HomeRun: Performing Curveball Trades quasi in Streaming for Fast Null Modeling of Graphs, Hypergraphs, and Binary Matrices

Michelle Contreras-Catalan
Universidad de Chile
Santiago, Chile
michellecontreras@ug.uchile.cl

Matteo Riondato
Amherst College
Amherst, MA, USA
mriondato@amherst.edu

Abstract

State-of-the-art algorithms for randomizing (or "null-modeling") graphs, hypergraphs, binary matrices, and market basket datasets are based on the Curveball trade, a procedure that takes two n-dimensional binary vectors (e.g., 1-hop neighborhood vectors, hyperedges, columns, transactions), and transforms them into two new vectors with the same L^1 norm, bitwise-XOR, and bitwise-AND as the original ones.

We present HomeRun, an algorithm to perform Curveball trades in an almost-streaming fashion. Previous approaches make multiple passes over the input vectors, and perform other operations with $O(n)$ computational cost. HomeRun instead performs a single linear scan over the data, and costs less than two passes. We achieve this speedup by framing a Curveball trade as a sampling procedure, and by using reservoir sampling and strict upper bounds to avoid having to compute the population and sample sizes.

Our experimental evaluation on real and artificial datasets shows that HomeRun achieves a 2x speedup over existing algorithms.

CCS Concepts

• **Theory of computation → Random walks and Markov chains**; **Generating random combinatorial structures**; **Random network models**; • **Mathematics of computing → Markov-chain Monte Carlo methods**; **Graph algorithms**; **Hypothesis testing and confidence interval computation**.

Keywords

Configuration Model, Network Science, Switch Chain

ACM Reference Format:
Michelle Contreras-Catalan and Matteo Riondato. 2026. HomeRun: Performing Curveball Trades quasi in Streaming for Fast Null Modeling of Graphs, Hypergraphs, and Binary Matrices. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3774904.3792157

Resource Availability:
The source code and datasets for this paper have been made publicly available at https://doi.org/10.7910/DVN/OIECJF.

WWW '26, Dubai, United Arab Emirates

1 Introduction

(Multi-, hyper-, bipartite, dyadic) graphs, binary matrices, and market basket transactional datasets are used to represent data in a multitude of scenarios, and the analysis of such data has never-ending applications in different areas, from the analysis of social and trade networks [26], to the study of bipartisan behavior in political institutions [18], to ecology [11], to building recommender systems [16], to understanding protein-protein interactions [35], gene mutations [33, 39], and the spread of contagious diseases [4].

The real goal of data analysis is to obtain new knowledge about the partially unknown, stochastic, Data Generation Process (DGP) that created the observed dataset $\mathring{D}$, not to "squeeze out" such dataset, which is a partial, noisy representation of the DGP [34, 45]. A null model $(\mathcal{D}, \pi)$ is a mathematical formalization of the DGP, encoding all available knowledge or assumptions about the DGP. It is a set $\mathcal{D}$ of all the possible datasets that the DGP may generate (thus including $\mathring{D}$), together with a distribution π over $\mathcal{D}$, representing the probability that a dataset in $\mathcal{D}$ is generated. Assumptions and previous knowledge are encoded as a set of properties that all datasets in $\mathcal{D}$ must preserve. Null models for the aforementioned types of data have been and are continuously proposed [1, 2, 10, 13–15, 28–32, 37, 38, and references therein] where the maintained set of properties is, e.g., the degree sequence (for graphs), the degree sequence and hyperedge dimension sequence (for hypergraphs), the row- and column-sum sequences (for binary matrices), or the items frequencies and transaction lengths (for transactional datasets).

The framework of statistical hypothesis testing [23] can be used to determine whether the results obtained from the analysis of the observed dataset $\mathring{D}$ can be reasonably explained by the existing knowledge or assumptions encoded in the null model, or whether these results carry new information about the DGP. In this framework, results from $\mathring{D}$ are compared to the empirical distribution of the same results obtained by drawing datasets from $\mathcal{D}$ according to π. An empirical p-value is computed, quantifying the probability that the DGP would generate a dataset with results as or more extreme than the ones obtained from $\mathring{D}$. A small p-value gives evidence that the observed dataset contains new information about the DGP. The key algorithmic challenge is to develop fast methods to draw samples from $\mathcal{D}$ according to π.

The set $\mathcal{D}$ is usually very rich and large, and π is an arbitrary distribution, so direct sampling is not an efficient approach [27]. The availability of $\mathring{D}$ makes Markov-Chain-Monte-Carlo (MCMC) methods very appealing, by running, from $\mathring{D}$, a Markov Chain (MC) over $\mathcal{D}$ with stationary distribution π. The modeling question is how to design such an MC, and the algorithmic question is how to make it efficient, i.e., how to ensure it converges to the stationary

distribution as fast as possible, both in the number of steps needed and in the elapsed wall-clock time. For many scenarios, for all the types of data we mentioned, such MCs are available (see Sect. 2 and Sect. 3.1.1), and many of them use the same procedure to select the next state of the MC, namely, the Curveball trade [40, 42], which takes two binary vectors (e.g., two vertex neighborhoods, two hyperedges, two matrix columns or rows, two transactions), and outputs two new vectors with the same number of non-zero entries, the same bitwise-AND, and the same bitwise-XOR, as the original ones (see Sect. 3.1). When the vectors represent vertex neighborhoods, a Curveball trade ensures that the degree sequence does not change. Similarly, for other types of data, the aforementioned sets of properties are preserved. Performing a Curveball trade is the key computational step in MCMC methods for sampling from null models for the aforementioned types of data, so it is imperative that the execution of this operation is fast, so the MC does not take much time to converge to the stationary distribution, and many samples from the null model can be drawn, leading to a precise empirical distribution that allows trustworthy hypothesis testing.

Contributions. We study the problem of performing Curveball trades as efficiently as possible. Our contributions are the following:

- We frame the Curveball trade as a procedure to draw a random sample of an input-dependent size from a specific input-dependent population (Sect. 3.1). This point of view, seemingly not noticed in the past, enables us to leverage on the extended literature on efficient random sampling.
- We present HomeRun, a new algorithm for performing Curveball trades. It performs a single linear scan of the input, followed by additional operations that cost strictly less than a scan in total, thus performing a Curveball trade *quasi* in streaming. Previous algorithms, by contrast, made multiple passes over the input and additional operations with linear cost (Sect. 4.1). At the basis of HomeRun is the use of reservoir sampling [43] (Sect. 3.2), an efficient way of creating a random sample when the size of the population is not known. Curveball trades pose the additional challenge that the sample size is not available a priori, as it is input dependent. We tackle this aspect in HomeRun by using strict upper bounds to the sample size that are refined as more information about the input is available.
- Our experimental evaluation on real and artificial datasets shows that HomeRun performs Curveball trades up to 2x faster than existing algorithms, and an equally faster convergence of MC methods based on Curveball trades.

2 Related work

A well-known approach to transform one binary matrix into another with the same row and column sums is the tetrad swap [36]. In this move, four not-necessarily-adjacent entries of the matrix forming either the submatrix $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$ or the submatrix $\left(\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right)$ are "swapped" to obtain the other submatrix. Ryser [36, Ch. 6] showed that there is a sequence of tetrad swaps transforming any binary matrix into any other binary matrix with the same row and column sums. Since then, this move has been at the basis of MCs for this space [17], but also for spaces defined over bipartite graphs [21, 29] (where the bi-adjacency matrix is considered), transactional datasets for market-basket-analysis [1, 19], hypergraphs [2, 9, 31], and graphs (where it involves switching two neighbors of two vertices) [15, 32]. Theoretical analysis of the mixing time of MCs based on tetrad swap showed that, in some cases, they may mix in polynomial time [12]. In practice though, the wall-clock time to find swappable entries (as not all 4-tuples of entries are swappable) often dominates the time taken by these MCs to converge to the stationary distribution. For this reason, Allendorf et al. [3] suggest a parallelization of the MC, and Wang [44] proposed RectangleLoop, a smarter way to find swappable 4-tuples in binary entries, which is slightly faster than the original tetrad swap, and may mix in fewer steps. Variants of the tetrad swap involving more than four entries have been used for sampling from more complex models [41], e.g., bipartite graphs with a prescribed number of caterpillars [29], and it has been shown that there are models for which it is not sufficient to swap up to a given amount of entries [30].

The introduction of the Curveball trade, originally proposed by Verhelst [42] and so named by Strona et al. [40], led to a significant change of paradigm: the Curveball trade is the first proposed move where more than four entries in a binary matrix may be swapped. The Curveball trade achieves this result by operating on whole columns or rows, i.e., on vectors, performing multiple tetrad swaps between elements in these vectors. Current state-of-the-art algorithms are based on the Curveball trade [6, 7, 10, 40], and, for many settings, from graphs to hypergraphs to transactional datasets, existing algorithms can be easily adapted to use the Curveball trade in place of the tetrad swap. Carstens and Kleer [8] discuss how MCs based on the Curveball trade mix at least as fast as those based on the tetrad swap. The original works introducing the Curveball trade were scant in algorithmic detail [40, 42]. Godard and Neal [20] present FastBall, a possible efficient approach to performing Curveball trades. We discuss it in detail in Sect. 4.1. As we point out there, FastBall makes many passes over the input vectors. Our main contribution in this work is HomeRun, an algorithm to perform Curveball trades with a single pass over the vectors. It can be used in all the aforementioned settings.

3 Preliminaries

We now introduce key concepts and notation.

For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, \dots, n\}$.

Let $\mathbf{a} \doteq \langle a_1, \dots, a_n \rangle \in \{0,1\}^n$ be a n-dimensional binary vector. We denote with $\overline{\mathbf{a}}$ its bitwise negation. Let $\|\mathbf{a}\|_1$ be its L^1-norm, i.e., the sum of the absolute values of the entries in $\mathbf{a}$, which is the same as its L^0-norm, i.e., the number of non-zero entries in $\mathbf{a}$, because $\mathbf{a}$ is binary. Let $\mathrm{nzi}(\mathbf{a}) \subseteq [n]$ be the set of indices s.t. $a_i = 1$, $i \in [n]$.

