fastball: A fast algorithm to sample binary matrices with fixed marginals

Karl Godard\textsuperscript{1} and Zachary P. Neal\textsuperscript{2,*}

\textsuperscript{1}Math Department, University of Michigan
\textsuperscript{2}Psychology Department, Michigan State University
*Corresponding author: zpneal@msu.edu

December 9, 2021

Abstract

Many applications require randomly sampling binary graphs with fixed degrees, or randomly sampling binary matrices with fixed row and column marginals. Although several algorithms to perform this sampling have been proposed, the “curveball” algorithm is the fastest that has been proven to sample uniformly at random. In this paper, we introduce a more efficient version of curveball called “fastball” that is roughly 8 times faster.

1 Introduction

Let $M$ be the space of all $n \times m$ binary matrices $M$ with row sums $R = r_1, r_2 \ldots r_n$ and column sums $C = c_1, c_2 \ldots c_m$. The need to randomly sample $M \in M$ with uniform probability arises in many different graph-theoretic contexts, where such samples are often used to build a null distribution of some graph property of interest, such as nestedness [5] or co-occurrence [12, 14]. When $M$ is square, it can represent a directed graph with specific in-degree and out-degree sequences [18, 16]. Alternatively, when $M$ is rectangular, it can represent a bipartite graph with specific row-node and column-node degree sequences [12, 9].

Several methods have been proposed for drawing such samples, and have been described from both matrix and graph perspectives. One simple approach involves filling an initially empty matrix with 0s and 1s following the Gale-Ryser theorem [11, 17], or adding edges between vertices in an initially empty graph via the configuration model [3]. An alternative Markov chain Monte Carlo (MCMC) approach involves repeatedly swapping checkerboard patterns (e.g., swapping $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ with $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$) in a matrix, or re-wiring edges in a graph [4]. Other more sophisticated methods have been proposed that rely on sequential
importance sampling [1, 8, 20] or simulated annealing [2]. Among these methods, the fastest [19, 7] that is proven to sample \( M \in \mathcal{M} \) uniformly at random [6] is the “curveball” algorithm. In this paper, we introduce a more efficient variant of curveball that we call “fastball”.

The remainder of the paper is organized in four sections. In section 2 we briefly describe the curveball algorithm. In section 3 we introduce the fastball algorithm, which is implemented in C++ and available in R via a wrapper. In section 4 we compare the running times of curveball and fastball to sample \( M \in \mathcal{M} \) for varying sizes of \( M \). We conclude in section 5 by briefly describing a potential application of fastball, and identifying directions for future research.

## 2 The curveball algorithm

Figure 1 outlines the steps of the curveball algorithm, which illustrates how a starting matrix \( M \) is randomized to yield a new matrix \( M' \) that is randomly chosen from \( \mathcal{M} \). The algorithm developers noted that this randomization process resembles how children may trade baseball cards, giving the algorithm its name, so we use this metaphor to make the example concrete [19]. The input is a binary matrix \( M \). We use uppercase letters A–D to denote the rows (i.e., children), and lowercase letters a–f to denote the columns (i.e., baseball cards).

In step (1), \( M \) is transformed into a list of the columns containing 1s in each row; for example, child A has cards a, c, e, and f. In step (2), two rows are randomly selected; for example, children A and B will engage in card trading. In step (3), the intersection \( I \) (i.e., cards held by both children), union \( U \) (i.e., cards held by either child), and symmetric difference \( S \) (i.e., cards held by only one child) of the two lists are computed, together with the cardinality of each list’s contribution to the symmetric difference \( S_A \) and \( S_B \) (i.e., each child’s number of unique cards). In step (4), the symmetric difference is shuffled, then the first \( S_A \) elements are placed in A’s list, while the last \( S_B \) elements are placed in B’s list. In the card trading metaphor, this mirrors the children placing their unique cards in a pile, then randomly drawing the same number of cards from
Algorithm 1 Curveball algorithm

Input: Adjacency matrix $M$ $(r \times c)$
Returns: Randomized matrix $M'$ $(r \times c)$ with the same row and column sums as $M$

