Parallel and I/O-efficient Randomisation of Massive Networks using Global Curveball Trades

Corrie Jacobien Carstens
University of Amsterdam, Netherlands
c.j.carstens@uva.nl

Michael Hamann
Karlsruhe Institute of Technology, Germany
michael.hamann@kit.edu

Ulrich Meyer
Goethe University, Frankfurt, Germany
umeyer@ae.cs.uni-frankfurt.de

Manuel Penschuck
Goethe University, Frankfurt, Germany
mpenschuck@ae.cs.uni-frankfurt.de

Hung Tran
Goethe University, Frankfurt, Germany
htran@ae.cs.uni-frankfurt.de

Dorothea Wagner
Karlsruhe Institute of Technology, Germany
dorothea.wagner@kit.edu

Abstract

Graph randomisation is a crucial task in the analysis and synthesis of networks. It is typically implemented as an *edge switching* process (*ESMC*) repeatedly swapping the nodes of random edge pairs while maintaining the degrees involved [23]. Curveball is a novel approach that instead considers the whole neighbourhoods of randomly drawn node pairs. Its Markov chain converges to a uniform distribution, and experiments suggest that it requires less steps than the established *ESMC* [6]. Since trades however are more expensive, we study Curveball's practical runtime by introducing the first efficient Curveball algorithms: the I/O-efficient *EM-CB* for simple undirected graphs and its internal memory pendant *IM-CB*. Further, we investigate *global trades* [6] processing every node in a single super step, and show that undirected global trades converge to a uniform distribution and perform superior in practice. We then discuss *EM-GCB* and *EM-PGCB* for global trades and give experimental evidence that *EM-PGCB* achieves the quality of the state-of-the-art *ESMC* algorithm *EM-ES* [15] nearly one order of magnitude faster.

2012 ACM Subject Classification Mathematics of computing → Random graphs

Keywords and phrases Graph randomisation, Curveball, I/O-efficiency, Parallelism

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.11

Supplement Material Stable versions of *IM-CB* and *EM-GCB* are released as part of NetworKit (`http://network-analysis.info`).

Funding This work was partially supported by Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2, ME 2088/4-2, and WA 654/22-2.

Acknowledgements We thank the anonymous reviewers for their many insightful comments and suggestions.

1 Introduction

In the analysis of complex networks, such as social networks, the underlying graphs are commonly compared to random graph models to understand their structure [17, 27, 34]. While simple models like Erdős-Rényi graphs [11] are easy to generate and analyse, they are too different from commonly observed powerlaw degree sequences [27, 26, 34]. Thus, random graphs with the same degree sequence as the given graph are frequently used [8, 17, 32]. In practice, many of these graphs are simple graphs, i.e. graphs without self-loops and multiple edges. In order to obtain reliable results in these cases, the graphs sampled need to be simple since non-simple models can lead to significantly different results [31, 32]. The randomisation of a given graph is commonly implemented as an edge switching Markov chain *ESMC* [8, 24].

Nowadays, massive graphs that cannot be processed in the RAM of a single computer, require new analysis algorithms to handle these huge datasets. In turn, large benchmark graphs are required to evaluate the algorithms' scalability – in terms of speed and quality. LFR is a standard benchmark for evaluating clustering algorithms which repeatedly generates highly biased graphs that are then randomised [18, 19]. [15] presents the external memory LFR generator *EM-LFR* and its I/O-efficient edge switching *EM-ES*. Although *EM-ES* is faster than previous results even for graphs fitting into RAM, it dominates *EM-LFR*'s running time. Alternative sampling via the Configuration Model [25] was studied to reduce the initial bias and the number of *ESMC* steps necessary [14]. Still, graph randomisation remains a major bottleneck during the generation of these huge graphs.

The Curveball algorithm has been originally proposed for randomising binary matrices while preserving row and column sums [35, 36] and has been adopted for graphs [5, 6]: instead of switching a pair of edges as in *ESMC*, Curveball trades the neighbours of two nodes in each step. Carstens et al. further propose the concept of a *global trade*, a super step composed of single trades targetting every node[1] in a graph once [6]. The authors show that global trades in bipartite or directed graphs converge to a uniform distribution, and give experimental evidence that global trades require fewer Markov chain steps than single trades. However, while fewer steps are needed, the trades themselves are computationally more expensive. Since we are not aware of previous efficient Curveball algorithms and implementations, we investigate this trade-off here.

Our contributions. We present the first efficient algorithms for Curveball: the (sequential) internal memory and external memory algorithms *IM-CB*[2] and *EM-CB* for the Simple Undirected Curveball algorithm (see section 4). Experiments in section 5, indicate that they are faster than the established edge switching approaches in practice.

In section 3, we show that random global trades lead to uniform samples of simple, undirected graphs and demonstrate experimentally in section 5 that they converge even faster than the corresponding number of uniform single trades. Exploiting structural properties of global trades, we simplify *EM-CB* yielding *EM-GCB* and the parallel I/O-efficient *EM-PGCB* which achieves *EM-ES*'s quality nearly one order of magnitude faster in practice (see section 5).

[1] For an odd number n of nodes, either a single node is left out or equivalently an isolated node is added.
[2] We prefix internal memory algorithms with `IM` and I/O-efficient algorithms with `EM`. The suffices `CB`, `GCB`, and `PGCB` denote Curveball, CB. with global trades, and parallel CB. with global trades respectively.

2 Preliminaries and Notation

We define the short-hand $[k] := \{1, \ldots, k\}$ for $k \in \mathbb{N}_{>0}$, and write $[\, x_i \,]_{i=a}^{b}$ for an ordered sequence $[x_a, x_{a+1}, \ldots, x_b]$.

Graphs and degree sequences. A graph $G = (V, E)$ has $n = |V|$ sequentially numbered nodes $V = \{v_1, \ldots, v_n\}$ and $m = |E|$ edges. Unless stated differently, graphs are assumed to be undirected and unweighted. To obtain a unique representation of an *undirected* edge $\{u, v\} \in E$, we use *ordered* edges $[u, v] \in E$ implying $u \leq v$; in contrast to a directed edge, the ordering is used algorithmically but does not carry any meaning. A graph is called *simple* if it contains neither multi-edges nor self-loops, i.e. $E \subseteq \{\, \{u, v\} \,|\, u, v \in V \text{ with } u \neq v \,\}$. For node $u \in V$ define the *neighbourhood* $\mathcal{A}_u := \{v : \{u, v\} \in E\}$ and *degree* $\deg(u) := |\mathcal{A}_u|$. Let $d_{\max} := \max_v\{\deg(v)\}$ be the maximal degree of a graph. A vector $\mathcal{D} = [\, d_i \,]_{i=1}^{n}$ is a degree sequence of graph G iff $\forall v_i \in V\colon\ \deg(v_i) = d_i$.