For two n-dimensional binary vectors $\mathbf{a}$ and $\mathbf{b}$, we denote with $\mathbf{a} \,\&\, \mathbf{b}$ their bitwise-AND, and with $\mathbf{a} \oplus \mathbf{b}$ their bitwise-XOR. We use $\mathbf{a} \setminus \mathbf{b}$ to denote the n-dimensional binary vector whose entries with value 1 are all and only those where the corresponding entry in $\mathbf{a}$ has value 1, and the corresponding entry in $\mathbf{b}$ has value 0, i.e.,

$$\mathbf{a} \setminus \mathbf{b} \doteq \mathbf{a} \,\&\, \overline{\mathbf{b}} \ .$$

Given a population $\mathcal{P}$ and a sample size $0 \le q \le |\mathcal{P}|$, let $\mathcal{Z}(\mathcal{P}, q)$ be the set of all subsets of $\mathcal{P}$ of size q. We say that $\mathcal{S} \in \mathcal{Z}(\mathcal{P}, q)$ is a *uniform sample of $\mathcal{P}$ of size q* when it is chosen uniformly at random from $\mathcal{Z}(\mathcal{P}, q)$. An algorithm creates a uniform sample when all elements of $\mathcal{Z}(\mathcal{P}, q)$ have the same probability to be output.

3.1 The Curveball trade

The Curveball trade (so named by Strona et al. [40] but originally introduced by Verhelst [42]) is an operation that, in its most general form, takes two n-dimensional vectors $\mathbf{a}$ and $\mathbf{b}$ and returns two n-dimensional vectors $\mathbf{c}$ and $\mathbf{d}$ such that[1]

(1) $\|\mathbf{a}\|_1 = \|\mathbf{c}\|_1$ and $\|\mathbf{b}\|_1 = \|\mathbf{d}\|_1$, i.e., the new vectors have the same number of non-zero entries as the original ones; and
(2) $\mathbf{a} \oplus \mathbf{b} = \mathbf{c} \oplus \mathbf{d}$, i.e., for every $i \in [n]$ s.t. $a_i \neq b_i$, it holds $c_i \neq d_i$; and
(3) $\mathbf{a} \,\&\, \mathbf{b} = \mathbf{c} \,\&\, \mathbf{d}$, i.e., for every $i \in [n]$ s.t. $a_i = b_i$, it holds $c_i = d_i = a_i = b_i$;

We frame a Curveball trade as a procedure based on uniform sampling from a specific population. This point of view is extremely convenient, although very different from the original presentation of a Curveball trade, as it opens up the connection, which we exploit to develop HomeRun, with efficient methods to build a sample. Under this lens, a Curveball trade proceeds as follows:

(1) Let $\mathcal{P}_{\mathbf{a},\mathbf{b}} \doteq \mathrm{nzi}(\mathbf{a} \oplus \mathbf{b}) \subseteq [n]$ be our population. In other words, $\mathcal{P}_{\mathbf{a},\mathbf{b}}$ is the set of indices $i \in [n]$ such that $a_i \neq b_i$. Let $q_{\mathbf{a},\mathbf{b}} \doteq \|\mathbf{a} \setminus \mathbf{b}\|_1$ be the sample size, that is, the number of indices $i \in [n]$ where $a_i = 1$ and $b_i = 0$. Finally, let $\mathcal{S}_{\mathbf{a},\mathbf{b}}$ be a uniform sample of $\mathcal{P}_{\mathbf{a},\mathbf{b}}$ of size $q_{\mathbf{a},\mathbf{b}}$;
(2) Set $c_i = 1$ and $d_i = 0$ for every $i \in \mathcal{S}_{\mathbf{a},\mathbf{b}}$;
(3) Set $d_i = 1$ and $c_i = 0$ for every $i \in \mathcal{P}_{\mathbf{a},\mathbf{b}} \setminus \mathcal{S}_{\mathbf{a},\mathbf{b}}$;
(4) For every $i \in [n] \setminus \mathcal{P}_{\mathbf{a},\mathbf{b}}$, set $c_i = d_i$ to the value of a_i (which is the same as the value of b_i). Bitwise, this step corresponds to setting $c_i = d_i = 1$ for every $i \in \mathrm{nzi}(\mathbf{a} \,\&\, \mathbf{b})$, and $c_i = d_i = 0$ for every $i \in \mathrm{nzi}(\overline{\mathbf{a}} \,\&\, \overline{\mathbf{b}})$.

This description leaves open several algorithmic questions about how to actually perform a Curveball trade. For example, how to compute the population, the sample size, the sample, and how to perform all the operations to set the values in the new vectors. In this work we answer these questions with HomeRun, an efficient algorithm to perform Curveball trades.

In the rest of this work, to simplify the presentation of algorithms and proofs, we make the assumption, w.l.o.g., that $\|\mathbf{a}\|_1 \leq \|\mathbf{b}\|_1$, as we can just switch the two vectors, and then switch the output vectors $\mathbf{c}$ and $\mathbf{d}$ just before returning them.

3.1.1 Using the Curveball trade. As we discussed in Sect. 2, Curveball trades are used in algorithms to sample from the space of binary matrices with prescribed row and column sums [5, 40], from the space of bipartite graphs with prescribed degree sequences [21, 29], graphs with prescribed degree sequence [6, 15], transactional datasets with fixed transaction lengths and item frequencies [1, 19], and, with a minor variation in the last step, hypergraphs with prescribed degree sequence and edge dimension [10]. We discuss here in more detail the case of binary matrices and of graphs, but all we say, including the algorithms we present in Sect. 4, can be adapted to all above scenarios and possibly more.

Let $M \in \{0,1\}^{n \times m}$ be a $n \times m$ binary matrix, where $\mathsf{r}_i(M)$ denotes its i^{th} row, $1 \leq i \leq n$, and $\mathsf{c}_j(M)$ its j^{th} column, $1 \leq j \leq m$. Let

$$\mathsf{R}(M) \doteq \langle \|\mathsf{r}_1(M)\|_1, \ldots, \|\mathsf{r}_n(M)\|_1 \rangle$$

be the vector of the row sums of M, and similarly

$$\mathsf{C}(M) \doteq \langle \|\mathsf{c}_1(M)\|_1, \ldots, \|\mathsf{c}_m(M)\|_1 \rangle$$

be the vector of the column sums of M.

For any $\mathbf{r} \in \mathbb{N}^n$, $\mathbf{c} \in \mathbb{N}^m$, define

$$\mathcal{M}(\mathbf{r}, \mathbf{c}) \doteq \left\{ M \in \{0,1\}^{n \times m} : \mathsf{R}(M) = \mathbf{r} \wedge \mathsf{C}(M) = \mathbf{c} \right\}$$

as the set of all $n \times m$ binary matrices with row-sums vector $\mathbf{r}$ and column-sums vector $\mathbf{c}$.

The set $\mathcal{M}(\mathsf{R}(\mathring{M}), \mathsf{C}(\mathring{M}))$, together with the uniform distribution over it, form a widely-used null model with applications from many fields (see Sect. 1). It encodes the available knowledge, or assumption, that the Data Generation Process only produces matrices with the same row- and column-sums vectors as the observed matrix $\mathring{M}$.

Given an observed $n \times m$ binary matrix $\mathring{M}$, a Markov-Chain-Monte-Carlo (MCMC) method to sample uniformly from $\mathcal{M}(\mathsf{R}(\mathring{M}), \mathsf{C}(\mathring{M}))$ involves running a Markov Chain (MC) from $S_0 = \mathring{M}$, as follows [40]. At each step $t \geq 0$, with the current state being the matrix S_t, one draws an unordered pair of distinct indices $(a, b) \in [m] \times [m]$ uniformly from the set of such pairs. One then performs a Curveball trade between the columns $\mathsf{c}_{S_t}(a)$ and $\mathsf{c}_{S_t}(b)$ of the current state S_t to obtain the columns $\mathbf{c}$ and $\mathbf{d}$ which replace respectively $\mathsf{c}_{S_t}(a)$ and $\mathsf{c}_{S_t}(b)$ in S_t to obtain the next state S_{t+1}. The fact that S_{t+1} belongs to $\mathcal{M}(\mathsf{R}(\mathring{M}), \mathsf{C}(\mathring{M}))$ follows from the properties of the Curveball trade: the first property guarantees that the column-sums vector is unchanged, and the second and third properties guarantee that the row-sums are unchanged. After a sufficient number of steps, the state of the MC is distributed uniformly over $\mathcal{M}(\mathsf{R}(\mathring{M}), \mathsf{C}(\mathring{M}))$ [5].

A similar null model can be defined on graphs with the same degree sequence as an observed one $\mathring{G}$ (see Sect. 2), which is known as the micro-canonical configuration model. MCMC algorithms to sample from this space proceed as follows [7]. Assume for now that the graphs are directed. They can be represented through their adjacency lists, i.e., for each vertex w, which, w.l.o.g., can be assumed to be in $[n]$, one keeps a n-dimensional binary vector $\mathbf{v}_w$ where the entries set to 1 are the (out-)neighbors of w. At each step, two distinct vertices a and b are sampled (from the same side of the graph, if the graph is bipartite), and a Curveball trade is performed between their neighborhood vectors $\mathbf{v}_a$ and $\mathbf{v}_b$. In this settings, the L^1 norm of a vector is the degree of its corresponding vertex, so the properties of the Curveball trade guarantee that the degree sequence is unchanged. The only minor complication occurs when the space of graphs from which to sample should only contains graphs without self-loops. In this case, the population for the Curveball trade is defined as $\mathcal{P}_{\mathbf{v}_a,\mathbf{v}_b} \doteq \mathrm{nzi}(\mathbf{v}_a \oplus \mathbf{v}_b) \setminus \{a, b\}$. This modified definition requires minimal changes to the algorithms we present in Sect. 4, so we do not discuss it further. When the (non-bipartite) graphs are undirected, an additional final step is required to propagate the changes in the graph structure resulting from the trade to the adjacency lists for the (former) neighbors of a and b involved in the trade [6]. This step is straightforward, and computationally cheap, so we do not consider it further in our presentation.

