t_k with $t_i < t_k < t_j$ it holds that $G(t_k)$ is not globally changed. Let $\mathcal{E}$ be a set of directed edges, where $(G(t_i), G(t_j)) \in \mathcal{E}$ iff $G(t_j)$ strongly depends on $G(t_i)$. Then $(\mathcal{G}(T), \mathcal{E})$ is the *dependency tree* of $\mathcal{G}(\mathcal{T})$.

The dependency tree reflects the obstacles in parallelization under the assumption that all strongly dependent states cannot be computed before the state they depend on. Under this assumption the best possible method for parallelization would be to compute any state in parallel which depends on a state that is already computed. The dependency tree exactly represents this behavior: an edge can be viewed as a computation step which leads from one state to another. If a state has several outgoing edges, i.e. several other states depend on it, the computation for each outgoing edge is done in parallel.

We can consider the height (depth) of the dependency tree as a measure of parallelization for a given simulation. By definition, any state of the dependency tree has an incoming edge from a different state, except the starting state $G(0)$. Any globally changed state depends on a globally changed state. There cannot be two globally changed states $G(t_1)$ and $G(t_2)$ which depend on the same state (assume w.l.o.g. that $t_1 < t_2$, then by definition of strong dependency t_1 cannot be globally changed). Hence, there is exactly one globally changed state for each level of the dependency tree, except for the last level. All non globally changed states are connected to globally changed states, hence the height of the tree is the number of globally changed states plus one. We can estimate the expected height of the dependency tree if we know the probability that the degree distribution changes in one time step of the Process 1. We are aware that the mean depth of the dependency tree as defined in our situation does not provide a full-blown average case proof in classical models as *PRAM*, however we believe that our analysis provides a strong indication that algorithms highly parallel in expected case are prohibited in classical models too.

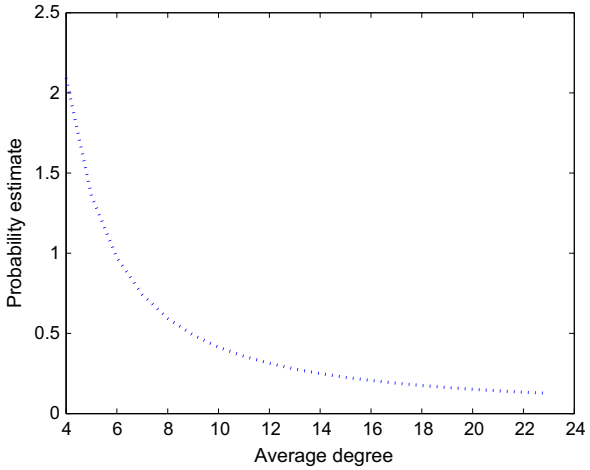

Fig. 4. Dependence of the probability estimate according to Theorem 6 on the average degree. We suppose $L = 1000$, $\alpha = 2$, $\beta_0 = \beta_1 = \beta_3 = 1$.

Theorem 7. *Let f be a strictly monotonic (decreasing or increasing) function with f(k + 1) − f(k) being injective. Assume the degree distribution changes in one discrete time step with a probability of at least P. Then the expected depth D(T) of the dependency tree* $(\mathcal{G}(\mathcal{T}), \mathcal{E})$ *when simulating T steps of Process 1 is bounded from below by PT, i.e.*

$$E[D(T)] \geqslant PT.$$

Proof. If the degree distribution changes in one discrete time step with probability P, then by Lemma 3 it holds that

$$\Pr[\forall k P(k,t) = P(k,t+1)] = \Pr[\langle f\rangle(t) = \langle f\rangle(t+1)], \tag{8}$$

if $f(k+1) - f(k)$ is injective. From Lemma 4 it follows for f being strictly monotonic (decreasing or increasing) that

$$\Pr[\langle f\rangle(t) \neq \langle f\rangle(t+1)] \leqslant \Pr[\forall v_i \in V : s(k,t) \neq s(k,t+1)], \tag{9}$$

where $s(k,t) = f(k_i(t))/(N\langle f\rangle(t))$. Combining (8) and (9) with the premise that a change in the degree distribution is bounded by P yields

$$P \leqslant \Pr[\forall k P(k,t) \neq P(k,t+1)] \leqslant \Pr[\forall v_i \in V : s(k,t) \neq s(k,t+1)].$$

This means, that the probability that the state at time step $t + 1$ is globally changed is not less than P.