Randomisation and Distributions. $\textsc{Pld}\,([a, b), \gamma)$ refers to an integer Powerlaw Distribution with exponent $-\gamma \in \mathbb{R}$ for $\gamma \geq 1$ and values from the interval $[a, b)$; let X be an integer random variable drawn from $\textsc{Pld}\,([a, b), \gamma)$ then $\mathbb{P}[X{=}k] \propto k^{-\gamma}$ (proportional to) if $a \leq k < b$ and $\mathbb{P}[X{=}k] = 0$ otherwise. A statement depending on some number $x > 0$ is said to hold *with high probability* if it is satisfied with probability at least $1 - 1/x^c$ for some constant $c \geq 1$. Let S be a finite set, $x \in S$ and let σ be permutation on S, we define $\mathrm{rank}_\sigma(x)$ as the number of elements positioned in front of x by σ.

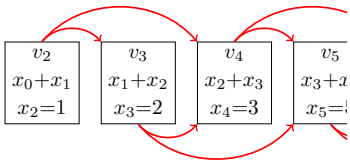
2.1 External-Memory Model

In contrast to classic models of computation, such as the unit-cost random-access machine, modern computers contain deep memory hierarchies ranging from fast registers, over caches and main memory to solid state drives (SSDs) and hard disks. Algorithms unaware of these properties may face performance penalties of several orders of magnitude.

We use the commonly accepted external memory (EM) model by Aggarwal and Vitter [1] to reason about the influence of data locality in memory hierarchies. It features two memory types, namely fast internal memory (IM or RAM) holding up to M data items, and a slow disk of unbounded size. The input and output of an algorithm are stored in EM while computation is only possible on values in IM. An algorithm's performance is measured in the number of I/Os required. Each I/O transfers a block of $B = \Omega(\sqrt{M})$ consecutive items between memory levels. Reading or writing n contiguous items is referred to as *scanning* and requires $\mathrm{scan}(n) := \Theta(n/B)$ I/Os. Sorting n consecutive items triggers $\mathrm{sort}(n) := \Theta((n/B) \cdot \log_{M/B}(n/B))$ I/Os. For all realistic values of n, B and M, $\mathrm{scan}(n) < \mathrm{sort}(n) \ll n$. Sorting complexity constitutes a lower bound for most intuitively non-trivial EM tasks [22]. EM queues use amortised $\mathcal{O}(1/B)$ I/Os per operation and require $\mathcal{O}(B)$ main memory [28]. An external priority queue (PQ) requires $\mathcal{O}(\mathrm{sort}(n))$ I/Os to push and pop n items [2].

2.2 TFP: Time Forward Processing

Time Forward Processing (*TFP*) is a generic technique to manage data dependencies of external memory algorithms [21]. Consider an algorithm computing values $x_1, \ldots, x_n$ in which the calculation of x_i requires previously computed values. One typically models these dependencies using a directed acyclic graph $G{=}(V, E)$. Every node $v_i \in V$ corresponds to the computation of x_i and an edge $(v_i, v_j) \in E$ indicates that the value x_i is necessary to compute




```
1 PQ.push(<key=2, value=0>); PQ.push(<key=2, value=1>)
2 foreach i ← 2, . . . , n do
3 |   sum ← 0
4 |   while PQ.min.key == i do                    // Two iterations
5 |   |   sum ← sum + PQ.remove-min().value
6 |   print("x_i =", sum)
7 |   PQ.push(<key=i+1, sum>); PQ.push(<key=i+2, sum>)
```

Figure 1 Left: Dependency graph of the Fibonacci sequence (ignoring base case). **Right:** Time Forward Processing to compute sequence.

x_j. For instance consider the Fibonacci sequence $x_0 = 0,\ x_1 = 1,\ x_i = x_{i-1} + x_{i-2}\ \forall i \geq 2$ in which each node v_i with $i \geq 2$ depends on exactly its two predecessors (see Fig. 1). Here, a linear scan for increasing i suffices to solve the dependencies.

In general, an algorithm needs to traverse G according to some topological order $\prec_T$ of nodes V and also has to ensure that each v_j can access values from all v_i with $(v_i, v_j) \in E$. The *TFP* technique achieves this as follows: as soon as x_i has been calculated, messages of the form $\langle v_j, x_i \rangle$ are sent to all successors $(v_i, v_j) \in E$. These messages are kept in a minimum priority queue sorting the items by their recipients according to $\prec_T$. By construction, the algorithm only starts the computation v_i once all predecessors $v_j \prec_T v_i$ are completed. Since these predecessors already removed their messages from the PQ, items addressed to v_i (if any) are currently the smallest elements in the data structure and can be dequeued. Using a suited EM PQ [2], TFP incurs $\mathcal{O}(\text{sort}(k))$ I/Os, where k is the number of messages sent.

3 Randomisation schemes

Here, we summarise the randomisation schemes ESMC [24] and Curveball for simple undirected graphs [5], and then discuss the notion of global trades. Since these algorithms iteratively modify random parts of a graph, they can be analysed as finite Markov chains. It is well known that any finite, irreducible, aperiodic, and symmetric Markov chain converges to the uniform distribution on its state space (e.g. [20]). Its *mixing time* indicates the number of steps necessary to reach the stationary distribution.

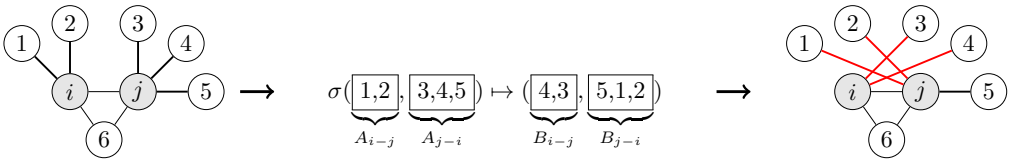
3.1 Edge-Switching

ESMC is a state-of-the-art randomisation method with a wide range of applications, e.g. the generation of graphs [15, 19], or the randomisation of biological datasets [16]. In each step, *ESMC* chooses two edges $e_1 = [u_1, v_1], e_2 = [u_2, v_2]$ and a direction $d \in \{0, 1\}$ uniformly at random and rewires them into $\{u_1, u_2\}, \{v_1, v_2\}$ if d=0 and $\{u_1, v_2\}, \{v_1, u_2\}$ otherwise. If a step yields a non-simple graph, it is skipped. *ESMC*'s Markov chain is irreducible [10], aperiodic and symmetric [23] and hence converges to the uniform distribution on the space of simple graphs with fixed degree sequence. While analytic bounds on the mixing time [12, 13] are impractical, usually a number of steps linear in the number of edges is used in practice [29].

3.2 Simple Undirected Curveball algorithm

Curveball is a novel randomisation method. In each step, two nodes trade their neighbourhoods, possibly yielding faster mixing times [5, 35, 36].