For bipartite graphs, a Curveball trade proceeds as in the case for directed graphs, except that we only consider the m vertices on one side of the graph, while there are n on the other side. An equivalent

[1] For ease of presentation, we discuss Curveball trades, and algorithms performing them, as returning two new vectors $\mathbf{c}$ and $\mathbf{d}$. The discussion can be easily adapted to transform the input vectors $\mathbf{a}$ and $\mathbf{d}$ in place.

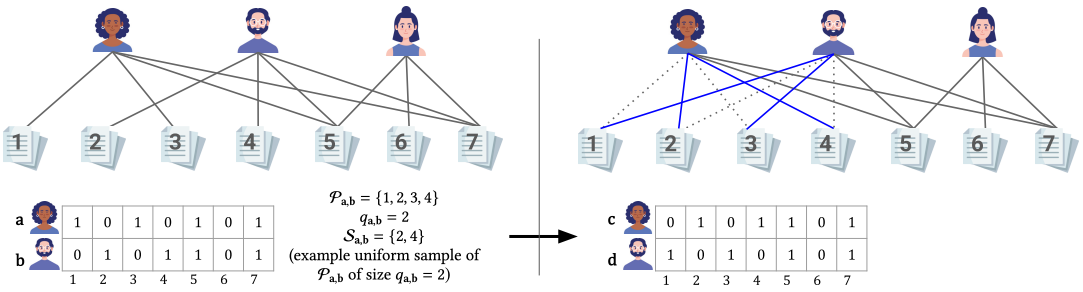


Figure 1: An example Curveball trade on a bipartite graph of scientists and the articles they authored. On the right graph, the new edges resulting from the trade are in blue, and the removed edges are dashed (and not part of the graph).

point of view is thinking that the Curveball trade is acting on the biadjacency matrix of the bipartite graph. An example of a Curveball trade on a bipartite graph (for $m = 3$ and $n = 7$), is shown in Fig. 1.

The Curveball trade variant between vectors of edge stubs defined by Chodrow [10] for hypergraphs (and called "pairwise shuffle") can also be easily handled by the algorithms we discuss.

As it should be clear from the above description, the key algorithmic step in sampling from the above null model of matrices or graphs is the Curveball trade. In this work we introduce an algorithm to perform such operation as fast as possible. We remark that our goal is not to modify Curveball trades so that the MC reaches the stationary distribution in fewer steps, rather to speed up the execution of each trade so that the MC reaches the stationary distribution in less wall-clock time.

Because Curveball trades are always performed in a setting like the above, where $\mathring{M}$ or $\mathring{G}$ is available, and the row- and column-sum vectors or the degrees do not change, i.e., none of the L^1-norms of the vectors change, we make the very mild assumption that the L^1-norms of the vectors $\mathbf{a}$, $\mathbf{b}$ involved in a Curveball trade are available in time $O(1)$. Indeed they could all be computed once from $\mathring{M}$ or $\mathring{G}$ and never again, as they do not change.

3.2 Reservoir sampling

Reservoir sampling [43] is a popular approach to create a uniform sample $\mathcal{S}$ of size q from a population $\mathcal{P}$ when the population size is not known a priori [22, Sect. 3.4.2]. Our algorithm HomeRun uses reservoir sampling with some tweaks (see Sect. 4.2). We describe here the most common algorithm for reservoir sampling [43, Algorithm R]. An interesting direction for future work is adapting HomeRun to use more refined methods for reservoir sampling (e.g., [43, Algorithm Z], or the algorithms by Li [25]).

Algorithm R scans the elements of $\mathcal{P}$ one by one, in an arbitrary order $\langle p_1, \ldots, p_{|P|} \rangle$, keeping track of the number $t \geq 1$ of elements seen so far (including the one currently being scanned). It also keeps an array r (the reservoir) of size q, initially empty (we consider array cells to be 1-indexed, for ease of presentation). For any time $t \leq q$, p_t is inserted into the t^{th} cell of r. For $t > q$, the algorithm flips a coin with heads bias $b_t = q/t$. If the outcome is heads, p_t is inserted in a cell of r chosen uniformly at random, replacing the element currently there. If the outcome is tails, the reservoir is not touched, and the algorithm moves to scan the next element.

HomeRun relies on the following property of Algorithm R.

Lemma 3.1 (43, Sect. 2). *After p_t has been handled, the elements in the reservoir r form a uniform random sample of $\langle p_1, \ldots, p_t \rangle$ of size $\min\{t, q\}$.*

We also make use of the following well-known fact, which can be summarized as "a uniform sample of a uniform sample is a uniform sample".

Fact 3.2. *Let $\mathcal{P}$ be a population, and $\mathcal{S}$ be a uniform sample of $\mathcal{P}$ of size q. Let $q' < q$, and let $\mathcal{S}'$ be a uniform sample of $\mathcal{S}$ of size q'. Then $\mathcal{S}'$ is a uniform sample of $\mathcal{P}$ of size q'.*

4 Algorithms for Curveball trades

We now describe two algorithms for performing Curveball trades, starting from FastBall [20], and then present our new algorithm HomeRun. As mentioned in Sect. 3.1.1, we discuss them in the most generic form, but they directly apply or can be straightforwardly applied to the cases where the vectors represent neighborhood of vertices, hyperedges of vertex stubs, market basket transactions, and more.

4.1 FastBall

FastBall [20] is a possible way to perform a Curveball trade. We describe it here to point out the inefficiencies that our algorithm HomeRun addresses (Sect. 4.2), and because FastBall serves as a baseline for our experimental evaluation of HomeRun (Sect. 5).

Algorithm 1 gives the pseudocode for FastBall. The algorithm takes in input two n-dimensional binary vectors $\mathbf{a}$ and $\mathbf{b}$, and performs a Curveball trade to return vectors $\mathbf{c}$ and $\mathbf{d}$ as follows. It first computes (lines 1–3), the number z of entries that are 1 in both $\mathbf{a}$ and $\mathbf{b}$ (i.e., $z = \|\mathbf{a} \,\&\, \mathbf{b}\|_1$). Then it creates a *victory vector* $\mathbf{v}$ that has $\|\mathbf{a}\|_1 - z$ entries set to 1 and $\|\mathbf{b}\|_1 - z$ entries set to 0 (line 4). We recall the assumption that the L^1-norm of input vectors can be obtained in time $O(1)$ (see Sect. 3.1.1). The dimensionality of $\mathbf{v}$ is the size $|\mathcal{P}_{a,b}|$ of the population, and the quantity $\|\mathbf{a}\|_1 - z$ is the sample size $q_{a,b}$. FastBall then shuffles $\mathbf{v}$ randomly, e.g., using the Fisher-Yates approach [22, Sect. 3.4.2] (line 5). The uniform sample $\mathcal{S}_{a,b}$ used by Curveball is implicitly defined by the victory vector $\mathbf{v}$ as follows. The algorithm iterates through the indices in $[n]$ (lines 8–13), keeping track, in the counter t, of the number of elements of the population $\mathcal{P}_{a,b}$ it has seen so far (line 10). When the current index i is an element of $\mathcal{P}_{a,b}$ (line 9), then i belongs to $\mathcal{S}_{a,b}$ iff v_t has value 1. If that is the case, then c_i is set to 1 and d_i to 0

(line 11), otherwise, since it must be $i \in \mathcal{P} \setminus \mathcal{S}$, d_i is set to 1 and c_i to 0 (line 12). If the current index i is not an element of the population (line 13), then c_i and d_i are set to the value of a_i (which is the same as the value of b_i), as required by the last step of Curveball.

Algorithm 1: FastBall

Input : Two n-dimensional binary vectors $\mathbf{a}$ and $\mathbf{b}$
Output: Two n-dimensional binary vectors resulting from a Curveball trade between $\mathbf{a}$ and $\mathbf{b}$

1 $z \leftarrow 0$;
2 **for** $i \leftarrow 1$ **to** n **do**
3 **if** $a_i = 1$ **and** $b_i = 1$ **then** $z \leftarrow z + 1$;
4 $\mathbf{v} \leftarrow \langle \underbrace{1, \ldots, 1}_{\|\mathbf{a}\|_1 - z}, \underbrace{0, \ldots, 0}_{\|\mathbf{b}\|_1 - z} \rangle$;
5 Randomly shuffle $\mathbf{v}$;
6 $\mathbf{c}, \mathbf{d} \leftarrow$ empty n-dimensional vectors;
7 $t \leftarrow 0$;
8 **for** $i \leftarrow 1$ **to** n **do**
9 **if** $a_i \neq b_i$ **then**
10 $t \leftarrow t + 1$;
11 **if** $v_t = 1$ **then** $c_i \leftarrow 1$, $d_i \leftarrow 0$;
12 **else** $d_i \leftarrow 1$, $c_i \leftarrow 0$;
13 **else** $c_i \leftarrow a_i$, $d_i \leftarrow a_i$;
14 **return** $\mathbf{c}$, $\mathbf{d}$

Let us now discuss the computational complexity of FastBall. There are multiple steps each taking time $O(n)$:

(1) the computation of z (lines 1–3);
(2) the creation of $\mathbf{v}$ (line 4);
(3) the shuffling of $\mathbf{v}$ (line 5); and
(4) the loop to set the entries of $\mathbf{c}$ and $\mathbf{d}$ (lines 8–13).

Thus, FastBall costs as much as four linear passes over the input data, i.e., slightly abusing notation, $O(4n)$. Our main goal for HomeRun is to decrease this cost to essentially a single pass.