By definition of the dependency tree, any globally changed state sits on a new level since it is connected to the direct predecessor state which is globally changed. That means the nth globally changed state is at depth $n - 1$, because the first state is globally changed and has depth 0.

Now we count the depth of the dependency tree by introducing a Bernoulli random variable b_t with $1 \leqslant t \leqslant T$ for any state $G(t)$ of Process 1, which is 1 if $G(t)$ is globally changed and 0 otherwise. Thus, the depth of the dependency tree is $D(T) = \sum_{t=1}^{T} b_t$. Then the expected depth is

$$E[D(T)] = E\left[\sum_{t=1}^{T} b_t\right] = \sum_{t=1}^{T} E[b_t],$$

where $E[b_t] = \Pr[b_t = 1]$. By (7) we have $\Pr[b_t = 1] \geqslant P$ and thus

$$E[D(T)] \geqslant \sum_{t=1}^{T} P = TP,$$

which completes the proof. □

The expected height of the dependency tree according Theorem 7 can be very large in real networks. As we have discussed above, one can expect probability P in Theorem 7 to be around 1/2 for many real networks with nonlinear preference function. Therefore the expected height of dependency tree can reach $T/2$ where T is the simulation time leading to considerable difficulties in parallelization of such processes. On the other hand, if the process has linear preference function or approximate results are satisfactory, the highly parallel algorithm presented in the next section can be used.

4. Parallel algorithm for linear preference function

In the previous section we analyzed Process 1 and we have shown that in many cases it is difficult if not impossible to construct a parallel algorithm. However, as we discussed in the introduction many processes observed in reality are supposed to have linear preference function. Therefore, it would be important to design parallel algorithms to simulate the stochastic processes in these networks. Indeed, as we show in the following section, this is possible for linear preference function in the form

$$f(k) = \beta k + k_0. \tag{10}$$

To derive a formulation of Process 1 suitable for parallel algorithms we need the concept of half-edges. Half-edge (v_i, E_j) is an object consisting of a node and an adjacent edge. In the following text we denote half-edges as H_i and the set of all half-edges as $\mathbb{H}$, where its size is $H = |\mathbb{H}|$. Selecting a vertex with linear preference $f(k) = k$ can be done by selecting a half-edge $H_i \in \mathbb{H}$ uniformly at random and selecting its vertex because each vertex v_i in the graph has attached a fraction $k_i/(2L) = (f(k_i))/(N\langle f\rangle)$ of all half-edges.

The rewiring Step 6 in Process 1 where we rewire an edge by changing one of its end nodes can also be formulated with half-edges. To represent rewiring we choose a half-edge and move it to some other node i.e. we exchange the node in the half-edge object with some other node. Because a half-edge is a pair (v_i, E_j), exchanging the node v_i with v_l does not change the other end of the edge i.e. before the step we have an edge represented by two half-edges $(v_i, E_j) - (v_j, E_j)$ and after the step we have an edge $(v_l, E_j) - (v_j, E_j)$. Therefore, in the following text we use a term "half-edge rewire" which means exchanging a vertex in a half-edge, this is equivalent to rewiring of an edge where one end vertex of the edge stays fixed and the other end vertex is exchanged.

The parallel algorithm for Process 1 is based on the reformulation of the preferential probability. The selection probability for a linear preference function (10) can be represented as

$$s(k_i) = \frac{f(k_i)}{N\langle f\rangle} = \frac{1}{N}\cdot\frac{Nk_0}{Nk_0+\beta 2L} + \frac{k_i}{2L}\cdot\frac{\beta 2L}{Nk_0+\beta 2L} = \frac{1}{N}\cdot c + \frac{k_i}{2L}\cdot(1-c), \tag{11}$$

where $1/N$ is the probability of selecting a vertex uniformly at random and $k_i/(2L)$ is a probability of selecting a vertex with linear preference $f(k_i) = k_i$. Additionally, the factors

$$c = \frac{Nk_0}{Nk_0+\beta 2L} \quad \text{and} \quad 1-c = \frac{\beta 2L}{Nk_0+\beta 2L}$$

are constant for a given network and their sum is 1. Therefore according to (11), selecting v_i with preference (10) is equivalent to the following two steps:

1. With probability c select $v_i \in V$ uniformly at random.
2. With probability $1-c$ select an half-edge H_i uniformly at random and take its vertex v_i.