▶ **Definition 1** (Simple Undirected Trade). Let $G = (V, E)$ be a simple graph, A be its adjacency list representation, and A_u be the set of neighbours of node u. A trade $t = (i, j, \sigma)$



$A_i = \{1, 2, 6, j\}$
$A_j = \{3, 4, 5, 6, i\}$

$B_{i-j} = \{3, 4\}$
$B_{j-i} = \{1, 2, 5\}$

$B_i = \{3, 4, 6, j\}$
$B_j = \{1, 2, 5, 6, i\}$

Figure 2 The trade (i, j, σ) between nodes i and j only considers edges to the disjoint neighbours $\{1, \ldots, 5\}$. For the reassigned disjoint neighbours we use the short-hand $B_{i-j} := \{x \mid x \in D_{ij}, \mathrm{rank}_\sigma(x) \leq |A_{i-j}|\}$ and $B_{j-i} := \{x \mid x \in D_{ij}, \mathrm{rank}_\sigma(x) > |A_{i-j}|\}$. The triangle $(i, j, 6)$ is omitted as trading any of its edges would either introduce parallel edges, self loops, or result in no change at all. Then, the given σ exchanges four edges.

from A to adjacency list B is defined by two nodes i and j, and a permutation $\sigma\colon D_{ij} \to D_{ij}$ where $A_{i-j} := A_i \setminus (A_j \cup \{j\})$ and $D_{ij} := A_{i-j} \cup A_{j-i}$. As shown in Fig. 2, performing t on G results in $B_i = (A_i \backslash A_{i-j}) \cup \{x \mid x \in D_{ij}, \mathrm{rank}_\sigma(x) \leq |A_{i-j}|\}$ and $B_j = (A_j \backslash A_{j-i}) \cup \{x \mid x \in D_{ij}, \mathrm{rank}_\sigma(x) > |A_{i-j}|\}$. Since edges are undirected, symmetry has to be preserved: for all $u \in A_i \backslash B_i$ the label j in adjacency list B_u is changed to i and analogously for $A_j \setminus B_j$.

Simple Undirected Curveball randomises a graph by repeatedly selecting a node pair $\{i, j\}$ and permutation σ on the disjoint neighbours uniformly at random. Its Markov chain is irreducible, aperiodic and symmetric and hence converges to the uniform distribution [6].

3.3 Undirected Global Trades

Trade sequences typically consist of pairs in which each constituent is drawn uniformly at random. While it is a well-known fact[3] that $\Theta(n \log n)$ trades are required in expectation until each node is included at least once, there is no apparent reason why this should be beneficial; in fact, experiments in section 5 suggest the contrary.

Carstens et al. propose the notion of *global trades* for directed or bipartite graphs as a 2-partition of all nodes implicitly forming $n/2$ node pairs to be traded in a single step [6]. This concept is not applicable to undirected graphs where in general the two directions (u, v) and (v, u) of an edge $\{u, v\}$ cannot be processed independently in a single step. We hence extend global trades to undirected graphs by interpreting them as a sequence of $n/2$ single trades which together target each node exactly once (we assume n to be even; if this is not the case we add an isolated node). Dependencies are then resolved by the order of this sequence.

▶ **Definition 2** (Undirected Global trade). Let $G = (V, E)$ be a simple graph and $\pi\colon V \to V$ be a permutation on the set of nodes. A *global trade* $T = (t_1, \ldots, t_\ell)$ for $\ell = \lfloor n/2 \rfloor$ is a sequence of trades $t_i = \{\pi(v_{2i-1}), \pi(v_{2i}), \sigma_i\}$. By applying T to G we mean that the trades $t_1, \ldots, t_\ell$ are applied successively starting with G.

Theorem 3 allows us to use global trades as a substitute for a sequence of single trades, as global trades preserve the stationary distribution of Curveball's Markov chain. The proof extends [6], which shows convergence of global trades in bipartite or directed graphs, to undirected graphs and uses similar techniques.

[3] For instance studied as the coupon collector problem.

▶ **Theorem 3.** *Let $G = (V, E)$ be an arbitrary simple undirected graph, and let Ω_G be the set of all simple directed graphs that have the same degree sequence as G. The Curveball algorithm with global trades and started at G converges to the uniform distribution on Ω_G.*

Proof. In order to prove the claim, we have to show irreducibility and aperiodicity of the Markov chain as well as symmetry of the transition probabilities.

For the first two properties it suffices to show that whenever there exists a single trade from state A to B, there also exists a global trade from A to B (see [4] for a similar argument).[4] Observe that there is a non-zero probability that a single trade does not change the graph, e.g. by selecting σ_i as the identity. Hence there is a non-zero probability that ...

- a global trade does not alter the graph at all. This corresponds to a self-loop at each state of the Markov chain and hence guarantees aperiodicity.
- all but one single trade of a global trade do not alter the graph. In this case, a global trade degenerates to a single trade and the irreducibility shown in [4] carries over.

It remains to show that the transition probabilities are symmetric. Let $\mathcal{T}^g_{AB}$ be the set of global trades that transform state A to state B. Then the transition probability between A and B equals the sum of probabilities of selecting a trade sequence from $\mathcal{T}^g_{AB}$. That is $P_{AB} = \sum_{T \in \mathcal{T}^g_{AB}} \mathbf{P}_A(T)$ where $\mathbf{P}_A(T)$ denotes the probability of selecting global trade T in state A.

The probability $\mathbf{P}_A(t)$ of selecting a single trade $t = (i, j, \sigma)$ from state A to state B equals the probability $\mathbf{P}_B(\tilde{t})$ of selecting the reverse trade $\tilde{t} = (i, j, \sigma^{-1})$ from state B to A [6]. We now define the reverse global trade of $T = (t_1, \ldots, t_\ell)$ as $\tilde{T} = (\tilde{t}_\ell, \ldots, \tilde{t}_1)$. It is straight-forward to check that this gives a bijection between the sets $\mathcal{T}^g_{AB}$ and $\mathcal{T}^g_{BA}$.

It remains to show that the middle equality holds in

$$P_{AB} = \sum\nolimits_{T \in \mathcal{T}^g_{AB}} \mathbf{P}_A(T) \overset{!}{=} \sum\nolimits_{\tilde{T} \in \mathcal{T}^g_{BA}} \mathbf{P}_B(\tilde{T}) = P_{BA}.$$

Let $T = (t_1, \ldots, t_\ell)$ be a global trade from A to B as implied by π and $A{=}A_1, \ldots, A_{\ell+1}{=}B$ be the intermediate states. We denote the reversal of T and π as $\tilde{T}$ and $\tilde{\pi}$ respectively and obtain $P_A(T) = \mathbf{P}(\pi)\mathbf{P}_{A_1}(t_1) \ldots \mathbf{P}_{A_\ell}(t_\ell) = \mathbf{P}(\tilde{\pi})\mathbf{P}_B(\tilde{t}_\ell) \ldots \mathbf{P}_{A_2}(\tilde{t}_1) = P_B(\tilde{T})$. Clearly $\mathbf{P}(\pi) = \mathbf{P}(\tilde{\pi})$ as we are picking permutations uniformly at random. The second equality follows from $\mathbf{P}_A(t) = \mathbf{P}_B(\tilde{t})$ for a single trade between A and B. ◀

4 Novel Curveball algorithms for undirected graphs

In this section we present the related algorithms *EM-CB*, *IM-CB*, *EM-GCB* and *EM-PGCB*. The algorithms receive a simple graph G and a *trade sequence* $T = [\{u_i, v_i\}]_{i=1}^{\ell}$ as input and compute the result of carrying out the trade sequence T (see section 3.2) in order.

