Combinatorics, Probability and Computing (2021), **30**, pp. 124–135
doi:10.1017/S0963548320000292

CAMBRIDGE
UNIVERSITY PRESS

ARTICLE

Approximately counting bases of bicircular matroids

Heng Guo[1,*] and Mark Jerrum[2,†]

[1]School of Informatics, University of Edinburgh, Informatics Forum, Edinburgh EH8 9AB, UK and [2]School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London E1 4NS, UK
[*]Corresponding author. Email: hguo@inf.ed.ac.uk

(Received 27 January 2019; revised 16 May 2020; accepted 15 May 2020; first published online 6 August 2020)

Abstract
We give a fully polynomial-time randomized approximation scheme (FPRAS) for the number of bases in bicircular matroids. This is a natural class of matroids for which counting bases exactly is #**P**-hard and yet approximate counting can be done efficiently.

2020 MSC Codes: Primary 68W20 Randomized algorithms, 60J10 Markov chains (discrete-time Markov processes on discrete state spaces); Secondary 05C31 Graph polynomials

1. Introduction

We introduce a new application of the 'popping' paradigm that has been used to design efficient perfect samplers for a number of combinatorial structures. Existing examples are cycle popping [25, 28], sink popping [3] and cluster popping [9, 10, 11], which, respectively, produce uniformly distributed spanning trees, sink-free orientations in undirected graphs, and root-connected subgraphs in directed graphs (and, as a consequence, connected subgraphs of an undirected graph). In doing so we provide an example of a natural class of matroids for which the basis-counting problem is hard (#**P**-complete) to solve exactly, but which is polynomial-time to solve approximately in the sense of fully polynomial-time randomized approximation schemes (or FPRAS). For basic definitions connected with the complexity of counting problems refer to [23] or [14].

Towards this end, we introduce 'bicycle popping' as a means to sample, uniformly at random, bases of a bicircular matroid.[1] Bicircular matroids are associated with undirected graphs and will be defined in the next section. Note that the main result and its proof can be understood in graph-theoretic terms, and no knowledge of matroid theory is needed beyond the exchange axiom. Our perfect sampling approach can be implemented to run in $O(n^2)$ time, where n is the number of vertices in the instance graph (refer to Section 4). Using a standard reduction, such a sampler can be used to construct an efficient randomized algorithm, indeed an FPRAS, for estimating the number of bases within a specified relative error (Theorems 5.1 and 6.1).

The computational complexity of counting bases of a matroid *exactly* is still only partially understood. According to the class of matroids under consideration, the exact counting problem may be polynomial-time, #**P**-complete or unresolved. Counting bases of a graphic matroid (*i.e.* counting spanning trees of a graph) is a classical problem and is well solved by Kirchhoff's matrix

[†]The work described here was supported by the EPSRC research grant EP/N004221/1 'Algorithms that Count'.

[1]'Bicycle popping' has the advantage of being easy to remember, but it is important to note that the term is unconnected with the concept of bicycle space of a graph.

tree theorem. This method extends fairly directly to the wider class of regular matroids [22]. The basis-counting problem for bicircular matroids, a restriction of the class of transversal matroids, was shown to be #**P**-complete by Giménez and Noy [8]. The status of the important case of binary matroids appears to be open [26].

Jerrum [15] showed that it is #**P**-hard to exactly count bases of certain sparse paving matroids. Combined with the approximation algorithm of Chávez Lomelí and Welsh [2], this result highlights a (presumably) exponential gap between exact and approximate counting. However, it could be said that this example is not particularly natural. Piff and Welsh [24] demonstrated that the number of paving matroids on a ground set of n elements is doubly exponential in n, so even representing the problem instance raises significant issues. Combined with the completeness result of Giménez and Noy [8], our FPRAS provides a more convincing and natural demonstration of the gap between exact and approximate counting for matroid bases.

After posting our paper on arXiv, we were made aware of independent work by Kassel and Kenyon [18], who have proposed essentially the same algorithm[2] for sampling from a weighted distribution on cycle-rooted spanning trees. Their interest in the algorithm is as a component in their proofs, for which correctness of the algorithm is obviously important and is proved in detail. The time complexity of their algorithm is not analysed in detail, though Kassel and Kenyon offer some brief remarks about the run-time of the algorithm on a square grid. Kassel [17] also observes the connection to sampling bases of a bicircular matroid, and notes that the corresponding counting problem is #P-complete.