If we formulate the preferential selection probability as above, we can equivalently transform Processes 1 and 3. We replace Steps 3–4 of Process 1 with Step 3 of Process 3, and Step 5 is replaced with the "if-then-else" Steps 5–9 using Eq. (11). After this transformations the process does not contain any step directly dependent on the selection probability $s(k)$ thus removing the obstacle which we analyzed in the previous section.

Process 3: SESPL

Require: a multigraph $G(V,E)$ with L edges and N vertices
1: $N_s \Leftarrow$ *StepLimit* {Initialize the number of process loops.}
2: **while** number of process loops smaller than N_s **do**
3: An half-edge H_i is selected uniformly at random. The vertex connected to H_i is denoted v_i.
4: A number $c \in [0,1]$ is selected uniformly at random.
5: **if** $c < Nk_0/(Nk_0 + \beta 2L)$ **then**
6: Select a vertex v_l uniformly at random.
7: **else**
8: Select a half-edge H_l uniformly at random. The corresponding vertex is denoted v_l.
9: **end if**
10: Rewire the half-edge H_i from v_i to v_l.
11: **end while**

The possibility to parallelize Process 3 is based on the fact that during the process steps the degree distribution is explicitly not used at all. As we have shown in the previous section, if we need to know the degree distribution to compute the selection probability the computation must basically stop until the full distribution is known. However, as a consequence of preference function linearity, all probabilistic steps in Process 3 choose an object uniformly at random i.e. the current degree distribution is not used. Because we do not need the degree distribution explicitly the algorithm does not need to know the identity of the graph vertices.

To explain the idea behind the algorithm in more detail, we need to introduce two sets of new objects: A_0 and H_0. H_0 contains all edges of the original graph represented as pairs of half-edges where the half-edge vertices are elements of a new set A_0 with the size $2L$ (see Fig. 6). In the following text we call the elements of the set A_0 virtual vertices and with a "map" we mean a function (which is not necessarily injective) from the set A_0 to V. Now we can change Process 3 to the following form:

Process 4: PSESPL

Require a multigraph $G(V,E)$ with L edges and N vertices
1: $N_s \Leftarrow$ *StepLimit* {Initialize the number of process loops.}
2: **while** number of process loops smaller than N_s **do**
3: An half-edge H_i is selected uniformly at random from H_0. The vertex connected to H_i is denoted $a_i \in A_0$.
4: A number $c \in [0,1]$ is selected uniformly at random.
5: **if** $c < Nk_0/(Nk_0 + \beta 2L)$ **then**
6: Select a vertex v_l uniformly at random from V.
7: **else**
8: Select a half-edge H_l uniformly at random. The corresponding vertex is denoted $a_l \in A_0$.
9: **end if**
10: Rewire the half-edge H_i from v_i to v_l if Step 5 was executed resp. to a_l if Step 8 was executed.
11: **end while**

The main idea in the algorithm design is that it is equivalent to make k steps of the original Process 3 or to make k steps of Process 4 using the virtual vertices from the set A_0 and then to map the virtual vertices to the graph vertices from the set V. In Fig. 6 we illustrate a few steps of Process 4 showing how the edges can move between the vertices from A_0 (virtual) and from V (original graph). If A_l is a virtual vertex and therefore an alias for a real vertex v_k, and a half-edge is rewired from A_i to A_l, then as a result at least 2 edges are connected to a single virtual vertex A_l. The number of half-edges connected to a single virtual vertex can be any number between 0 and $2L$. The number of virtual vertices acting as an alias for the same real vertex v_k can also be any number between 0 and $2L$. The sum of all half-edges connected to all aliases of v_k in addition to the half-edges directly connected with v_k represents the degree of v_k. Once a virtual vertex A_i has degree 0, no edge will ever be attached to A_i again.

Algorithm 5: Parallel algorithm for Process 4.