EM-CB and *IM-CB* are sequential solutions suited to process arbitrary trade sequences T. For our analysis, we assume T's constituents to be drawn uniformly at random (as expected in typical applications). Both algorithms share a common design, but differ in the data structures used. *EM-CB* is an I/O-efficient algorithm while *IM-CB* is optimised for small graphs allowing for unstructured accesses to main memory. In contrast, *EM-GCB* and *EM-PGCB* process global trades only. This restricted input model allows us to represent the trade sequence T implicitly by hash functions which further accelerates trading.

[4] Since each global trade can be emulated by its $n/2$ decomposed single trades, the reverse is true for a hop of $n/2$ single trade steps. Due to dependencies however the transition probabilities generally do not match, see $V = \{1, 2, 3, 4\}$ and $E = \{[1, 2], [3, 4]\}$ for a simple counterexample.

Algorithm 1: *EM-CB.*

```
   Data: Trade sequence T, simple graph G = (V, E) by edge list E
   // Preprocessing: Compute Dependencies
 1 foreach trade t_i = (u, v) ∈ T for increasing i do
 2 |   Send messages ⟨u, t_i⟩ and ⟨v, t_i⟩ to Sorter SorterTtoV
 3 Sort SorterTtoV lexicographically        // All trades of a node are next to each other
 4 foreach node u ∈ V do
 5 |   Receive S(u) = [t_1, ..., t_k] from k messages addressed to u in SorterTtoV
 6 |   Set t_{k+1} ← ∞                                  // t_1 = ∞ iff u is never active
 7 |   Send ⟨t_i, u, t_{i+1}⟩ to SorterDepChain for i ∈ [k]
 8 |   foreach directed edge (u, v) ∈ E do
 9 |   |   if u < v then
10 |   |   |   Send message ⟨v, u, t_1⟩ via PqVtoV
11 |   |   else
12 |   |   |   Receive t_1^v from unique message received via PqVtoV
13 |   |   |   if t_1 ≤ t_1^v then  Send message ⟨t_1, u, v, t_1^v⟩ via PqTtoT
   |   |   |   else                 Send message ⟨t_1^v, v, u, t_1⟩ via PqTtoT
14 Sort SorterDepChain
   // Main phase - Currently at least the first trade has all information it needs
15 foreach trade t_i = (u, v) ∈ T for increasing i do
16 |   Receive successors τ(u) and τ(v) via SorterDepChain
17 |   Receive neighbours A_G(u), A_G(v) and their successors τ(·) from PqTtoT
18 |   Randomly reassign disjoint neighbours, yielding new neighbours A'_G(u) and A'_G(v).
19 |   foreach (a, b) ∈ ({u} × A'_G(u)) ∪ ({v} × A'_G(v)) do
   |   |   if τ_a = ∞ and τ_b = ∞ then  Output final edge {a, b}
20 |   |   else if τ_a ≤ τ_b then       Send message ⟨τ_a, a, b, τ_b⟩ via PqTtoT
   |   |   else                         Send message ⟨τ_b, b, a, τ_a⟩ via PqTtoT
```

At core, all algorithms perform trades in a similar fashion: In order to carry out the i-th trade $\{u_i, v_i\}$, they retrieve the neighbourhoods $\mathcal{A}_{u_i}$ and $\mathcal{A}_{v_i}$, shuffle[5] them, and then update the graph. Once the neighbourhoods are known, trading itself is straight-forward. We compute the set of disjoint neighbours $D = (\mathcal{A}_{u_i} \cup \mathcal{A}_{v_i}) \setminus (\mathcal{A}_{u_i} \cap \mathcal{A}_{v_i})$ and then draw $|\mathcal{A}_{u_i} \cap D|$ nodes from D for u_i uniformly at random while the remaining nodes go to v_i. If $\mathcal{A}_{u_i}$ and $\mathcal{A}_{v_i}$ are sorted this requires only $\mathcal{O}(|\mathcal{A}_{u_i}| + |\mathcal{A}_{v_i}|)$ work and scan$(|\mathcal{A}_{u_i}| + |\mathcal{A}_{v_i}|)$ I/Os (see also proof of Lemma 6 if the neighbourhoods fit into RAM). Hence we focus on the harder task of obtaining and updating the adjacency information.

4.1 EM-CB: A sequential I/O-efficient Curveball algorithm

EM-CB is an I/O-efficient Curveball algorithm to randomise undirected graphs as detailed in Alg. 1. This basic algorithm already contains crucial design principles which we further explore with *IM-CB*, *EM-GCB* and *EM-PGCB* in sections 4.2 and 4.4 respectively.

The algorithm encounters the following challenges. After an undirected trade $\{u, v\}$ is carried out, it does not suffice to only update the neighbourhoods $\mathcal{A}_u$ and $\mathcal{A}_v$: consider the case that edge $\{u, x\}$ changes into $\{v, x\}$. Then this switch also has to be reflected in the neighbourhood of $\mathcal{A}_x$. Here, we call u and v *active* nodes while x is a *passive* neighbour.

[5] In contrast to Definition 2, we do not consider the permutation σ of disjoint neighbours as part of the input, but let the algorithm choose one randomly for each trade. We consider this design decision plausible as the set of disjoint neighbours only emerges over the course of the execution.

In the EM setting another challenge arises for graphs exceeding main memory; it is prohibitively expensive to directly access the edge list since this unstructured pattern triggers $\Omega(1)$ I/Os for each edge processed with high probability.

EM-CB approaches these issues by abandoning a classical static graph data structure containing two redundant copies of each edge. Following the *TFP* principle, we rather interpret all trades as a sequence of points over time that are able to receive messages. Initially, we send each edge to the earliest trade one of its endpoints is active in.[6] This way, the first trade receives one message from each neighbour of the active nodes and hence can reconstruct $\mathcal{A}_{u_1}$ and $\mathcal{A}_{v_1}$. After shuffling and reassigning the disjoint neighbours, *EM-CB* sends each resulting edge to the trade which requires it next. If no such trade exists, the edge can be finalised by committing it to the output.

The algorithm hence requires for each (actively or passively) traded node u, the index of the next trade in which u is actively processed. We call this the *successor* of u and define it to be ∞ if no such trade exists. The dependency information is obtained in a preprocessing step; given $T = [\,\{u_i, v_i\}\,]_{i=1}^{\ell}$, we first compute for each node u the monotonically increasing index list $\mathcal{S}(u)$ of trades in which u is actively processed, i.e. $\mathcal{S}(u) := \big[\, i \,|\, u \in t_i \text{ for } i \in [\ell] \,\big] \circ [\infty]$.