Even more recently, Anari, Liu, Oveis Gharan and Vinzant [1] have shown that the expansion of the so-called 'basis exchange graph' for any matroid is at least 1. This result implies that a random walk on the basis exchange graph is rapidly mixing, and provides a Markov chain Monte Carlo (MCMC) approach to sampling bases of any matroid. The mixing time has subsequently been sharpened by Cryan, Guo and Mousa [4]. The only requirement for the Markov chain approach is that there exists an efficient independence oracle to verify whether a given set is a basis. Since this requirement certainly holds for bicircular matroids, these works yield an alternative approach to sampling bases of a bicircular matroid. The Markov chain method is very different to ours and does not give a perfect sampler (though the deviation of the output distribution from uniformity decays exponentially fast in the run-time). Also, the analysis of the expansion factor of the basis exchange graph is technically challenging, while the analysis of our popping algorithm is relatively elementary. Before the work of Anari, Liu, Oveis Gharan and Vinzant [1], the basis exchange graph was known to be an expander only in special cases. Most notably, Feder and Mihail [6] showed that the class of so-called 'balanced matroids', a strict superset of the class of regular matroids, has expansion factor at least 1. (See also [16] for improvements and simplifications.) Furthermore, all paving matroids admit an FPRAS for the number of their bases, as shown by Chávez Lomelí and Welsh [2], through the straightforward Monte Carlo method.

2. Bicycle popping

For a graph $G=(V,E)$, let $n=|V|$ and $m=|E|$. When $m\geqslant n$ and G is connected, we associate a *bicircular* matroid $B(G)$ with G. The ground set is E, and a subset $R\subseteq E$ is independent if every connected component of (V,R) has at most one cycle. Thus the set of bases of $B(G)$ is

$$\mathcal{B}=\{R \mid \text{every connected component of } (V,R) \text{ is unicyclic}\}.$$

In particular, if $R\in\mathcal{B}$, then $|R|=n$. Let $\pi_{\mathcal{B}}(\cdot)$, or simply $\pi(\cdot)$, denote the uniform distribution over $\mathcal{B}$. We refer the reader to [21] for more details of bicircular matroids. Giménez and Noy [8]

[2]There are a couple of fine differences, such as not rejecting 2-cycles, and popping cycles randomly instead of deterministically. These differences do not change the nature of the algorithm.

Algorithm 1 Bicycle popping

```
Let $S$ be a subset of arcs obtained by assigning each arc $a_v$ independently and uniformly among all
   neighbours of $v$;
while a bad event $B_e$ or $B_C$ is present do
    Let Bad be the set of vertices that are contained in any edge $e$ or cycle $C$ such that $B_e$ or $B_C$
       occurs;
    Re-randomize $\{a_v \mid v \in \mathsf{Bad}\}$ to get a new $S$;
end
return the undirected version of $S$
```

have shown that counting the number of bases for bicircular matroids is #**P**-complete. See also [7] for extremal bounds on this number.

We now associate a random arc $a_v = (v, w)$ to each vertex $v \in V$, which is uniform over all neighbours w of v. Given an arbitrary assignment $\sigma = (a_v)_{v \in V}$, consider the directed graph (V, σ) with exactly those $|V|$ arcs. It is easy to see that each (weakly) connected component of this graph has the same number of arcs as vertices. Thus there is exactly one (directed) cycle per connected component. Let $U(\sigma) \subseteq E$ be the subset of edges of G obtained by dropping the direction of arcs in σ. Consider the distribution $\tau(\cdot)$ on subsets of E induced by σ via the mapping $U(\sigma)$. There are two reasons why $\tau(\cdot)$ is not quite the same as $\pi(\cdot)$.

(1) It is possible to have 2-cycles in σ, in which case at least one connected component of $U(\sigma)$ will be a tree rather than a unicyclic graph.
(2) Every cycle in σ of length greater than 2 may be reversed without changing $U(\sigma)$. Thus, in $\tau(\cdot)$, each subgraph with k connected components arises in 2^k ways, skewing the distribution towards configurations with more connected components.

For each edge $e \in E$, let B_e denote the event that a 2-cycle is present at e, that is, both orientations of e appear in σ. For each cycle C in G, we fix an arbitrary orientation and let B_C denote the event that C is oriented this way in σ. If we further condition on neither B_e nor B_C happening, the resulting distribution τ induced by $U(\sigma)$ is exactly $\pi(\cdot)$.