Require M computing units
1: $E \Leftarrow 2L$ {L is the number of edges in the network.}
2: $N_s \Leftarrow StepLimit$ {Initialize the number of process steps.}
3: Send E,N_s to every proc. unit
4: {Parallel phase: all processing units work independently.}
5: **while** all processing units are working **do**
6: construct the set V, the set of edges E, the set of half-edges H_0 and the set of virtual vertices A_0.
7: **while** number of process steps smaller than N_s **do**
8: execute a process loop of Process 4
9: **end while**
10: **end while**
11: {Sequential phase.}
12: **for** $i = 0$ to $M - 1$ **do**
13: Computing unit $M - i$ computes the map $A_0 \to V$
14: Send the map to the unit $M - i - 1$
15: **end for**
16: **return** The computing unit 1 returns the map $A_0 \to V$ {The final map defines the graph after $M \times I$ steps of Process 3}

The algorithm has two phases, a parallel phase and a sequential phase (see Fig. 5). The parallel phase comprises Steps 6–9 in Algorithm 5. At the beginning of the parallel phase, every processor constructs its own graph taking the set of L edges and assigning to every edge two adjacent vertices from A. Also, the set H_0 of half-edges and the set V of vertices are constructed. After that the process loop is repeated (steps 7–9 in Algorithm 5) simulating the stochastic discrete time until the prescribed number of time steps (process loops) N_s is reached. The basic rewiring situations are illustrated in Fig. 6, the edges can move between the vertices from sets V,A_0.

The sequential phase of the algorithm occurs in a cycle (see Fig. 5 and Steps 12–15 of Algorithm 5). We can suppose that we have M equivalent processing units $U_1,\ldots,U_M$ and that the unit U_M knows the initial graph. During the sequential phase the last U_M will now replace all the local virtual vertices with real vertices and send the whole graph topology to U_{M-1}. U_{M-1} can now accept the received topology as the topology U_{M-1} initially started with, by replacing the own virtual vertices with real vertices to generate its final graph. This sequential process continues as shown in Algorithm 5 until U_1 has replaced its own virtual vertices. After that U_1 returns the final graph.

The parallel speedup of the algorithm is linear in the limit of long process simulation time. The topology of the graph can be stored simply by memorizing the vertices half-edges are connected with. For M processors and $2L$ half-edges, the time we need for the sequential part of the algorithm is $O(M{\cdot}L)$. Considering that during the parallel phase the processing units work fully independently, the running time of this phase is approximately T/M, where T is number of process loops (discrete time) we want to simulate. Together with the sequential part we have a total running time:

$$T_p(M,L) = \frac{T}{M} + O(M \cdot L) = \frac{T + O(M^2 \cdot L)}{M}. \tag{12}$$

Therefore the parallel speedup on M processors with a large running time T is linear:

$$S_p = \lim_{T\to\infty} \frac{T \cdot M}{T + O(M^2 \cdot L)} = M. \tag{13}$$

This is also supported by the experiments where we show that in practical situations the influence of the sequential phase can be neglected. Moreover, the complexity of the sequential part of the algorithm can be further improved by the observation that the order in which the maps are applied has no influence on the result. Therefore, we can suppose that a tree organization of the map transition in a bottom-up fashion can introduce a parallel logarithmic speedup within this phase of the algorithm.

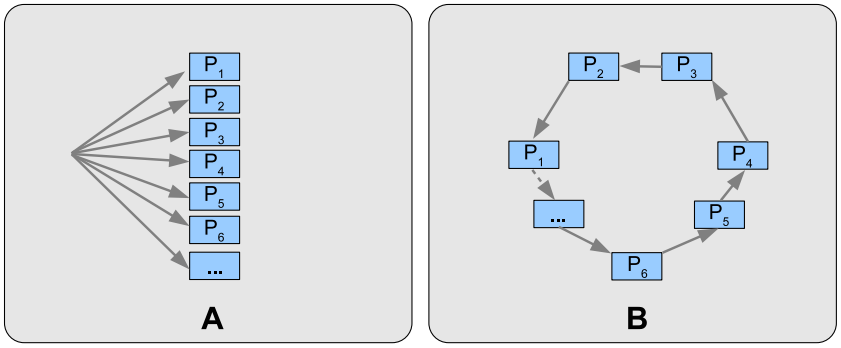

Fig. 5. The overall structure of the parallel algorithm. The first (major) phase of the algorithm is parallel, where the initial network is sent to all computing nodes. After that, all processors iterate the Process 4 independently. The iterations are occurring in a virtual manner, i.e. the processors do not know the real mapping of nodes. The second, normally much faster phase of the algorithm (see the complexity analysis in the text), depicted in Picture B, is sequential, because the processors are sequentially updating the final mapping of virtual nodes to the network node labels.