4.2 HomeRun

The key idea of HomeRun is to use reservoir sampling (Sect. 3.2). Indeed, creating the sample $\mathcal{S}_{\mathbf{a},\mathbf{b}}$ is the crucial operation in a Curveball trade, but neither $\mathcal{P}_{\mathbf{a},\mathbf{b}}$ nor its size are available at the start, and reservoir sampling is developed exactly for such a situation.

Curveball trades also pose an additional challenge: the sample size $q_{\mathbf{a},\mathbf{b}}$ is also unknown a priori, as it depends, like $\mathcal{P}_{\mathbf{a},\mathbf{b}}$, on properties of the input vectors (Sect. 3.1). HomeRun tackles this second challenge by first computing an upper bound $\hat{q} \geq q_{\mathbf{a},\mathbf{b}}$, and creating a reservoir r with initial size $\hat{q}$. As HomeRun goes through the elements of the input vectors, it refines (i.e., decreases) $\hat{q}$, and shrinks r so it has size $\hat{q}$, while guaranteeing that its contents are still a uniform sample of the population elements seen so far. Once all the elements of the input vectors have been handled, it is guaranteed that $\hat{q} = q_{\mathbf{a},\mathbf{b}}$. We defer the theoretical analysis to after the description the algorithm.

HomeRun works as follows (pseudocode in Alg. 2). Let $\mathbf{a}$ and $\mathbf{b}$ be the input vectors. The upper bounds $\hat{q}'$ and $\hat{q}''$ are initialized to $\|\mathbf{a}\|_1$ and $n - \|\mathbf{b}\|_1$ respectively (lines 1 and 2), and the reservoir r is created as an empty vector with size the minimum of the two upper bounds (line 3). HomeRun uses t to keep track of the number of elements of the population seen so far (line 8).

Algorithm 2: HomeRun

Input : Two n-dimensional binary vectors $\mathbf{a}$ and $\mathbf{b}$
Output: Two n-dimensional binary vectors resulting from a Curveball trade between $\mathbf{a}$ and $\mathbf{b}$

1 $\hat{q}' \leftarrow \|\mathbf{a}\|_1$;
2 $\hat{q}'' \leftarrow n - \|\mathbf{b}\|_1$;
3 $r \leftarrow$ empty array of size $\min\{\hat{q}', \hat{q}''\}$;
4 $t \leftarrow 0$;
5 $\mathbf{c}, \mathbf{d} \leftarrow$ empty n-dimensional vectors;
6 **for** $i \leftarrow 1$ **to** n **do**
7 **if** $a_i \neq b_i$ **then**
8 $t \leftarrow t + 1$;
9 **if** $t \leq r.\texttt{size}()$ **then** $r[t] \leftarrow i$;
10 **else**
11 $f \leftarrow$ flip of coin with head bias $r.\texttt{size}()/t$;
12 **if** f *is heads* **then**
13 $h \leftarrow$ index chosen uniformly at random from $[r.\texttt{size}()]$;
14 $d_{r[h]} \leftarrow 1$, $c_{r[h]} \leftarrow 0$;
15 $r[h] \leftarrow i$;
16 **else** $d_i \leftarrow 1$, $c_i \leftarrow 0$;
17 **else**
18 $c_i \leftarrow a_i$, $d_i \leftarrow a_i$;
19 **if** $a_i = 1$ **then** $\hat{q}' \leftarrow \hat{q}' - 1$;
20 **else** $\hat{q}'' \leftarrow \hat{q}'' - 1$;
21 **if** $\min\{\hat{q}', \hat{q}''\} < r.\texttt{size}()$ **then**
22 **if** $t \geq r.\texttt{size}()$ **then**
23 $j \leftarrow$ index chosen uniformly at random from $[r.\texttt{size}()]$;
24 $d_{r[j]} \leftarrow 1$, $c_{r[j]} \leftarrow 0$;
25 $r[j] \leftarrow r[r.\texttt{size}()]$;
26 Shrink r by removing its last cell;
27 **for** $i \leftarrow 1$ **to** $r.\texttt{size}()$ **do** $c_{r[i]} \leftarrow 1$, $d_{r[i]} \leftarrow 0$;
28 **return** $\mathbf{c}$, $\mathbf{d}$

The algorithm then goes through $\mathbf{a}$ and $\mathbf{b}$ (loop on lines 6–26). If the current index i is an element of the population (line 7), then the reservoir sampling approach is used to possibly update r (lines 9–16). In particular, when an index is evicted from the reservoir, then it will definitively not belong to $\mathcal{S}_{\mathbf{a},\mathbf{b}}$, so the entries of $\mathbf{d}$ and $\mathbf{c}$ at that index can be set to 1 and to 0 respectively (line 14). In the case that the outcome of the biased coin flip is tails, the current index i will not belong to $\mathcal{S}_{\mathbf{a},\mathbf{b}}$, allowing to set d_i to 1 and c_i to 0 (line 16).

If the current index is not an element of $\mathcal{P}_{\mathbf{a},\mathbf{b}}$ (lines 17–26), then HomeRun first sets c_i and d_i to the value of a_i (line 18), as required by the last step of a Curveball trade, and then updates the upper bounds as follows: if $a_i = 1$, then $\hat{q}'$ is decreased by one (line 19), otherwise $\hat{q}''$ is (line 20). If such a decrease leads to the minimum of these two quantities to also decrease (line 21), then the reservoir r is shrunk by one cell (line 26), ensuring, if r is currently full (line 22),

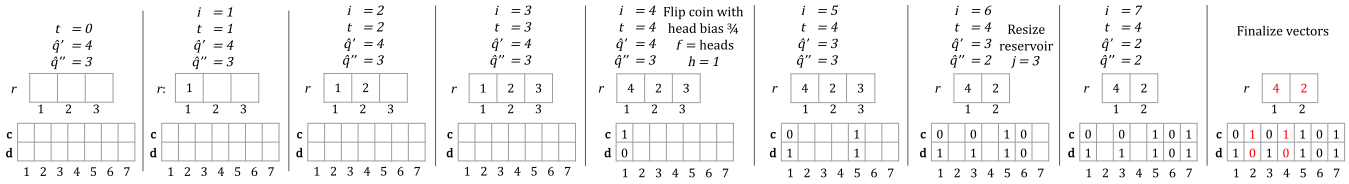


Figure 2: The evolving state of HomeRun's variables during an example run with inputs the vectors a and b from Fig. 1. The leftmost panel is before the loop on lines 6–26 starts, and the rightmost just before the return statement.

that the removed element is chosen uniformly at random from those in r (lines 23–25). As we prove in the analysis later, the final size of the reservoir is exactly $q_{\mathbf{a},\mathbf{b}}$.

Once HomeRun has gone through all the indices, the indices in r form $\mathcal{S}_{\mathbf{a},\mathbf{b}}$, so, for each j of them, c_j is set to 1 and d_j to 0 (line 27), after which c and d are returned. Figure 2 shows an example execution of HomeRun.

Clearly, HomeRun does a pass over the input vectors in the main loop, and each iteration has a constant cost. The final loop (line 27) performs $q_{\mathbf{a},\mathbf{b}}$ iterations of constant cost. Thus the total cost is $O(n + q_{\mathbf{a},\mathbf{b}})$. While $q_{\mathbf{a},\mathbf{b}}$ is at most $\|\mathbf{a}\|_1$, as we assumed $\|\mathbf{a}\|_1 \leq \|\mathbf{b}\|_1$, thus at most n, in practice it is usually much smaller: vectors from real datasets are very sparse, e.g., have L^1-norm $O(\log n)$, as it often is the case when a vector represents the neighborhood of a vertex in a social network. Thus, while not costing exactly like one pass over the data, HomeRun comes very close to that, and we can argue that it performs a Curveball trade *quasi* in a streaming fashion.

Theorem 4.1. *HomeRun performs a Curveball trade involving the input vectors* **a** *and* **b**.

This result relies on the following technical lemma, whose proof is in App. A.1.

Lemma 4.2. *After HomeRun has exited the loop on lines 6–26, the reservoir r is a uniform sample of $\mathcal{P}_{\mathbf{a},\mathbf{b}}$ of size $q_{\mathbf{a},\mathbf{b}}$, i.e., can act as $\mathcal{S}_{\mathbf{a},\mathbf{b}}$ for a Curveball trade between* **a** *and* **b**.

Proof of Thm. 4.1. Consider first the indices $i \in \mathcal{P}_{\mathbf{a},\mathbf{b}}$. Only such indices can be in r, as no element is added to r when $a_i \neq b_i$ (lines 17–26).

From Lemma 4.2 and line 27, it follows that $c_i = 1$ and $d_i = 0$ for all $i \in \mathcal{S}_{\mathbf{a},\mathbf{b}}$, as required by the second step of a Curveball trade.

Consider now the indices $i \in \mathcal{P}_{\mathbf{a},\mathbf{b}}$ that are not in r after the loop on lines 6–26 has completed. From Lemma 4.2, these are the indices in $\mathcal{P}_{\mathbf{a},\mathbf{b}} \setminus \mathcal{S}_{\mathbf{a},\mathbf{b}}$. For any i of them, either i was never in r, or it was in r at some point and then was evicted. In the former case, HomeRun must have performed the setting operations on line 16, so $d_i = 1$ and $c_i = 0$. In the latter case, either i was evicted from r as part of the reservoir sampling process, so on line 14 d_i gets value 1 and c_i gets value 0, or i was evicted from r when r was shrunk due to the decrease in the upper bounds to the sample size, in which case line 24 ensures that $d_i = 1$ and $c_i = 0$. Therefore, it holds $d_i = 1$ and $c_i = 0$ for any $i \in \mathcal{P}_{\mathbf{a},\mathbf{b}} \setminus \mathcal{S}_{\mathbf{a},\mathbf{b}}$, as required by the third step of a Curveball trade.