▶ **Example 4.** Let $G = (V, E)$ be a simple graph with $V = \{v_1, v_2, v_3, v_4\}$ and trade sequence $T = [t_1\colon \{v_1, v_2\}, t_2\colon \{v_3, v_4\}, t_3\colon \{v_1, v_3\}, t_4\colon \{v_2, v_4\}, t_5\colon \{v_1, v_4\}]$. Then, the successors $\mathcal{S}$ follow as $\mathcal{S}(v_1) = [1, 3, 5, \infty]$, $\mathcal{S}(v_2) = [1, 4, \infty]$, $\mathcal{S}(v_3) = [2, 3, \infty]$, $\mathcal{S}(v_4) = [2, 4, 5, \infty]$.

This information is then spread via two channels:

- After preprocessing, *EM-CB* scans $\mathcal{S}$ and T conjointly and sends $\langle t_i, u_i, t_i^u \rangle$ and $\langle t_i, v_i, t_i^v \rangle$ to each trade t_i. The messages carry the successors t_i^u and t_i^v of the trade's active nodes.
- When sending an edge as described before, we augment it with the successor of the passive node. Initially, this information is obtained by scanning the edge list E and $\mathcal{S}$ conjointly. Later, it can be inductively computed since each trade receives the successors of all nodes involved.

▶ **Lemma 5.** *For an arbitrary trade sequence T of length ℓ, EM-CB has a worst-case I/O complexity of $\mathcal{O}[\mathrm{sort}(\ell) + \mathrm{sort}(n) + \mathrm{scan}(m) + \ell d_{\max}/B \log_{M/B}(m/B)]$. For r global trades, the worst case I/O complexity is $\mathcal{O}(r[\mathrm{sort}(n) + \mathrm{sort}(m)])$.*

Proof. Refer to the full article [7] for the proof. ◀

4.2 IM-CB: An internal memory version of EM-CB

While *EM-CB* is well-suited if memory access is a bottleneck, we also consider the modified version *IM-CB*. As shown in section 5, *IM-CB* is typically faster for small graph instances. *IM-CB* uses the same algorithmic ideas as *EM-CB* but replaces its priority queues and sorters[7] by unstructured I/O into main memory (see [7] for details):

- Instead of sending neighbourhood information in a TFP-fashion, we now rely on a classical adjacency vector data structure $\mathcal{A}_G$ (an array of arrays). Similarly to *EM-CB*, we only

[6] If an edge connects two nodes that are both actively traded we implicitly perform an arbitrary tie-break.

[7] The term *sorter* refers to a container with two modes of operation: in the first phase, items are pushed into the write-only sorter in an arbitrary order by some algorithm. After an explicit switch, the filled data structure becomes read-only and the elements are provided as a lexicographically non-decreasing stream which can be rewound at any time. While a sorter is functionally equivalent to filling, sorting and reading back an EM vector, the restricted access model reduces constant factors in the implementation's runtime and I/O-complexity [3].

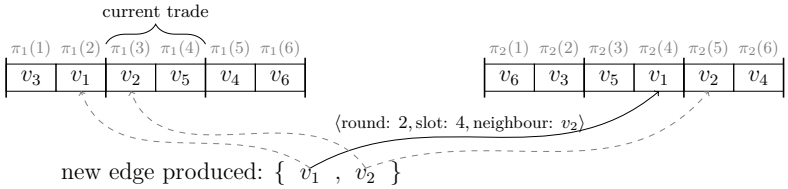


Figure 3 During the trade $j{=}1, i_1{=}3, i_2{=}4$ the edge $\{v_1, v_2\}$ is produced; the arrows indicate positions considered as successors. Since v_1 and v_2 are already processed in round $j{=}1$, π_2 is used to compute the successor. Then, the message is sent to v_1 in round 2 as v_1 is processed before v_2.

keep one directed representation of an undirected edge. As an invariant, an edge is always placed in the neighbourhood of the incident node traded before the other. To speed-up these insertions, *IM-CB* maintains unordered neighbourhood buffers.

- *IM-CB* does not forward successor information, but rather stores $\mathcal{S}$ in a contiguous block of memory. The algorithm additionally maintains the vector $\mathcal{S}_{\text{idx}}[1 \dots n]$ where the i-th entry points to the current successor of node v_i. Once this trade is reached, the pointer is incremented giving the next successor.

▶ **Lemma 6.** *For a random trade sequence T of length ℓ, IM-CB has an expected running time of $\mathcal{O}(n + \ell + m + \ell m/n)$. In the case of r many global trades (each consisting of $n/2$ normal trades) the running time is given by $\mathcal{O}(n + rm)$.*

Proof. Refer to the full article [7] for the proof. ◀

4.3 EM-GCB: An I/O-efficient Global Curveball algorithm

EM-GCB builds on *EM-CB* and exploits the regular structure of global trades to simplify and accelerate the dependency tracking. As discussed in section 3.3, a global trade can be encoded as a permutation $\pi\colon [n] \to [n]$ by interpreting adjacent ranks as trade pairs, i.e. $T_\pi = [\,\{v_{\pi(2i-1)}, v_{\pi(2i)}\}\,]_{i=1}^{n/2}$. In this setting, a sequence of global trades is given by r permutations $[\,\pi_j\,]_{j=1}^{r}$. The model simplifies dependencies as it is not necessary to explicitly gather $\mathcal{S}$ and communicate successors.

As illustrated in Fig. 3, we also change the addressing scheme of messages. While *EM-CB* sends messages to specific nodes in specific trades, *EM-GCB* exploits that each node v_i is actively traded only once in each round j and hence can be addressed by its position $\pi_j(i)$. Successors can then be computed in an ad hoc fashion; let a trade of adjacent positions $i_1 < i_2$ of the j-th global trade produce (amongst others) the edge $\{v_x, v_y\}$. The successor of v_x (and analogously the one of v_y) is $\mathcal{S}_{j,i_2}[v_x] = (j, \pi_j(x))$ if v_x is processed later in round j (i.e. $\pi_j(x)/2 > i_2$) and otherwise $\mathcal{S}_{j,i_2}[v_x] = (j{+}1, \pi_{j+1}(x))$. Here we imply an untraded additional function $\pi_{r+1}(x) = x$ which avoids corner cases and generates an ordered edge list as a result of the r-th global trade.

To reduce the computational cost of the successor computation, *EM-GCB* supports fast injective functions $f\colon X \to Y$ where $[n] \subseteq X$ and $[n] \subseteq Y$. In contrast to the original permutations, their relevant image $\{\, f(x) \mid x \in [n] \,\}$ may contain gaps which are simply skipped by *EM-GCB*. This requires minor changes in the addressing scheme.