Partial rejection sampling [12] provides a useful framework to sample from a product distribution conditioned on a number of bad events not happening. In particular, we call a collection of bad events *extremal* if any two bad events are either probabilistically independent or disjoint (*i.e.* they cannot both occur). It is straightforward to verify that the collection of bad events $\{B_e \mid e \in E\} \cup \{B_C \mid C \text{ is a cycle in } G\}$ is extremal. (The reason is similar to the cycle popping algorithm. See [12, Section 4.2]. In fact, the bad events here are either identical to or more restrictive than those for cycle popping.) For an extremal instance, to draw from the desired distribution, we only need to randomly initialize all variables, and then repeatedly re-randomize variables responsible for occurring bad events. This is Algorithm 1, which we call 'bicycle popping'.

We need to be a little bit careful about bad events (B_C), since there are potentially exponentially many cycles in G. We cannot afford to dictate the unfavourable orientation *a priori*, but rather need to figure it out as the algorithm executes. This is not difficult to get around, since we only need an arbitrary (but deterministic) orientation of each cycle. For example, we may arbitrarily order all vertices, and give a sign $\pm$ to each direction of an edge according to the ordering. The sign of an odd-length cycle is the product over all its edges, and the sign of an even-length cycle is the product over all but the least indexed edge. Then we can simply declare all orientations with a $+$ sign 'bad'. An alternative is to reject cycles randomly, which is considered in [18] and described in Section 6.

Since the extremal condition is satisfied, applying [12, Theorem 8] we get the correctness of Algorithm 1.

Proposition 2.1. *Conditioned on terminating, the output of Algorithm 1 is exactly $\pi(\,\cdot\,)$.*

We remark that bicircular popping, Algorithm 1, differs from cycle popping [25] by associating random variables to *all* vertices, and differs from cluster popping [9, 11] by associating random variables to vertices rather than edges.

3. Run-time analysis

An advantage of adopting the partial rejection sampling framework is that we have a closed-form formula for the expected run-time of these algorithms on extremal instances.

In the general setting of partial rejection sampling, the target distribution to be sampled from is a product distribution over variables, conditioned on a set of 'bad' events $(A_i)_{i\in\mathcal{I}}$ not happening for some index set $\mathcal{I}$. Let T_i be the number of resamplings of event A_i. Let q_i be the probability such that exactly A_i occurs, and let $q_\emptyset$ be the probability such that none of $(A_i)_{i\in\mathcal{I}}$ occurs, both under the product distribution. Suppose $q_\emptyset > 0$ as otherwise the support of $\pi(\,\cdot\,)$ is empty. For extremal instances, [12, Lemma 12] and the first part of the proof of [12, Theorem 13] yield

$$\mathbb{E}\, T_i = \frac{q_i}{q_\emptyset}. \tag{3.1}$$

Let T be the number of resampled variables. By linearity of expectation and (3.1),

$$\mathbb{E}\, T = \sum_{i\in\mathcal{I}} \frac{q_i \cdot |\mathsf{var}(A_i)|}{q_\emptyset}. \tag{3.2}$$

(See also [10, equation (2)].) We note that an upper bound similar to the right-hand side of (3.2) was first shown by Kolipaka and Szegedy [19] in a much more general setting but counting only the number of resampled events.

Specializing to Algorithm 1, let q_e and q_C be the corresponding quantity for bad events B_e and B_C, respectively. Let Ω_0 be the set of assignments so that no bad event happens, and let Ω_e (or Ω_C) be the set of assignments of $(a_v)_{v\in V}$ so that exactly B_e (or B_C) happens and none of the other bad events happen. Then $|\Omega_0| = |\mathcal{B}|$. For a bad event B, let $\mathsf{var}(B)$ be the set of variables defining B, namely $\mathsf{var}(B_e) = \{a_u, a_v\}$ if $e = (u, v) \in E$ and $\mathsf{var}(B_C) = \{a_v \mid v \in C\}$ if C is a cycle in G. Define

$$\Omega_E^{\mathsf{var}} := \{(\sigma, a_v) \mid \exists e \in E,\ \sigma \in \Omega_e,\ a_v \in \mathsf{var}(B_e)\}$$

and

$$\Omega_{\text{cycle}}^{\mathsf{var}} := \{(\sigma, a_v) \mid \exists \text{a cycle } C,\ \sigma \in \Omega_C,\ a_v \in \mathsf{var}(B_C)\}.$$