The vectors **c** and **d** have $c_i = d_i = a_i$ in all indices $i \in [n]$ where $a_i = b_i$, thanks to the setting operation on line 18. Thus, HomeRun performs the last step of a Curveball trade correctly, so it performs all of them correctly. □

The upper bounds $\hat{q}'$ and $\hat{q}''$ to the sample size $q_{\mathbf{a},\mathbf{b}}$ are strict, in the following sense: for any $i \in \{0, \ldots, n\}$, there exist vectors $\mathbf{a}'$ and $\mathbf{b}'$ whose entries up to i included are the same as **a** and **b**, and for which $q_{\mathbf{a}',\mathbf{b}'}$ is exactly the value of $\min\{\hat{q}'_i, \hat{q}''_i\}$. That is, better upper bounds cannot be obtained with only the information available at every step of the algorithm.

Table 1: Graph characteristics

Name	Vertices (n)	Edges	Avg. Degree
wiki-Vote	7,115	103,689	29.146
sx-mathoverflow	24,818	239,978	19.339
cit-HepPh	34,546	421,578	24.406
p2p-Gnutella31	62,586	147,892	4.726
soc-Epinions1	75,879	508,837	15.447
ego-twitter	81,306	1,768,149	43.493
sx-superuser-c2q	94,548	291,030	6.156
sx-superuser-c2a	101,052	289,487	5.729
sx-superuser	194,085	924,886	9.530

5 Experimental evaluation

We report here the results of our experimental evaluation of HomeRun on real and artificial graphs. The goal is to assess its behavior in terms of the time taken to perform a Curveball trade, and how that impacts the time for the Markov chain on the space of matrices with fixed row- and column-sum vectors to convergence to the stationary distribution. The behavior of HomeRun is compared to that of FastBall, as a baseline.

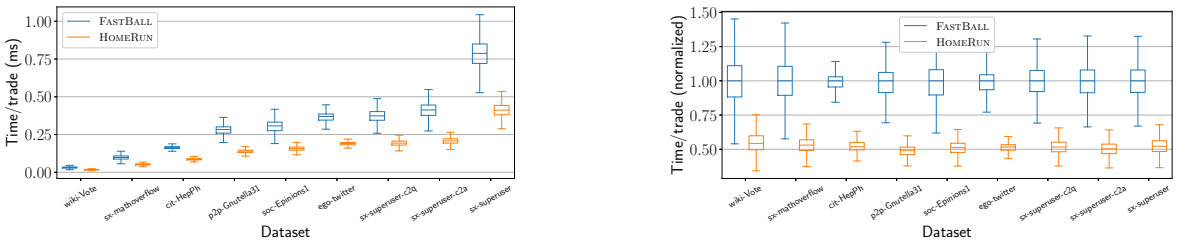
Implementation and environment. We implemented FastBall and HomeRun in Java 21, aiming to optimize both implementations as much as possible.[2] The experiments were run on a machine with two Intel Xeon Silver 4210R CPUs at 2.40GHz and 384GB of RAM, running FreeBSD 15.

5.1 Curveball trade time

The purpose of this experiment is to evaluate the distribution of the time taken by HomeRun to perform a Curveball trade.

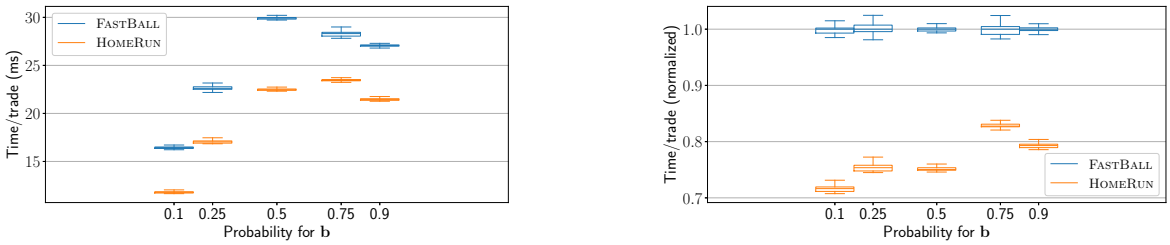
We use the Java Microbenchmark Harness (JMH), which isolates results from the possible skewness due to stochasticities in the Java Virtual Machine behavior, to continuously perform Curveball trades between the randomly chosen pairs of vectors in a dataset, for five JHM iterations of 10 seconds each, with two JMH warm-up iterations of five seconds each. Each such experiment was repeated three times (i.e., three JMH forks). The time taken for each Curveball trade is a data point, and we study the distribution of these values.

[2] Code available from https://doi.org/10.7910/DVN/OIECJF.



(a) Raw time **(b) Normalized time**

Figure 3: Times to perform a Curveball trade.



(a) Raw time **(b) Normalized time**

Figure 4: Times to perform a Curveball trade vs. probability $p_{\mathbf{b}}$ for $\mathbf{b}$ — $p_{\mathbf{a}} = 0.1$, $n = 1,000,000$.

Datasets. For real datasets, we used directed graphs from SNAP[24]. We report their salient characteristics in Table 1. As explained in Sect. 3.1.1, Curveball trades on directed graphs are executed between vectors representing the neighbors of vertices. We restrict to directed graphs because they do not need any additional postprocessing after the Curveball trade is performed, versus what is done for undirected graphs (see Sect. 3.1.1). We created artificial datasets each with exactly two vectors $(\mathbf{a}, \mathbf{b})$ with $n = 1$ million, setting each of the vector entries to 1 independently at random with probabilities $p_{\mathbf{a}}$ and $p_{\mathbf{b}}$, for $(p_{\mathbf{a}}, p_{\mathbf{b}}) \in \{(0.1, 0.1), (0.1, 0.25), (0.1, 0.5), (0.1, 0.75), (0.1, 0.9), (0.25, 0.5), (0.25, 0.75), (0.25, 0.9), (0.5, 0.5)\}$.

Results. Figure 3 shows the distribution of the "raw" time (in ms) taken by HomeRun and FastBall to perform a Curveball trade on the real graphs. The graphs are ordered on the x-axis by increasing number of vertices, but the axis is not to scale. Figure 3a shows the raw times, and Fig. 3b shows the times normalized by the median raw time for FastBall, for each dataset. For both algorithms, the time to perform a trade increases with the number of nodes (i.e., with n), with the absolute time for HomeRun seemingly having a smaller growth factor than the one for FastBall, suggesting that it scales better with n. This impression is indeed confirmed by the normalized times. The median speedup is around 2x, i.e., the normalized time for HomeRun is around 50% of that of FastBall, especially on larger graphs. The distributions of the times are disjoint on even moderately large graphs, indicating that HomeRun is always faster than FastBall. From the distribution of the normalized times we can see that the variance of the move times for HomeRun is also much smaller than the one for FastBall.

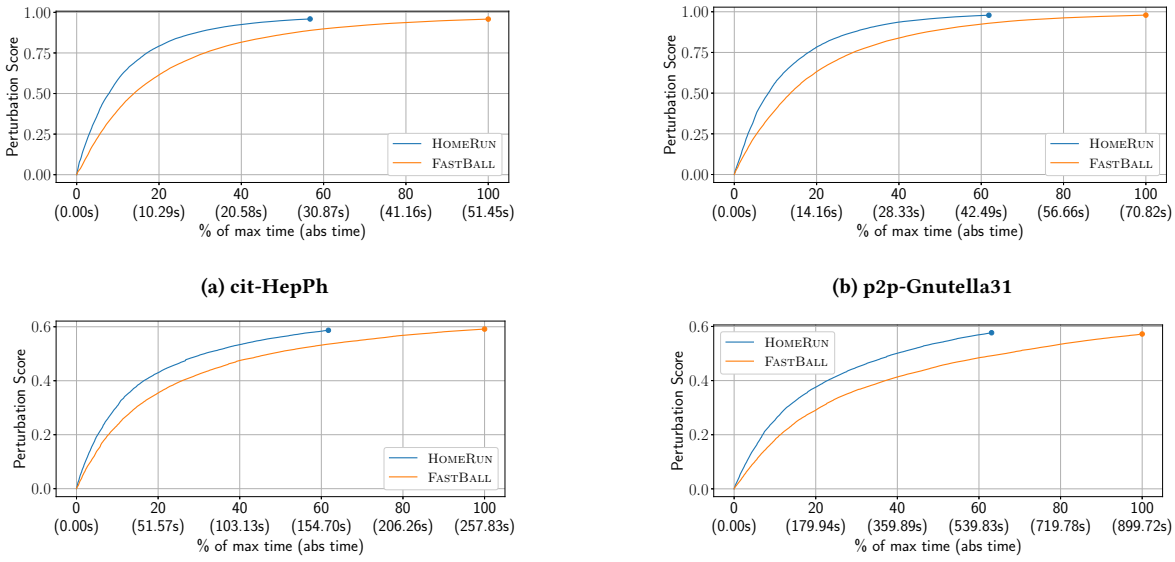
In Fig. 4 we show the behavior of the running times (raw in Fig. 4a, normalized in Fig. 4b) on artificial datasets, as the probability $p_{\mathbf{a}}$ for entries of $\mathbf{a}$ is fixed to 0.1, and the one for $\mathbf{b}$ changes in $\{0.1, 0.25, 0.5, 0.75, 0.9\}$, for $n = 1000000$ (results for other values of $(p_{\mathbf{a}}, p_{\mathbf{b}})$ are in App. A.2, and are qualitatively similar). The x-axes are to scale. HomeRun remains quicker than FastBall as $p_{\mathbf{b}}$ grows. The raw times decrease as $\mathbf{b}$'s density $p_{\mathbf{b}}$ becomes higher than 0.5 because the sample size $q_{\mathbf{a},\mathbf{b}}$ decreases as a consequence, as for many indices $i \in [n]$ it will hold $a_i = b_i = 1$. The normalized times stay relatively constant as the probability for $\mathbf{b}$ change, suggesting that the speedup of HomeRun does not depend on the vector densities.