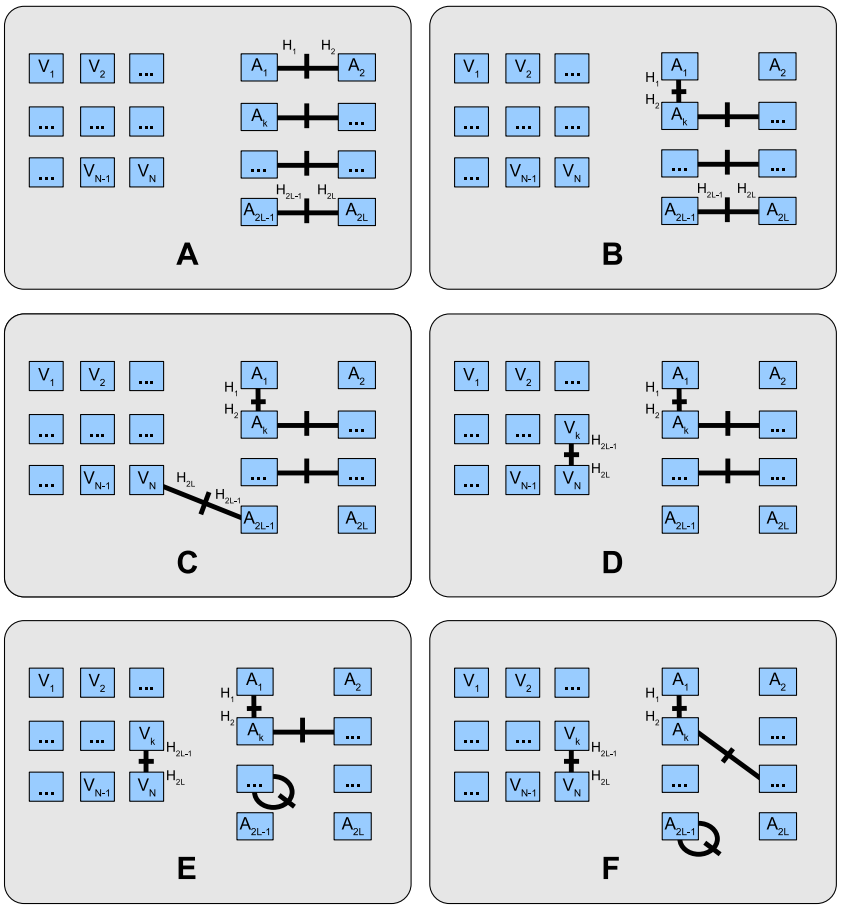

Fig. 6. Illustration of Process 4. Picture A represents the initial state of the process, where all L edges in the initial graph are leading to the virtual vertices from the set A_0 and on the left the set all vertices V is represented. H_i denotes the half-edges. Picture B shows the network after the first rewiring process loop. The half-edge H_2 was selected for rewiring in the process Step 3, in Step 8 the half-edge H_k leading to A_k was selected that means the edge $A_1 - A_2$ goes to $A_1 - A_k$. Similarly in Picture C we see the graph after next process loop where in Step 5 a vertex from V was selected. The next process loop (result in Picture D) is similar moving the edge from A_{2L-1} to v_k. In Picture E and F next two loops of the process are illustrated which create a self loop on a virtual vertex and rewire another edge between the virtual vertices.

We can also think of a modified version of the parallel algorithm, where the processing units are simulating a smaller chunk of process loops (time steps) and the sequential phases of the algorithm are interleaving the parallel phases. In this case the sequential map reconstruction process has to cycle several times through all processors until the final discrete

process time is reached. In this case, each processor has to restart the parallel phase with a new set of virtual vertices, once finished the intermediate mapping step. This version can be useful when each processor has to check or correct its own progress from time to time, or when we need some intermediate results.