In practice, we use functions from the family of linear congruential maps $H_p := \{\, h_{a,b} \mid 1 \le a < p \text{ and } 0 \le b < p \,\}$ with $h_{a,b}(x) \equiv [(ax + b) \mod p]$ where p is the smallest prime number $p \ge n$. Random choices from H_p are well suited for *EM-GCB* since they

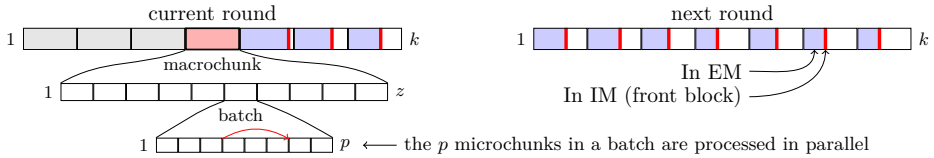


Figure 4 *EM-PGCB* splits each global trade into k *macrochunks* and maintains an external memory queue for each. Before processing a macrochunk, the buffer is loaded into IM and sorted, and further subdivided into z batches each consisting of p microchunks. A type (ii) message is visualised by the red intra-batch arrow.

are 2-universal[8] and contain only $\mathcal{O}(\log(n))$ gaps (see [7] for details). They are also bijections with an easily computable inverse $h_{a,b}^{-1}$ that allows *EM-GCB* to determine the active node $h_{a,b}^{-1}(i)$ traded at position i; this operation is only performed once for each traded position. *EM-GCB* also supports non-invertible functions. This can be implemented with messages $\langle h(i), i\rangle$ that are generated for $1 \leq i \leq n$ and delivered using TFP.

4.4 EM-PGCB: An I/O-efficient parallel Global Curveball algorithm

EM-PGCB adds parallelism to *EM-GCB* by concurrently executing multiple sequential trades. As in Fig. 4, we split a global trade into *microchunks* each containing a similar number of node pairs and then execute a *batch* of p such subdivisions in parallel. The batch's size is a compromise between intra-batch dependencies (messages are awaited from another processor) and overhead caused by synchronising threads at the batch's end (see [7] for details).

EM-PGCB processes each microchunk similarly as in *EM-CB* but differentiates between messages that are sent (i) within a microchunk, (ii) between microchunks of the same batch (iii) and microchunks processed later. Each class is transported using an optimised data structure (see below) and only type (ii) messages introduce dependencies between parallel executions and are resolved as follows: each processor retrieves the messages that are sent to its next trade and checks whether all information required is available by comparing the number of messages to the active nodes' degrees. If data is missing the trade is skipped and later executed by the processor that adds the last missing neighbour.

For graphs with $m = \mathcal{O}(M^2/B)$ edges[9], we optimise the communication structure for type (iii) messages. Observe that *EM-PGCB* sends messages only to the current and the subsequent round. We partition a round into k *macrochunks* each consisting of $\Theta(n/k)$ contiguous trades. An external memory queue is used for each macrochunk to buffer messages sent to it; in total, this requires $\Theta(kB)$ internal memory. Before processing a macrochunk, all its messages are loaded into IM, subsequently sorted and arranged such that missing messages can be directly placed to the position they are required in. This can also be overlapped with the processing of the previous macrochunk. The number k of macrochunks should be as small as possible to reduce overheads, but sufficiently large such that all messages of a macrochunk fit into main memory (see [7] for details).

▶ **Theorem 7.** *EM-PGCB requires* $\mathcal{O}(r \cdot [\mathrm{sort}(n) + \mathrm{sort}(m)])$ *I/Os to perform* r *global trades.*

Proof. Observe that we can analyse each of the r rounds individually. A constant amount of auxiliary data is needed per node to provision gaps for missing data, to detect whether a

[8] i.e. given one node in a single trade, the other is uniformly chosen among the remaining nodes.
[9] Even with as little as 1 GiB of internal memory, several billion edges are supported.

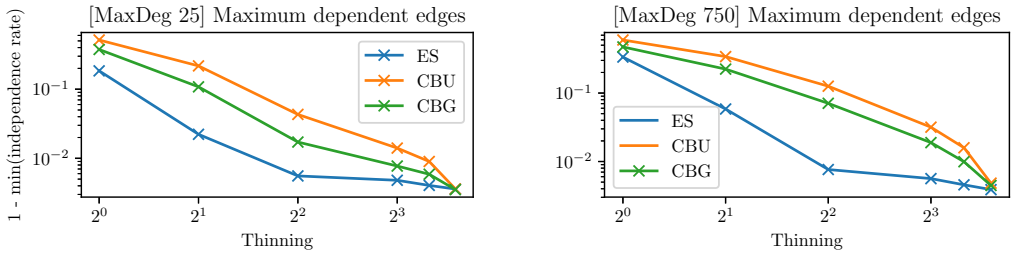


Figure 5 Fraction of edges still correlated as a function of the thinning parameter k for graphs with $n = 2{\cdot}10^3$ nodes and degree distribution $\textsc{Pld}([a, b), \gamma)$ with $\gamma = 2$, $a = 5$, and $b \in \{25, 750\}$. The (not thinned) long Markov chains of edge switching (ES), uniform Curveball (CBU) and global Curveball (CBG) contain 6000 super steps each.

trade can be executed and (if required) to invert the permutation. This accounts for $\Theta(n)$ messages requiring sort(n) I/Os to be delivered. Using an ordinary PQ, the analysis of *EM-CB* (see Lemma 5) carries over, requiring sort(m) I/Os for a global trade. ◀

5 Experimental Evaluation

In this section we evaluate the quality of the proposed algorithms and analyse the runtime of our C++ implementations.[10] *EM-CB*, *IM-CB*, *EM-GCB* are designed as modules of NetworKit [33]; due to their superior performance, only the latter two were added to the library and are available since release 4.6. *EM-PGCB*'s implementation is developed separately and facilitates external memory data structures and algorithms of STXXL [9].

Intuitively, graphs with skewed degree distributions are hard instances for Curveball since it shuffles and reassigns the disjoint neighbours of two trading nodes. Hence, limited progress is achieved if a high-degree node trades with a low-degree node. Since our experiments support this hypothesis, we focus on graphs with powerlaw degree distributions as difficult but highly relevant graph instances. Our experiments use two parameter sets:

- *(lin)* – The maximal possible degree scales linearly as a function of the number n of nodes. The degree distribution $\textsc{Pld}([a, b), \gamma)$ is chosen as $a = 10$, $b = n/20$ and $\gamma = 2$.
- *(const)* – The extremal degrees are kept constant. In this case the parameters are chosen as $a = 50$, $b = 10000$ and $\gamma = 2$.

We select these configurations to be comparable with [15] where both parameter sets are used to evaluate *EM-ES*. The first setting *(lin)* considers the increasing average degree of real-world networks as they grow. The second setting *(const)* approximates the degree distribution of the Facebook network in May 2011 (refer to [14] for details). Runtimes are measured on the following off-the-shelf machine: Intel Xeon E5-2630 v3 (8 cores at 2.40GHz), 64GB RAM, 2× Samsung 850 PRO SATA SSD (1 TB), Ubuntu Linux 16.04, GCC 7.2.

5.1 Mixing of Edge-Switching, Curveball and Global Curveball

We are not aware of any practical theoretical bounds on the mixing time of Markov chains of Curveball, Global Curveball or edge switching (see section 3). Hence, we quantitatively study the progress made by Curveball trades compared to edge switching and approximate the

[10] Code used for the presented benchmarks can be found at our fork `https://github.com/hthetran/networkit` (*IM-CB* and *EM-CB*) and `https://github.com/massive-graphs/extmem-lfr` (*EM-PGCB*).

mixing time of the underlying Markov chains by a method developed in [30]. This criterion is a more sensitive proxy to the mixing time than previously used alternatives, such as the local clustering coefficient, triangle count and degree assortativity [14].