5.2 Markov chain convergence time

We now study the cumulative effect of using HomeRun to perform Curveball trades when running a Markov Chain (MC) starting from an observed graph $\mathring{G}$ until its state (approximately) reaches the uniform stationary distribution (see Sect. 3.1.1 for details on the MC). We do not compare with RectangleLoop [44] because it takes much longer to converge than Curveball-based algorithms.

Detecting convergence. To empirically detect when the MC has reached convergence, we first measure, every w steps, for some user-specified w, the perturbation score [40] $\mathrm{p}(\mathring{M}, M_t)$ between the adjacency matrix $\mathring{M}$ of the starting graph $\mathring{G}$ and the adjacency matrix M_t of the current state G_t, defined as

$$\mathrm{p}(\mathring{M}, M_t) \doteq \frac{1}{\sum_{i \in m} \left\| \mathrm{c}_{\mathring{M}}(i) \right\|_1} \sum_{(i,j) \in [n] \times [m]} \max\left\{ \mathring{M}(i,j) - M_t(i,j), 0 \right\},$$



(a) cit-HepPh **(b) p2p-Gnutella31**

(c) sx-superuser-c2q **(d) sx-superuser**

Figure 5: Perturbation score vs. cumulative wall-clock time before convergence.

i.e., as the fraction of entries with value 1 in $\mathring{M}$ that have value 0 in M_t. The convergence of the perturbation score as t grows is a widely-used proxy for the convergence of MC over spaces of binary matrices [1, 2, 40, 44]. We then consider the slope c of the least-squares-fit line of the k most recent measurements, for a user specified k. We declare convergence to be reached when $w \cdot k \cdot c \leq (z/100)\mathrm{p}(\mathring{M}, M_t)$, i.e., when the interpolated change in perturbation score over the considered window of $w \cdot k$ steps is less than z% of the most recent measurement of the perturbation score $\mathrm{p}(\mathring{M}, M_t)$, where z is a parameter fixed by the user. We then obtain the elapsed wall-clock time when convergence is reached. For our experiments, we set $w = 1000$, $k = 4$ and $z = 0.1$. These parameters impact the number of steps when convergence is detected, not the time. Thus they affect both algorithms in the same way, as the algorithms run the same MC.

Results. In Fig. 5 we show the perturbation score as function of the normalized time to convergence for the slowest algorithm to converge, which was always FastBall. Results for other datasets are in App. A.2, due to space limitations. HomeRun satisfies the convergence condition earlier in time, maintaining the speedups we observed in the previous experiments. We remark that the number of *steps*, i.e., of Curveball trades, performed by each algorithm before reaching convergence, is (up to $\pm w$, as we only check for convergence every w steps) the same for both algorithms (plots in App. A.2), which is not surprising, as they are both running the same Markov chain, but HomeRun performs each step faster: we are really simulating a sprint race with this experiment, as both algorithms have to cover the same "distance" along the same "track" and the first one to the "finish line" is the winner. As the figures show, the winner is HomeRun.

6 Conclusion and future directions

We introduce HomeRun, an algorithm to perform Curveball trades. Curveball trades are the key operation of many procedures for the statistically-sound analysis of datasets, from binary matrices to hypergraphs. Previous algorithm made multiple passes over the data, while HomeRun performs a single scan, plus some additional operations that cost less than a linear scan. This speedup is obtained through the use of reservoir sampling and strict upper bounds to the sample size. The results of our experimental evaluation show that HomeRun performs Curveball trades up to twice faster than the existing state of the art.

An interesting direction for future work involves modifying HomeRun to use skip-optimized of reservoir sampling ([43, Algorithm X], or the algorithms by Li [25]), to avoid flipping a biased coin for every element of the population, and skip directly to the next element to be inserted in the reservoir. More generally, it would be interesting to develop a version of the Curveball trade for positive-valued matrices, to speed up sampling from their spaces.

Acknowledgments

Part of this work was developed as MCC's undergraduate honors thesis in computer science at Amherst College. This work is supported in part by the National Science Foundation grant CAREER-2238693 (https://www.nsf.gov/awardsearch/showAward?AWD_ID=2238693). We acknowledge the work of members of the Amherst College Data* Mammoths on parts of the code.

References

[1] Maryam Abuissa, Alexander Lee, and Matteo Riondato. 2023. ROhAN: Row-order agnostic null models for statistically-sound knowledge discovery. *Data Mining and Knowledge Discovery* 37, 4 (01 Jul 2023), 1692–1718. doi:10.1007/s10618-023-00938-4

[2] Maryam Abuissa, Matteo Riondato, and Eli Upfal. 2025. DiNgHy: Null Models for Non-Degenerate Directed Hypergraphs. In *ECML PKDD'25*.

[3] Daniel Allendorf, Ulrich Meyer, Manuel Penschuck, and Hung Tran. 2023. Parallel global edge switching for the uniform sampling of simple graphs with prescribed degrees. *J. Parallel and Distrib. Comput.* 174 (2023), 118–129.

[4] Jacob Charles Wright Billings, Mirko Hu, Giulia Lerda, Alexey N Medvedev, Francesco Mottes, Adrian Onicas, Andrea Santoro, and Giovanni Petri. 2019. Simplex2vec embeddings for community detection in simplicial complexes. *arXiv preprint arXiv:1906.09068* (2019).

[5] Corrie J. Carstens. 2015. Proof of uniform sampling of binary matrices with fixed row sums and column sums for the fast curveball algorithm. *Physical Review E* 91, 4 (2015), 042812.

[6] Corrie Jacobien Carstens, Annabell Berger, and Giovanni Strona. 2018. A unifying framework for fast randomization of ecological networks with fixed (node) degrees. *MethodsX* 5 (2018), 773–780.

[7] Corrie Jacobien Carstens, Michael Hamann, Ulrich Meyer, Manuel Penschuck, Hung Tran, and Dorothea Wagner. 2018. Parallel and I/O-efficient Randomisation of Massive Networks using Global Curveball Trades. In *26th Annual European Symposium on Algorithms (ESA 2018)*.

[8] Corrie Jacobien Carstens and Pieter Kleer. 2018. Speeding up switch Markov chains for sampling bipartite graphs with given degree sequence. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*. 36:1–36:18.

[9] Philip Chodrow and Andrew Mellor. 2020. Annotated hypergraphs: models and applications. *Applied network science* 5, 9 (2020).

[10] Philip S. Chodrow. 2020. Configuration models of random hypergraphs. *Journal of Complex Networks* 8, 3 (2020), cnaa018.

[11] George W. Cobb and Yung-Pin Chen. 2003. An application of Markov chain Monte Carlo to community ecology. *The American Mathematical Monthly* 110, 4 (2003), 265–288.

[12] Péter L Erdős, Catherine Greenhill, Tamás Róbert Mezei, István Miklós, Dániel Soltész, and Lajos Soukup. 2022. The mixing time of switch Markov chains: a unified approach. *European Journal of Combinatorics* 99 (2022), 103421.

[13] Egil Ferkingstad, Lars Holden, and Geir Kjetil Sandve. 2015. Monte Carlo Null Models for Genomic Data. *Statist. Sci.* 30, 1 (2015), 59–71.

[14] Rico Fischer, Jorge C Leitao, Tiago P Peixoto, and Eduardo G Altmann. 2015. Sampling motif-constrained ensembles of networks. *Physical review letters* 115, 18 (2015), 188701.

[15] Bailey K. Fosdick, Daniel .B Larremore, Joel Nishimura, and Johan Ugander. 2018. Configuring random graph models with fixed degree sequences. *Siam Review* 60, 2 (2018), 315–355.

[16] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Bay Vo, Tin Truong Chi, Ji Zhang, and Hoai Bac Le. 2017. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 4 (2017), e1207.

[17] Alex M. Fout. 2022. *New Methods for Fixed-Margin Binary Matrix Sampling, Fréchet Covariance, and MANOVA Tests for Random Objects in Multiple Metric Spaces.* Ph. D. Dissertation. Colorado State University.

[18] James H. Fowler. 2006. Connecting the Congress: A Study of Cosponsorship Networks. *Political Analysis* 14, 04 (2006), 456–487. doi:10.1093/pan/mpl002

[19] Aristides Gionis, Heikki Mannila, Taneli Mielikäinen, and Panayiotis Tsaparas. 2007. Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 3 (2007), 14.

[20] Karl Godard and Zachary P. Neal. 2022. fastball: A fast algorithm to randomly sample bipartite graphs with fixed degree sequences. *Journal of Complex Networks* 10, 6 (2022), cnac049.

[21] Ravi Kannan, Prasad Tetali, and Santosh Vempala. 1999. Simple Markov-chain algorithms for generating bipartite graphs and tournaments. *Random Structures & Algorithms* 14, 4 (1999), 293–308.

[22] Donald E. Knuth. 1998. *Seminumerical algorithms* (3 ed.). The Art of Computer Programming, Vol. 2. Addison Wesley.

[23] Erich Leo Lehmann and Joseph P. Romano. 2022. *Testing Statistical Hypotheses* (4 ed.). Springer.

[24] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[25] Kim-Hung Li. 1994. Reservoir-Sampling Algorithms of Time Complexity $O(n(1+log(N/n)))$. *ACM Transactions on Mathematical Software (TOMS)* 20, 4 (1994), 481–493.

[26] Qi Luo, Zhenzhen Xie, Yu Liu, Dongxiao Yu, Xiuzhen Cheng, Xuemin Lin, and Xiaohua Jia. 2024. Sampling hypergraphs via joint unbiased random walk. *World Wide Web* 27, 2 (2024), 15.

[27] Jeffrey W. Miller and Matthew T. Harrison. 2013. Exact sampling and counting for fixed-margin matrices. *The Annals of Statistics* 41, 3 (June 2013). doi:10.1214/13-aos1131

[28] Leonardo Pellegrina, Matteo Riondato, and Fabio Vandin. 2019. Hypothesis Testing and Statistically-sound Pattern Mining. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. ACM, New York, NY, USA, 3215–3216. doi:10.1145/3292500.3332286

[29] Giulia Preti, Gianmarco De Francisci Morales, and Matteo Riondato. 2022. Alice and the Caterpillar: A More Descriptive Null Models for Assessing Data Mining Results. In *Proceedings of the 22nd IEEE International Conference on Data Mining*. 418–427. doi:10.1109/ICDM54844.2022.00052

[30] Giulia Preti, Gianmarco De Francisci Morales, and Matteo Riondato. 2024. Impossibility result for Markov chain Monte Carlo sampling from microcanonical bipartite graph ensembles. *Physical Review E* 109, 5 (2024), L053301.