Let $L$ be, for each row of $M$, a list of columns containing a 1
for $i \in \{0, 1, 2, ..., T\}$ do
    Select two random rows, $R_1$ and $R_2$, from $L$
    Let $I = R_1 \cup R_2$
    Let $U = R_1 \cap R_2$
    Let $S = U \setminus I$
    Shuffle elements of $S$
    Let $S_1$ be the first $|R_1| - |I|$ elements of $S$
    Let $S_2$ be the last $|R_2| - |I|$ elements of $S$
    Form new row $N_1 = I + S_1$
    Form new row $N_2 = I + S_2$
    In $M$, write $N_1$ in place of $R_1$ and $N_2$ in place of $R_2$
end for

the pile that they put in. In this example, while A uniquely held cards a, c, and e before the trade, after the trade he uniquely holds cards a, b, and d. In step (5), each list’s new unique elements is recombined with its non-unique elements; for example, after the trade child A has cards a, b, and d (from the trade) and f (not eligible for trading). In step (6), these two lists are recombined with the lists of the remaining rows; for example, these two children rejoin the group of other children. Steps (2)–(6) are repeated $T$ times, each time randomly choosing two new rows (i.e., the children engage in $T$ dyadic trading sessions). Numerical experiments suggest that $T = 5n$ times is sufficient to ensure random mixing in matrices up to $100 \times 100$ [19, 6, 7]. In the final step (7), the resulting list of locations of 1s is transformed back into a binary matrix $M'$. When the goal is to draw $N$ random samples of $M' \in \mathcal{M}$, steps (1)–(7) are repeated $N$ times.

Algorithm 1 formally summarizes the steps of the curveball algorithm [19].

The curveball algorithm offers several advantages over alternative approaches to randomizing a binary matrix, or to randomly sampling $M \in \mathcal{M}$. First, curveball has a faster mixing time than checkerboard swap methods because while swap methods can only make one swap and thus can only alter the positions of two 1s and two 0s in each iteration, a single curveball trade can involve many such swaps. The example illustrated in Figure 1 shows how four 1s and four 0s were swapped in one trade (shaded in gray). Second, curveball has been proven to sample $M \in \mathcal{M}$ uniformly at random by proving that its Markov Chain is finite, irreducible, and aperiodic [6, 13]. Finally, curveball has been implemented in both R and Python [19], making it readily available to many users.
3 The fastball algorithm

The fastball algorithm modifies curveball in four ways that retain its advantages, while improving its computational efficiency. Figure 2 outlines the steps of the fastball algorithm, while algorithm 2 formally summarizes these.

First, fastball is implemented in C++, which is generally more efficient than higher-level languages such as R or Python. The R wrapper function fastball(), which is integrated into the backbone package [9] via Rcpp [10], makes this C++ implementation available in a convenient form.

Second, although both curveball and fastball sample from $M$ by randomizing $M$ to yield $M'$, unlike curveball which transforms $M$ into a list $L$ when each sample is drawn, fastball performs this operation only once outside the sampling loop. In Figure 2, the dashed line marking the operations performed to draw each sample from $M$ excludes step (1), and similarly in algorithm 2 the input is a list and not a matrix. This improves the efficiency of sampling $N$ matrices from $M$ because while curveball must perform $M \rightarrow L$ $N$ times, fastball performs it only one time.

Third, fastball requires fewer and simpler set operations than curveball in step (3) of Figure 2. Specifically, in step (3), the cardinality of the intersection $I$ (i.e., the number of cards held by both children) of the two lists is computed, and from this the number of entries unique to list $A (S_A)$ and to list $B (S_B)$ are computed (i.e., each child’s number of unique cards). In addition, a random vector $V$ is constructed that contains $S_A$ As and $S_B$ Bs; we call this the ‘victory...
Algorithm 2 Fastball algorithm

**Input:** List of lists $L$ containing, for each row of an adjacency matrix $M$, a list of columns containing a 1

**Returns:** Randomized matrix $M'$ with the same row and column sums as $M$

for $i \in \{0, 1, 2, ..., T\}$ do
    Select two random rows, $R_1$ and $R_2$, from $L$
    Let $I = |R_1 \cap R_2|$
    Let $S_1 = |R_1| - I$
    Let $S_2 = |R_2| - I$
    Let $V$ be a vector with $|S_1|$ 1’s and $|S_2|$ 2’s in random positions
    Let $a = 0$, $b = 0$, $c = 0$
    while $a \neq |R_1|$ and $b \neq |R_2|$ do
        if $R_1[a] = R_2[b]$ then
            Append $R_1[a]$ to $N_1$ and append $R_2[b]$ to $N_2$
            Increment $a$ and $b$
        else if $R_1[a] < R_2[b]$ then
            Let $n = V[c]$
            Append $R_1[a]$ to $N_n$
            Increment $a$ and $c$
        else if $R_1[a] > R_2[b]$ then
            Let $n = V[c]$
            Append $R_2[b]$ to $N_n$
            Increment $b$ and $c$
        end if
    end while
    In $L$, replace $R_1$ with $N_1$ and replace $R_2$ with $N_2$
end for

vector’ for reasons that are clear in the next step. This step is more efficient because it does not require enumerating the contents of any sets, but relies only on cardinalities.