Intuitively, one determines the number of Markov chain steps required until the correlation to the initial state decays. Starting from an initial graph G_0, the Markov chain is executed for a large number of steps, yielding a sequence $(G_t)_{t\geq 0}$ of graphs evolving over time. For each occurring edge e, we compute a boolean vector $(Z_{e,t})_{t\geq 0}$ where a 1 at position t indicates that e exists in graph G_t. We then derive the *k-thinned* series $(Z^k_{e,t})_{t\geq 0}$ only containing every k-th entry of the original vector $(Z_{e,t})_{t\geq 0}$ and use k as a proxy for the mixing time.

To determine if k Markov chain steps suffice for edge e to lose the correlation to the initial graph, the empirical transition probabilities of the k-thinned series $(Z^k_{e,t})_{t\geq 0}$ are fitted to both an independent and a Markov model respectively. If the independent model is a better fit, we deem edge e to be independent.

The results presented here consider only small graphs due to the high computational cost involved. However, additional experiments suggest that the results hold for graphs at least one order of magnitude larger which is expected as powerlaw distributions are scale-free.

We compare a sequence of uniform (single) trades, global trades and edge switching and visually align the results of these schemes by defining a *super step*. Depending on the algorithm a super step corresponds to either a single global trade, $n/2$ uniform trades or m edge-swaps. Comparing $n/2$ uniform trades with a global trade seems sensible since a global trade consists of exactly $n/2$ single trades, furthermore randomising with $n/2$ single trades considers the state of $2m$ edges which is also true for m edge-swaps. The alignment accounts for the fact that a single Curveball Markov chain step may execute multiple neighbour switches, thus easily outperforming *ESMC* in a step-by-step comparison.

Fig. 5 contains a selection of results obtained for small powerlaw graph instances using this method (see [7] for the complete dataset). Progress is measured by the fraction of edges that are still classified as correlated, i.e. the faster a method approaches zero the better the randomisation. We omit an in-depth discussion of uniform trades and rather focus on global trades which consistently outperform the former (cf. section 3.2).

In all settings *ESMC* shows the fastest decay. The gap towards global trades growths temporarily as the maximal degree is increased which is consistent with our initial claim that skewed degree distributions are challenging for Curveball. The effect is however limited and in all cases performing 4 global trades for each edge switching super step gives better results. This is a pessimistic interpretation since typically $10m$ to $100m$ edge switches are used to randomise graphs in practice; in this domain global trades perform similarly well and 20 global trades consistently give at least the quality of $10m$ edge switches.

5.2 Runtime performance benchmarks

We measure the runtime of the algorithms proposed in section 4 and compare them to two state-of-the-art edge switching schemes (using the authors' C++ implementations):

- *VL-ES* is a sequential IM algorithm with a hashing-based data structure optimised for efficient neighbourhood queries and updates [37]. To achieve comparability, we removed connectivity tests, fixed memory management issues, and adopted the number of swaps.
- *EM-ES* is an EM edge switching algorithm and part of *EM-LFR*'s toolchain [15].

We carry out experiments using the *(const)* and *(lin)* parameter sets, and limit the problem sizes for internal memory algorithms to avoid exhaustion of the main memory. For

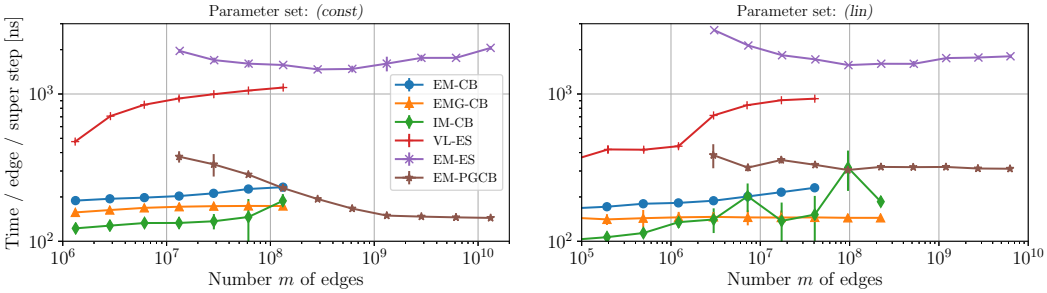


Figure 6 Runtime per edge and super step (global trade or m edge swaps) of the proposed algorithms *IM-CB*, *EM-CB* and *EM-PGCB* compared to state-of-the-art IM edge switching *VL-ES* and EM edge switching *EM-ES*. Each data point is the median of $S \geq 5$ runs over 10 super steps each. The left plot contains the (const)-parameter set, the right one (lin). Observe that the super steps of different algorithms advance the randomisation process at different speeds (see discussion).

each data point we carry out 10 super steps (i.e. 10 global trades or $10m$ edge swaps) on a graph generated with Havel-Hakimi from a random powerlaw degree distribution.

Figure 6 presents the walltime per edge and super step including pre-computation[11] required by the algorithms but excluding the initial graph generation process. The plots include (mostly small) errorbars corresponding to the unbiased estimation of the standard deviation of S repetitions per data point (with different random seeds).

The number k of macrochunks does not significantly affect *EM-PGCB*'s performance for small graphs due to comparably high synchronisation cost. In contrast, adjusting k for larger graphs can noticeably increase the performance of *EM-PGCB*. We thus experimentally determined the value $k = 32$ for both *(const)* and *(lin)* with $n = 10^7$ nodes and use that value for all other instances.

All Curveball algorithms outperform their direct competitors significantly – even if we pessimistically executed two global trades for each edge switching super step (see section 5.1). For large instances of *(const)* *EM-PGCB* carries out one super step 14.3 times faster than *EM-ES* and 5.8 times faster for *(lin)*. *EM-PGCB* also shows a superior scaling behaviour with an increasing speed-up for larger graphs. Similarly, *IM-CB* processes super steps up to 6.3 times faster than *VL-ES* on *(const)* and 5.1 times on *(lin)*.

On our test machine, the implementation of *IM-CB* outperforms *EM-CB* in the internal memory regime; *EM-GCB* is faster for large graphs. As indicated in [7], this changes qualitatively for machines with slower main memory and smaller cache; on such systems the unstructured I/O of *IM-CB* and *VL-ES* is more significant rendering *EM-CB* and *EM-GCB* the better choice with a speed-up factor exceeding 8 compared to *VL-ES*.

6 Conclusion and outlook

We applied *global* Curveball trades to undirected graphs simplifying the algorithmic treatment of dependencies and showed that the underlying Markov chain converges to a uniform distribution. Experimental results show that global trades yield an improved quality compared to a sequence of uniform trades of the same size.

[11] For *VL-ES* we report only the swapping process and the generation of the internal data structures.