[31] Giulia Preti, Adriano Fazzone, Giovanni Petri, and Gianmarco De Francisci Morales. 2024. Higher-order null models as a lens for social systems. *Physical Review X* 14, 3 (2024), 031032.

[32] Giulia Preti, Matteo Riondato, Aristides Gionis, and Gianmarco De Francisci Morales. 2025. Polaris: Sampling from the Multigraph Configuration Model with Prescribed Color Assortativity. In *Proceedings of the 18th ACM International Conference on Web Search and Data Mining (WSDM '25)*. 30–39.

[33] Raissa T. Relator, Aika Terada, and Jun Sese. 2018. Identifying statistically significant combinatorial markers for survival analysis. *BMC medical genomics* 11, 2 (2018), 31.

[34] Matteo Riondato. 2023. Statistically-sound Knowledge Discovery from Data. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*. Society for Industrial and Applied Mathematics, 949–952. doi:10.1137/1.9781611977653.ch107

[35] Anna Ritz, Brendan Avent, and T. M. Murali. 2017. Pathway Analysis with Signaling Hypergraphs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 14, 5 (Sept. 2017), 1042–1055. doi:10.1109/tcbb.2015.2459681

[36] Herbert John Ryser. 1963. *Combinatorial Mathematics.* American Mathematical Society.

[37] Fabio Saracco, Riccardo Di Clemente, Andrea Gabrielli, and Tiziano Squartini. 2015. Randomizing bipartite networks: the case of the World Trade Web. *Scientific reports* 5, 1 (2015), 1–18.

[38] Fabio Saracco, Giovanni Petri, Renaud Lambiotte, and Tiziano Squartini. 2024. Entropy-based random models for hypergraphs. arXiv:2207.12123 [cs.SI] https://arxiv.org/abs/2207.12123

[39] Jun Sese, Aika Terada, Yuki Saito, and Koji Tsuda. 2014. Statistically significant subgraphs for genome-wide association study. In *Statistically Sound Data Mining*. 29–36.

[40] Giovanni Strona, Domenico Nappo, Francesco Boccacci, Simone Fattorini, and Jesus San-Miguel-Ayanz. 2014. A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. *Nature communications* 5, 1 (2014), 4114.

[41] Lionel Tabourier, Camille Roth, and Jean-Philippe Cointet. 2011. Generating constrained random graphs using multiple edge switches. *Journal of Experimental Algorithmics* 16, 1 (2011).

[42] Norman D. Verhelst. 2008. An efficient MCMC algorithm to sample binary matrices with fixed marginals. *Psychometrika* 73, 4 (2008), 705–728.

[43] Jeffrey S. Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Software* 11, 1 (1985), 37–57.

[44] Guanyang Wang. 2020. A fast MCMC algorithm for the uniform sampling of binary matrices with fixed margins. *Electronic Journal of Statistics* 14 (2020), 1690–1706.

[45] Albrecht Zimmermann. 2014. The Data Problem in Data Mining. *SIGKDD Explor.* 16, 2 (2014), 38–45.

A Appendix

A.1 Postponed proofs

We now prove the technical Lemma 4.2.

Proof of Lemma 4.2. We first show that, once the loop on lines 6–26 has completed, the reservoir r contains exactly $q_{\mathbf{a},\mathbf{b}}$ elements.

We use $\hat{q}'_i$ and $\hat{q}''_i$, for $i \in [n]$, to denote the value of $\hat{q}'$ and $\hat{q}''$ at the end of i^{th} iteration of the loop on lines 6–26. We use $\hat{q}'_0$ and $\hat{q}''_0$ to denote the initial values of these variables, which are respectively $\|\mathbf{a}\|_1$ and $n - \|\mathbf{b}\|_1$ (lines 1 and 2). Our goal is to show that $\min\{\hat{q}'_n, \hat{q}''_n\} = q_{\mathbf{a},\mathbf{b}}$. We are actually going to show a stronger result, i.e., that, for any $i \in \{0, \ldots, n\}$, it holds

$$\begin{aligned} \hat{q}'_i &= q_{\mathbf{a},\mathbf{b}} + \left|\left\{i+1 \le j \le n : a_j = 1 \wedge b_j = 1\right\}\right| \\ \hat{q}''_i &= q_{\mathbf{a},\mathbf{b}} + \left|\left\{i+1 \le j \le n : a_j = 0 \wedge b_j = 0\right\}\right| . \end{aligned} \tag{1}$$

These identities clearly imply $\min\{\hat{q}'_n, \hat{q}''_n\} = q_{\mathbf{a},\mathbf{b}}$. We show that they are true by induction on i.

The base case for $i = 0$ is immediate by definition of the initial values for the bounds.

Fix now any $0 < k \le n$, and assume that the identities in Eq. (1) hold for every $0 \le i < k$. If $a_k \ne b_k$ (lines 7–16), HomeRun does not modify the values of $\hat{q}'$ and $\hat{q}''$. The invariant is maintained because in this case

$$\begin{aligned} \hat{q}'_k = \hat{q}'_{k-1} &= q_{\mathbf{a},\mathbf{b}} + \left|\left\{k \le j \le n : a_j = 1 \wedge b_j = 1\right\}\right| \\ &= q_{\mathbf{a},\mathbf{b}} + \left|\left\{k+1 \le j \le n : a_j = 1 \wedge b_j = 1\right\}\right|, \end{aligned} \tag{2}$$

and similarly for $\hat{q}''_k = \hat{q}''_{k-1}$.

If instead $a_k = b_k$ (lines 17–26), HomeRun decreases $\hat{q}'$ by one if $a_k = 1$ (line 19), and $\hat{q}''$ otherwise (line 20). In the first case ($a_k = 1$), the invariant clearly still holds for $\hat{q}''$, following the same reasoning as in Eq. (2). In this case, for $\hat{q}'$, we have

$$\begin{aligned} \hat{q}'_k = \hat{q}'_{k-1} - 1 &= q_{\mathbf{a},\mathbf{b}} + \left|\left\{k \le j \le n : a_j = 1 \wedge b_j = 1\right\}\right| - 1 \\ &= q_{\mathbf{a},\mathbf{b}} + \left|\left\{k+1 \le j \le n : a_j = 1 \wedge b_j = 1\right\}\right|, \end{aligned}$$

thus the invariant also holds. In the second case ($a_k = 0$), a similar reasoning leads to the same conclusion.

Since r has always size equal to $\min\{\hat{q}', \hat{q}''\}$, then it contains the correct number of elements $q_{\mathbf{a},\mathbf{b}}$ at the end of the loop on lines 6–26. We now show that, at the end of the loop, the elements in r are a uniform sample of $\mathcal{P}_{\mathbf{a},\mathbf{b}}$ of this size.

If at any point $\min\{\hat{q}', \hat{q}''\}$ becomes zero, then r also has size zero. From the first part of the proof we then know that it must be $q_{\mathbf{a},\mathbf{b}} = 0$, thus indeed r "contains" a uniform sample of $\mathcal{P}_{\mathbf{a},\mathbf{b}}$ of size zero, i.e., the empty set.

Assume then that $\min\{\hat{q}', \hat{q}''\} \ne 0$ throughout the whole execution of the algorithm. We now show that at the end of every iteration i of the loop, r contains a uniform sample of size $\min\{\hat{q}', \hat{q}''\}$ of the t elements of the population seen so far, if $t > \min\{\hat{q}', \hat{q}''\}$, or all t of them, for $t \ge \min\{\hat{q}', \hat{q}''\}$, where again we are considering the value of t at the end of iteration i of the loop. We call this Fact A, and prove it by induction on i.

At the first iteration of the loop, i.e., for $i = 1$, if $1 \in \mathcal{P}_{\mathbf{a},\mathbf{b}}$ (i.e., $a_1 \ne b_1$), then t takes value 1 on line 8, and since r has size at least 1 (from our assumption above), then $r[1]$ takes value 1, and is the only element in r, which therefore contains all the elements (i.e., the only element) of the population seen so far. If instead $a_1 = b_1$, i.e., $1 \notin \mathcal{P}_{\mathbf{a},\mathbf{b}}$, r remains empty, so it still contains all the (zero) elements of the population seen so far. Thus, our base case is complete.

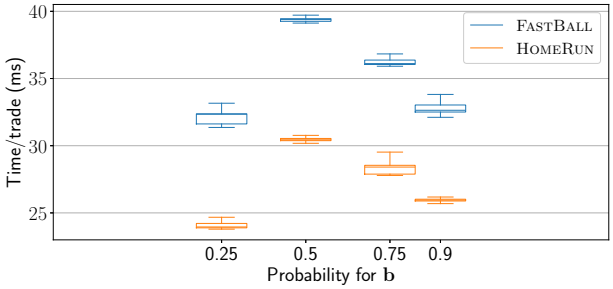
Fix now any $1 < k \le n$, and assume that Fact A holds for $1 \le i < k$.

Let us first consider the case $t \le \min\{\hat{q}', \hat{q}''\}$. Similarly to the base case, if $k \in \mathcal{P}_{\mathbf{a},\mathbf{b}}$ (i.e., $a_k \ne b_k$), then t must have had value strictly less than $\min\{\hat{q}', \hat{q}''\}$ at the beginning of the iteration, and was incremented by one on line 8. In this case, the condition on line 9 is true, and k is added to r in position t. If $k \notin \mathcal{P}_{\mathbf{a},\mathbf{b}}$ (i.e., $a_k = b_k$), then again like in the base case, r is not modified, and it holds from the inductive hypothesis that it contains all the t elements of the population seen so far.