Finally, fastball uses a different method for performing the swaps in step (4) of Figure 2, which corresponds to a different way that children might trade baseball cards. Specifically, in step (4), elements from the two sorted lists are compared pairwise. If the two elements do not match, then the corresponding entry of $V$ is used to determine which list the lower-valued element is placed in (see Rounds 1–5 in Figure 2; the else if statements in Algorithm 2). If the two elements do match, then the elements are retained in both lists (see Round 6, and the if statement). This described a card trading game in which two children begin with their alphabetically sorted card decks, and each places a card face up. If the cards match, both children retain the face up card (i.e., there is no trade), and they both place a new card face up. If the cards do not match, the ‘victory vector’ determines which child wins the alphabetically-earlier card, while the losing child turns a new card face up from their deck. This step is more efficient
Table 1: Running times of curveball and fastball algorithms to randomly sample 100 \( \mathbf{M} \in \mathcal{M} \).

| Matrix Dimensions | Running time (in seconds) | Times |
|-------------------|---------------------------|-------|
| 500 \times 500    | 11.17                     | 1.25  | 8.94 |
| 1000 \times 1000  | 37.23                     | 4.79  | 7.77 |
| 5000 \times 5000  | 833.14                    | 117.41| 7.10 |
| 10000 \times 10000| 3828.04                   | 470.96| 8.13 |

because it exploits C++’s efficiency in comparing elements of sorted lists.

Despite these four modifications, the outcome of the trades performed by fastball are identical to those performed by curveball. Therefore, fastball’s Markov Chain is also finite, irreducible, and aperiodic, and samples \( \mathbf{M} \in \mathcal{M} \) uniformly at random [6, 13].

4 Comparison of running times

To evaluate the improvement in running time achieved by fastball compared to curveball, we used each algorithm to randomly sample 100 \( \mathbf{M} \in \mathcal{M} \), for four sizes of \( \mathbf{M} \). Table 1 shows, for each size of \( \mathbf{M} \), each algorithm’s running time in seconds on an Apple M1 Max processor. Although these running times are processor-dependent, we are primarily interested in the relative speed of fastball compared to curveball, which is reported in the last column. We observe that fastball is roughly 8 times faster than curveball when randomly sampling large matrices.

5 Discussion

In this paper, we have introduced the fastball algorithm as a faster variant of the existing curveball algorithm for sampling binary matrices with fixed marginals. The fastball algorithm randomly samples matrices \( \mathbf{M} \) with uniform probability from the space of all matrices with given marginals \( \mathcal{M} \) roughly 8 times faster than curveball. This more efficient algorithm has particular application for graph pattern detection, where random graphs must be generated to build a null distribution of a graph property such as nestedness [5] or co-occurrence [12, 14]. The fastball algorithm has been implemented in C++, and is available in R using the \texttt{fastball()} function in the \texttt{backbone} package [9].

Although there are many potential applications of fastball, we briefly describe one application for which its efficiency is particularly useful. Consider a bipartite graph \( \mathbf{B} \) that records whether agent \( i \) is associated with artifact \( k \), and its bipartite projection \( \mathbf{P} = \mathbf{BB}' \), where the weight of the edge between agents \( i \) and \( j \) \( (P_{ij}) \) is equal to the number of artifacts \( k \) with which they are both associated. One common goal is determining whether \( P_{ij} \) is statistically
significantly larger than expected at random. For example, one may infer that legislators $i$ and $j$ are political allies if they collaborate on significantly more bills than expected at random [14], or that species $i$ and $j$ are symbiotic if they live together in significantly more locations than expected at random [12]. The fixed degree sequence model (FDSM) makes this determination by comparing $P_{ij}$ to $P^*_{ij}$ obtained from the projection of a random bipartite graph $B^*$ sampled from $B$, the space of all bipartite matrices with the same degree sequences as $B$ [9]. In this application, there are two reasons that efficient sampling of $B^* \in B$ is essential. First, each time a $B^*$ is sampled, its projection must be constructed to compute $P^*_{ij}$, which is itself computationally intensive. Second, when a familywise error rate correction is performed to control for the inflated Type-I error introduced by testing every edge weight $P_{ij}$, the number of samples of $B^*$ that must be drawn may be very large [15].