5. Experimental Results

To verify the parallel algorithm from the previous section we developed an experimental framework on the HPC Cluster Brutus [6] located at ETH Zurich. Brutus is currently ranking 88 in the TOP500 [16] list of high-performance computers. The Brutus architecture consists of 10,000 cores with several types of AMD processors. The ranking was done on a homogeneous subsystem of Brutus consisting of 410 nodes with four quad-core AMD Opteron 8380 CPUs and 32 GB of RAM (6560 cores). All nodes are connected to the cluster's Gigabit Ethernet backbone, 256 nodes use a high-speed Quadrics QsNetII network and 508 nodes are connected to a high-speed InfiniBand QDR network. To program the parallel algorithm we used the MPI library [12].

The performance of the algorithm is measured in seconds. Every measurement consists of three time measures of Algorithm 5: the initialization time during which the initial graph is sent to all computing units (denoted with "distribute"), time of parallel computation ("rewire") and time of the sequential collection phase ("collect"). To reduce the noise, we repeat the computation for every parameter set 10 times and average the results. The process is simulated for $5 \cdot 10^8$ steps on a network with $4 \cdot 10^6$ edges and 10^6 nodes. To consider the parallel speedup, we repeat the same simulation with $1, 2, 4, \ldots, 64$ and 128 cores lying on nodes connected with InfiniBand QDR network. As we argued above, it is sufficient to store only half-edges to represent the graph (the vertices with degree 0 does not need to be stored for our algorithm), therefore to illustrate the scaling with respect to the graph size, we repeat the experiments with 2, 4 and 8 times 10^6 edges. The sequential part of the algorithm (collect) is implemented using a bottom-up tree data structure which has logarithmic time dependency on the number of processors.

The queue management system of Brutus cluster does not allow to have more processes resp. threads on 1 core. However, we were principally interested whether a hyperthreading or other architectural aspects can bring a principal speedup for our particular type and implementation of the algorithm. Because the operating system in our algorithm interprets the same sequence of instructions, we expected a neglectable effect. This fact is confirmed by our experiments, illustrated in Fig. 7 which were computed on our development platform where the computation was constrained to 2 cores located on one processor. In this set of experiments we distributed the task to the increasing number of processes (x-axis) to see how the speedup of the algorithm depends on the number of parallel processes on one core. The results confirm that in our case it is optimal to distribute 1 Process per core. Marginally, it is interesting to observe the difference between the even and the odd number of processes.

In Figs. 8 and 9 we can observe that the parallel algorithm scales linearly over the cluster. For the given number of iteration ($5 \cdot 10^8$) the peak performance is achieved with 64 processes allocating 1 process for every core. After that the saturation phase occurs caused by the sequential part of algorithm. The scaling with increasing size of the graph is also linear as was predicted in the algorithm analysis. This is visible in Fig. 9 where the speedup only differs in the saturation phase where

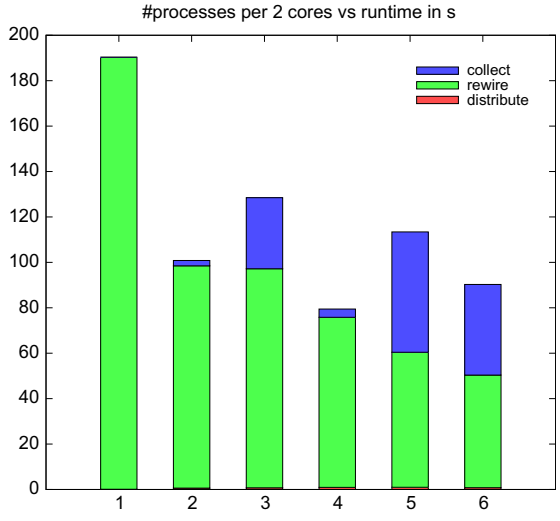

Fig. 7. The results of experiments showing the dependence between number of processes (x-axis) on one core and the algorithm performance (y-axis). The performance is measured in seconds, and the time bars have three time components: initialization time (distribute), the parallel phase time (rewire) and the collection time (collect). The process was simulated for $5 \cdot 10^8$ steps on a network with 10^6 nodes and $4 \cdot 10^6$ edges. The optimal choice in our case is to allocate 1 process per core because hyperthreading does help only marginally in the case of core overloading with even number of processes.

the sequential part of the algorithm prevails. During the parallel phase of Algorithm 5 the processors work independently from each other. This can be observed in both Figs. 8 and 9. The overall speedup does not change its character during the passage between different nodes which happens at multiples of 16 cores. This shows that if the task is distributed across more computing nodes resp. processors there is no observable slow-down effect because of the network latency.