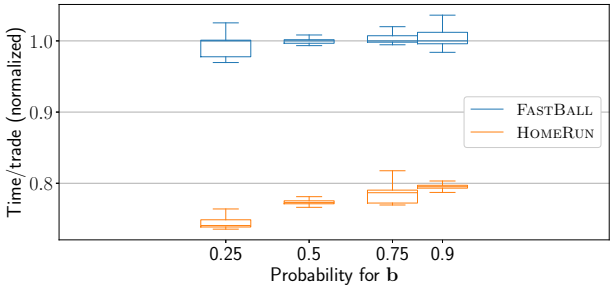
Consider now the case $t > \min\{\hat{q}', \hat{q}''\}$. If $k \in \mathcal{P}_{\mathbf{a},\mathbf{b}}$ (i.e., $a_k \ne b_k$), then Fact A follows from the correctness of reservoir sampling, i.e., Lemma 3.1, as HomeRun follows the reservoir sampling approach on lines 10–16. Otherwise, it follows from Fact 3.2, as HomeRun shrinks r by removing an element in it uniformly at random on lines 23–26. □

A.2 Additional experimental results

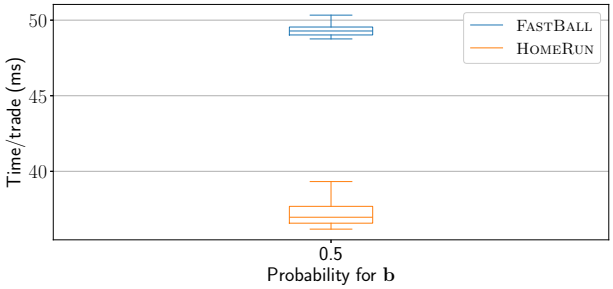
See the following pages.



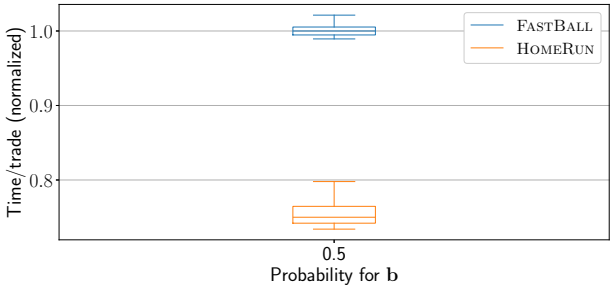
(a) Raw time — $p_a = 0.25$, $n = 1000000$.



(b) Normalized time — $p_a = 0.25$, $n = 1000000$.

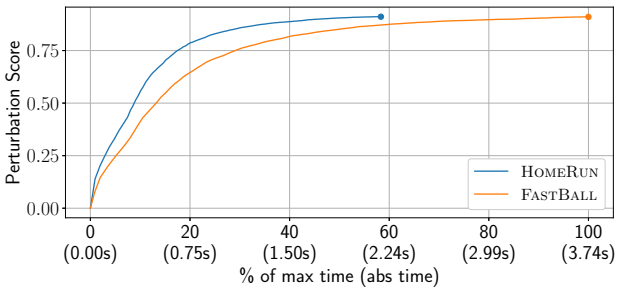


(c) Raw time — $p_a = 0.5$, $n = 1000000$.

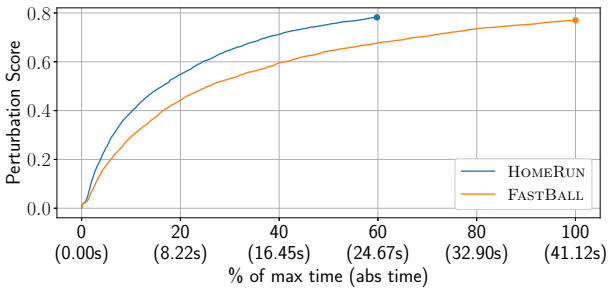


(d) Normalized time — $p_a = 0.5$, $n = 1000000$.

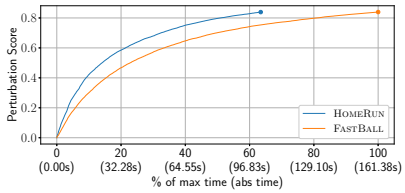
Figure 6: Times to perform a Curveball trade vs. probability p_b for b.



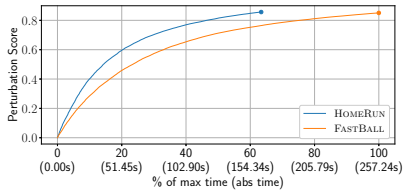
(a) wiki-Vote



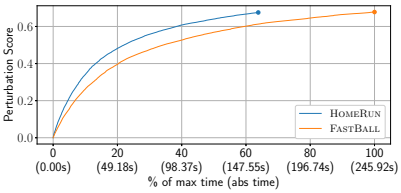
(b) sx-mathoverflow



(c) soc-Epinions1

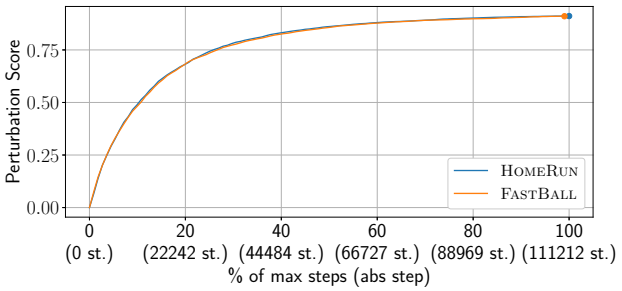


(d) ego-twitter

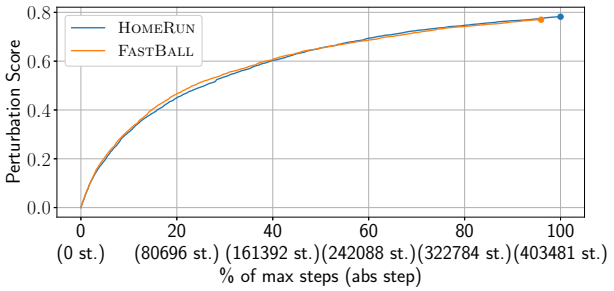


(e) sx-superuser-c2a

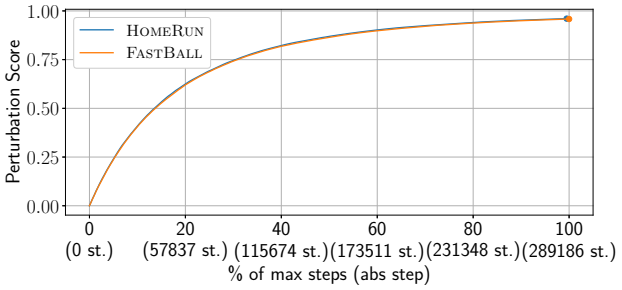
Figure 7: Perturbation score vs. cumulative wall-clock time before convergence.



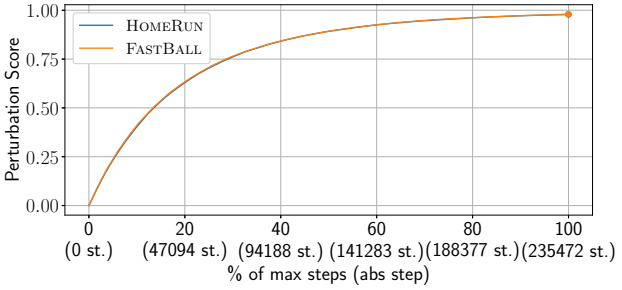
(a) wiki-Vote



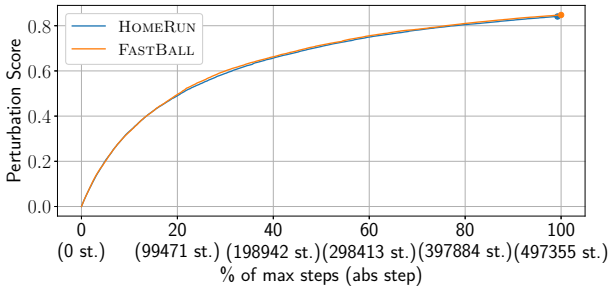
(b) sx-mathoverflow



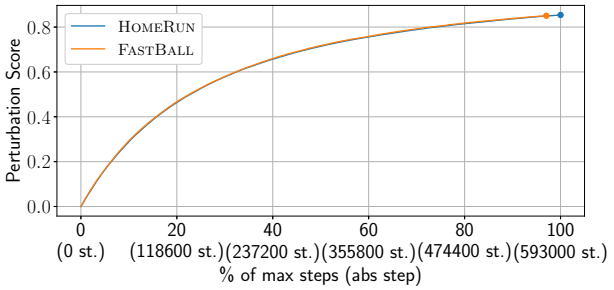
(c) cit-HepPh



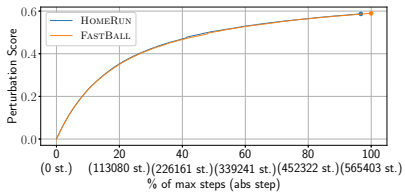
(d) p2p-Gnutella31



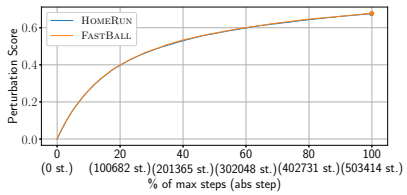
(e) soc-Epinions1



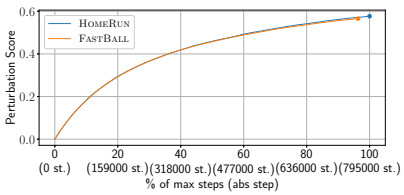
(f) ego-tiwtter



(g) sx-superuser-c2q



(h) sx-superuser-c2a



(i) sx-superuser

Figure 8: Perturbation score vs. cumulative steps before convergence.