In addition to improving the efficiency of matrix and graph sampling, this work also offers a starting point for two promising directions for further research. First, relatively little is known about curveball’s or fastball’s mixing time. Prior work has shown that their Markov Chain is rapidly mixing whenever checkerboard swap algorithms are rapidly mixing [7] and that $5n$ trades is sufficient for random mixing when $M$ is small (e.g., $100 \times 100$) and random. Because fastball can perform trades much more quickly than curveball, it can be used to conduct numerical experiments of mixing time on larger matrices or matrices with unique structures. Second, although parametric null distributions of some graph properties are known under some types of marginal constraints (e.g., co-occurrence when row marginals are fixed [15]), parametric null distributions are unknown under fixed-marginal constraints. Fastball’s ability to efficiently and randomly sample matrices with fixed-marginal constraints makes it possible to generate smooth empirical null distributions, which may shed light on their parametric forms. However, in the absence of precise mixing times or parametric null distributions, fastball’s speed makes it practical to perform a very large number of trades and to draw a very large number of samples from which empirical null distributions can be constructed.

**Contributor Statement**

K.G. conceptualized, drafted, and implemented the fastball algorithm in C++; Z.P.N. tested the algorithm and implemented a wrapper in R. Both authors drafted and revised the paper.

**Acknowledgements**

This work was supported by the National Science Foundation (#2016320).
References

[1] R. Admiraal and M. S. Handcock. Networksis: a package to simulate bipar-
tite graphs with fixed marginals through sequential importance sampling. Journal of Statistical Software, 24(8), 2008.

[2] I. Bezáková, N. Bhatnagar, and E. Vigoda. Sampling binary contingency tables with a greedy start. Random Structures & Algorithms, 30(1-2):168–205, 2007.

[3] J. Blanchet and A. Stauffer. Characterizing optimal sampling of binary contingency tables via the configuration model. Random Structures & Algorithms, 42(2):159–184, 2013. doi:10.1002/rsa.20403.

[4] A. A. Boroojeni, J. Dewar, T. Wu, and J. M. Hyman. Generating bipartite networks with a prescribed joint degree distribution. Journal of complex networks, 5(6):839–857, 2017. doi:10.1093/comnet/cnx014.

[5] M. Bruno, F. Saracco, D. Garlaschelli, C. J. Tessone, and G. Caldarelli. The ambiguity of nestedness under soft and hard constraints. Scientific reports, 10(1):1–13, 2020.

[6] 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(4):042812, 2015. doi:10.1103/PhysRevE.91.042812.

[7] C. J. Carstens and P. Kleer. 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). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.36.

[8] Y. Chen, I. H. Dinwoodie, and S. Sullivant. Sequential importance sampling for multiway tables. The Annals of Statistics, 34(1):523–545, 2006. doi:10.1214/009053605000000822.

[9] R. Domagalski, Z. P. Neal, and B. Sagan. Backbone: An r package for extracting the backbone of bipartite projections. Plos one, 16(1):e0244363, 2021. doi:10.1371/journal.pone.0244363.

[10] D. Eddelbuettel, R. François, J. Allaire, K. Ushey, Q. Kou, N. Russel, J. Chambers, and D. Bates. Rcpp: Seamless r and c++ integration. Journal of statistical software, 40(8):1–18, 2011.

[11] D. Gale et al. A theorem on flows in networks. Pacific J. Math, 7(2):1073–1082, 1957. doi:10.2140/pjm.1957.7.1073.

[12] N. J. Gotelli. Null model analysis of species co-occurrence patterns. Ecology, 81(9):2606–2621, 2000. doi:10.1890/0012-9658(2000)081[2606: NMAOSC]2.0.CO;2.
[13] M. Mitzenmacher and E. Upfal. Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge university press, 2017.

[14] Z. P. Neal. The backbone of bipartite projections: Inferring relationships from co-authorship, co-sponsorship, co-attendance and other co-behaviors. Social Networks, 39:84–97, 2014. doi:10.1016/j.socnet.2014.06.001.

[15] Z. P. Neal, R. Domagalski, and B. Sagan. Comparing alternatives to the fixed degree sequence model for extracting the backbone of bipartite projections. Scientific Reports, 2021.

[16] M. E. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. Physical review E, 64(2):026118, 2001. doi:10.1103/PhysRevE.64.026118.

[17] H. J. Ryser. Combinatorial properties of matrices of zeros and ones. Canadian Journal of Mathematics, 9:371–377, 1957. doi:10.4153/CJM-1957-044-3.

[18] T. A. Snijders. Enumeration and simulation methods for 0–1 matrices with given marginals. Psychometrika, 56(3):397–417, 1991. doi:10.1007/BF02294482.

[19] 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(1):1–9, 2014. doi:10.1038/ncomms5114.

[20] N. D. Verhelst. An efficient mcmc algorithm to sample binary matrices with fixed marginals. Psychometrika, 73(4):705–728, 2008.