6. Conclusions

In our paper we analyzed a parallel simulation of scale-free networks. For a fundamental class of non-growing networks with linear preference function, we provide a highly parallel algorithm, which has two phases. The first phase is fully parallel and has speedup M, where M is the number of processing units. The second phase is sequential, but independent on simulation time T, which is the main source of complexity. On the other hand, we theoretically analyzed dependencies which are prohibiting parallelism for an exact process. Our theoretical interest stems from the difficulties we observed when we tried to design a parallel algorithm for simulation of general scale-free networks. The difficulties are arising from the fact that the preferential selection step, which is used in most scale-free network models, is strongly dependent on the degree

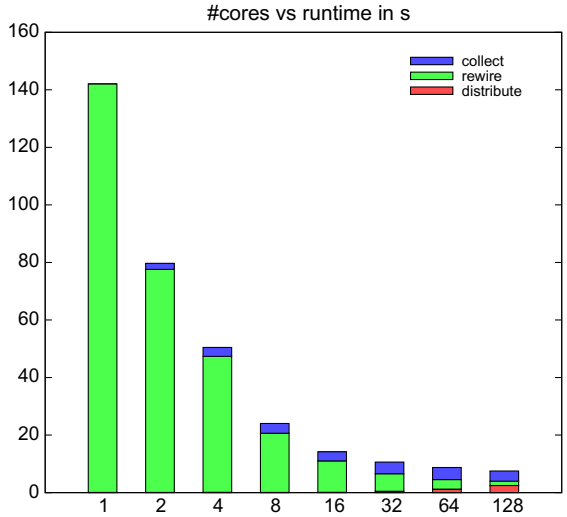

Fig. 8. The figure illustrates a set of experiments for Algorithm 5 where the task is distributed on up to 128 cores. The performance is measured in seconds, and the bars contain three time components: the initialization time (distribute), the parallel phase time (rewire) and the collection time (collect). The process was simulated for $5 \cdot 10^8$ steps on a network with 10^6 nodes and $4 \cdot 10^6$ edges. For fixed number of simulation steps, the sequential part of the algorithm must prevail, if the number of cores increases over certain threshold.

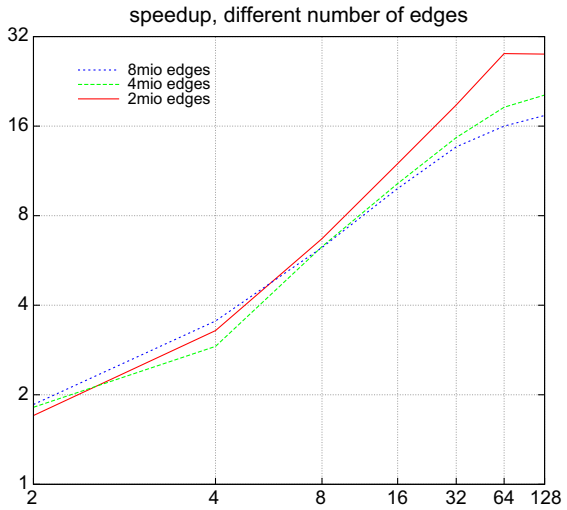

Fig. 9. The overall speedup of Algorithm 5 is linear up to the point of saturation where the sequential part prevails. The algorithm also linearly scales with the size of the graph as is illustrated by three experiments with $2 \cdot 10^6$,$4 \cdot 10^6$ and $8 \cdot 10^6$ edges.

distribution. This in turn changes with high probability during a stochastic equilibrium process which generates scale-free networks. This is valid for a wide range of nonlinear preference functions.

For many networks, the linear preference function can provide a sufficient approximation for the simulation of network evolution. However, an interesting future question, which we believe is solvable, is whether one can approximate the nonlinear preference function with a piecewise linear function. To provide a parallel algorithm in this case, it is necessary to consider at least two problems. The first problem is how to handle the transition from one linear function segment to another. We suppose that a method is needed how to pre-compute larger maps providing sufficient information to handle the transition to a different linear segment. The second problem is to generalize the factorization in Eq. (11).

Another method to handle the preferential attachment consist in approximations of the process which would allow to neglect the effect of degree distribution changes. We suppose that such method is possible for a wide class of networks but more research would be needed to know under which conditions and to which extent such propagation can be neglected. Naturally, for many practical problems an empirical evidence can be elaborated, how many process steps can be simulated before a global update of degree distribution is necessary.