We presented *IM-CB* and *EM-CB*, the first efficient algorithms for Simple Undirected Curveball algorithms; they are optimised for internal and external memory respectively. Our I/O-efficient parallel algorithm *EM-PGCB* exploits the properties of global trades and executes a super step 14.3 times faster than the state-of-the-art edge switching algorithm *EM-ES*; for *IM-CB* we demonstrate speed-ups of up to 6.3 (in a conservative comparison the speed-ups should be halved to account for the differences in mixing times of the underlying Markov chains). The implementations of all three algorithms are freely available and are in the process of being incorporated into *EM-LFR* and considered for NetworKit.

References

1 A. Aggarwal, J. Vitter, et al. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.

2 L. Arge. *The buffer tree: A new technique for optimal I/O-algorithms*, pages 334–345. Springer Berlin Heidelberg, 1995. doi:10.1007/3-540-60220-8_74.

3 A. Beckmann, R. Dementiev, and J. Singler. Building a parallel pipelined external memory algorithm library. In *IPDPS'09*, 2009. doi:10.1109/IPDPS.2009.5161001.

4 C. J. Carstens. Proof of uniform sampling of binary matrices with fixed row sums and column sums for the fast curveball algorithm. *Physical Review E*, 91:042812, 2015.

5 C. J. Carstens. *Topology of Complex Networks: Models and Analysis*. PhD thesis, RMIT University, January 2016.

6 C. J. Carstens, A. Berger, and G. Strona. Curveball: a new generation of sampling algorithms for graphs with fixed degree sequence. *CoRR*, 2016. arXiv:1609.05137.

7 C. J. Carstens, M. Hamann, U. Meyer, M. Penschuck, H. Tran, and D. Wagner. Parallel and I/O-efficient randomisation of massive networks using global curveball trades. *CoRR*, abs/1804.08487, 2018.

8 G. W. Cobb and Y.-P. Chen. An application of markov chain monte carlo to community ecology. *The American Mathematical Monthly*, 110(4):265–288, 2003.

9 R. Dementiev, L. Kettner, and P. Sanders. STXXL: standard template library for XXL data sets. *Software: Practice and Experience*, 38(6):589–637, 2008. doi:10.1002/spe.844.

10 R. B. Eggleton and D. A. Holton. *Simple and multigraphic realizations of degree sequences*, pages 155–172. Springer Berlin Heidelberg, 1981. doi:10.1007/BFb0091817.

11 P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 1959.

12 C. Greenhill. A polynomial bound on the mixing time of a markov chain for sampling regular directed graphs. *The Electronic Journal of Combinatorics*, 18(1):P234, 2011.

13 C. Greenhill. The switch markov chain for sampling irregular graphs: Extended abstract. In *Proceedings of SODA '15*, pages 1564–1572, 2015.

14 M. Hamann, U. Meyer, M. Penschuck, H. Tran, and D. Wagner. I/O-efficient generation of massive graphs following the LFR benchmark. *CoRR*, 2017. arXiv:1604.08738.

15 M. Hamann, U. Meyer, M. Penschuck, and D. Wagner. I/O-efficient generation of massive graphs following the LFR benchmark. In *ALENEX*, 2017. doi:10.1137/1.9781611974768.

16 F. Iorio, M. Bernardo-Faura, A. Gobbi, T. Cokelaer, G. Jurman, and J. Saez-Rodriguez. Efficient randomization of biological networks while preserving functional characterization of individual nodes. *BMC bioinformatics*, 17(1):542, 2016.

17 S. Itzkovitz, R. Milo, N. Kashtan, G. Ziv, and U. Alon. Subgraphs in random networks. *Physical review E*, 68:026127, Aug 2003. doi:10.1103/PhysRevE.68.026127.

18 A. Lancichinetti and S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, 80:016118, Jul 2009. doi:10.1103/PhysRevE.80.016118.

19 A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78:046110, 2008. doi:10.1103/PhysRevE.78.046110.

20 D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, Providence, Rhode Island, 2009.

21 A. Maheshwari and N. Zeh. *A Survey of Techniques for Designing I/O-Efficient Algorithms*, pages 36–61. Springer Berlin Heidelberg, 2003.

22 U. Meyer, P. Sanders, and J. Sibeyn. *Algorithms for Memory Hierarchies: Advanced Lectures*. Springer Berlin Heidelberg, 2003. doi:10.1007/3-540-36574-5.

23 C. G. M. Mihail and E. Zegura. The markov chain simulation method for generating connected power law random graphs. In *Proceedings of ALENEX '03*. SIAM, 2003.

24 R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *CoRR*, 2003. arXiv:cond-mat/0312028.

25 M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Struct. Algorithms*, 6(2/3):161–179, 1995.

26 M. E. J. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256, 2003. doi:10.1137/S003614450342480.

27 M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64:026118, Jul 2001. doi:10.1103/PhysRevE.64.026118.

28 R. Pagh. *Basic external memory data structures*, pages 36–61. Springer Berlin Heidelberg, 2003.

29 J. Ray, A. Pinar, and C. Seshadhri. *Are We There Yet? When to Stop a Markov Chain while Generating Random Graphs*, pages 153–164. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-30541-2_12.

30 J. Ray, A. Pinar, and C. Seshadhri. A stopping criterion for markov chains when generating independent random graphs. *J. of Compl. Net.*, 3(2), 2015. doi:10.1093/comnet/cnu041.

31 W. E. Schlauch, E. Á. Horvát, and K. A. Zweig. Different flavors of randomness: comparing random graph models with fixed degree sequences. *Social Network Analysis and Mining*, 5(1):1–14, 2015. doi:10.1007/s13278-015-0267-z.

32 W. E. Schlauch and K. A. Zweig. Influence of the null-model on motif detection. In *ASONAM'15*, NY, USA, 2015. ACM. doi:10.1145/2808797.2809400.

33 C. L. Staudt, A. Sazonovs, and H. Meyerhenke. NetworKit: A tool suite for large-scale complex network analysis. *Network Science*, 4(04), 2016. doi:10.1017/nws.2016.20.

34 S. H. Strogatz. Exploring complex networks. *Nature*, 410(6825):268, 2001.

35 G. Strona, D. Nappo, F. Boccacci, S. Fattorini, and J. San-Miguel-Ayanz. A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. *Nature Communications*, 5:4114–, 2014. doi:10.1038/ncomms5114.

36 N. D. Verhelst. An efficient MCMC algorithm to sample binary matrices with fixed marginals. *Psychometrika*, 73(4):705–728, 2008.

37 F. Viger and M. Latapy. Fast generation of random connected graphs with prescribed degrees. *CoRR*, feb 2005. Source code available at https://www-complexnetworks.lip6.fr/~latapy/FV/generation.html. arXiv:cs/0502085.

Revision Notice

This is a revised version of the eponymous paper appeared in the proceedings of ESA 2018 (LIPIcs, volume 112, `http://www.dagstuhl.de/dagpub/978-3-95977-081-1` published in August, 2018), in which the concept of global trades is now correctly attributed to Carstens, Berger, Strona. Curveball: a new generation of sampling algorithms for graphs with fixed degree sequence. arXiv:1609.05137.

Dagstuhl Publishing – August 27, 2018.