Theoretically, it would be interesting to investigate larger classes of preference functions. In Theorem 6, a different method might be needed to estimate the probability for different classes of networks and preference functions. The estimation method would also need improvement if we want to consider very sparse networks which have average degree near zero.

Acknowledgement

The authors thank professor Peter Widmayer for inspiring discussions and ongoing support of this project. We would also like to thank the anonymous reviewers for their suggestions which improved the quality of the paper.

References

[1] R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (2002) 47–97.
[2] D.A. Bader, K. Madduri, Snap, small-world network analysis and partitioning: An open-source parallel graph framework for the exploration of large-scale networks, in: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, April 2008, pp. 1–12.
[3] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (509) (1999).
[4] A.-L. Barabási, R. Albert, H. Jeong, Mean-field theory for scale-free random networks, Physica A (272) (1999).
[5] S. Boccalettia, V. Latorab, Y. Morenod, M. Chavezf, D.-U. Hwanga, Complex networks: Structure and dynamics, Physics Reports 424 (2006) 175–308.
[6] ETH Cluster Brutus. <http://en.wikipedia.org/wiki/brutus_cluster>, 2009. (Wikipedia on Brutus, the high-performance cluster at ETH Zurich0.
[7] A. Cami, N. Deo, Techniques for analyzing dynamic random graph models of web-like networks: An overview, Networks 51 (4) (2008) 211–255.
[8] L. da Fontoura Costa, O.N. Oliveira Jr., G. Travieso, F.A. Rodrigues, P.R.V. Boas, M.P. Viana, L. Antiqueira, L.E. Correa da Rocha, Analyzing and modeling real-world phenomena with complex networks: A survey of Applications, arXiv.org, (arXiv:0711.3199v3), 2008.
[9] G. D'Angelo, S. Ferretti, Simulation of scale-free networks, in: Simutools'09: Proceedings of the Second International Conference on Simulation Tools and Techniques ICST, Belgium, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, 2009, pp. 1–10.
[10] S.N. Dorogovtsev, J.F.F. Mendes, Evolution of Networks, Oxford University Press, 2003.
[11] T.S. Evans, A.D.K. Plato, Exact solution for the time evolution of network rewiring models, Physical Review E 75 (056101) (2007).
[12] MPI Forum. <http://www.mpi-forum.org/>, 2009. (World Wide Web electronic publication of official MPI standards documents).
[13] T. Hruz, M. Natora, M. Agrawal, Higher-order distributions and non-growing complex networks without multiple connections, Physical Review E 77 (046101) (2008).
[14] H. Jeong, B. Tombor, R. Albert, Z.N. Oltvai, A.-L. Barabasi, The large-scale organization of metabolic networks, Nature 407 (2000) 651–654.
[15] K. Madduri, D.A. Bader, Compact graph representations and parallel connectivity algorithms for massive dynamic network analysis, in: International Parallel and Distributed Processing Symposium, 0:1–11, 2009.
[16] H. Meuer, E. Strohmaier, J. Dongarra, H. Simon. <http://www.top500.org>, November 2009. (World Wide Web electronic publication of the Top 500 Supercomputer Sites).
[17] M. Mihail, C. Papadimitriou, A. Saberi, On certain connectivity properties of the internet topology, Journal of Computer and System Sciences 72 (2006) 239–251.
[18] M.E.J. Newman, The structure and function of complex networks, Physical Review E 45 (2) (2003) 167256.
[19] Kwangho Park, Ying-Cheng Lai, Nong Ye, Self-organized scale-free networks, Physical Review E 72 (026131) (2005).
[20] A. Wagner, D.A. Fell, The small-world inside large metabolic networks, Proceedings of the Royal Society of London B 268 (2001) 1803–1810.
[21] D.J. Watts, S.H. Strogatz, Collective dynamics of small-world networks, Nature 393 (1998) 440–442.
[22] A. Yoo, K. Henderson, Parallel generation of massive scale-free graphs. arxiv.org, (arXiv:1003.3684v1), 2010.
[23] A. Youssef, A parallel algorithm for random walk construction with application to the Monte Carlo solution of partial differential equations, IEEE Transactions on Parallel and Distributed Systems 4 (3) (1993) 355–360.