Then

$$\sum_{e\in E} \frac{q_e \cdot |\mathsf{var}(A_i)|}{q_\emptyset} = \frac{|\Omega_E^{\mathsf{var}}|}{|\Omega_0|} \quad \text{and} \quad \sum_{C \text{ is a cycle}} \frac{q_C \cdot |\mathsf{var}(A_i)|}{q_\emptyset} = \frac{|\Omega_{\text{cycle}}^{\mathsf{var}}|}{|\Omega_0|}.$$

Proposition 3.1. *Let T be the number of resampled variables of Algorithm 1. Then*

$$\mathbb{E}\, T = \frac{|\Omega_E^{\mathsf{var}}|}{|\Omega_0|} + \frac{|\Omega_{\text{cycle}}^{\mathsf{var}}|}{|\Omega_0|}.$$

We bound these ratios using a combinatorial encoding idea. Namely, we want to design an injective mapping from Ω_E^{var} or $\Omega_{\text{cycle}}^{\mathsf{var}}$ to Ω_0. To make the mapping injective, we in fact have to record some extra information. We first deal with $\Omega_{\text{cycle}}^{\mathsf{var}}$.

Lemma 3.2. *For a connected graph $G = (V, E)$ with $m \geqslant n$, where $m = |E|$ and $n = |V|$,*

$$|\Omega_{\text{cycle}}^{\text{var}}| \leqslant n|\Omega_0|.$$

Proof. We define a 'repairing' mapping $\varphi\colon \Omega_{\text{cycle}}^{\text{var}} \to \Omega_0 \times V$, as follows. For $\sigma \in \Omega_C$, we define σ_{fix} to be the same as σ except that the orientation of C is reversed. Clearly $\sigma_{\text{fix}} \in \Omega_0$. Let

$$\varphi(\sigma, a_v) = (\sigma_{\text{fix}}, v) \quad \text{if } \sigma \in \Omega_C \text{ and } v \in C.$$

We claim that φ is injective. To see this, given σ_{fix} and v, we simply flip the orientations of the cycle containing v to recover σ. Since φ is injective, we have that $|\Omega_{\text{cycle}}^{\text{var}}| \leqslant n|\Omega_0|$. □

For Ω_E^{var}, the proof is slightly more involved. For $\sigma \in \Omega_e$, if we contract e, this component is a directed tree rooted at e, where all edges are directed toward e.

Lemma 3.3. *For a connected graph $G = (V, E)$ with $m \geqslant n$, where $m = |E|$ and $n = |V|$,*

$$|\Omega_E^{\text{var}}| \leqslant 2n(n-1)|\Omega_0|.$$

Proof. Let $\Omega_E \coloneqq \bigcup_{e \in E} \Omega_e$. Then $|\Omega_E^{\text{var}}| = 2|\Omega_E|$.

Fix an arbitrary ordering of all vertices and edges. Our goal is to define an injective 'repairing' mapping $\varphi\colon \Omega_E \to \Omega_0 \times V \times E$. For $\sigma \in \Omega_e$, find the connected component of $U(\sigma)$ containing the edge $e = (v_1, v_2)$, and let its vertex set be S. Depending on whether $S = V$, there are two cases.

(1) If $S \neq V$, then, since the graph G is connected, there must be at least one edge joining the component to the rest of the graph. Pick the first such edge (u, u') where u is in S and u' is not.
(2) Otherwise $S = V$. Then, since the graph has at least n edges, there must be at least one edge not in $U(\sigma)$. Let $e' = (u, u')$ be the first such edge, and let C be the cycle resulting from adding e' to $U(\sigma)$. Suppose the correct orientation on C induces the orientation $u \to u'$ on e'.

Let $u = u_1, u_2, \ldots, u_\ell = v_1$ be the unique path between u and v_1 in $U(\sigma)$. (The vertex v_1 is chosen arbitrarily from the two endpoints of e.) Let σ_{fix} be the assignment so that a_{u_i} points to u_{i-1}, where $u_0 = u'$, and all other variables are unchanged from σ. It is easy to verify that $\sigma_{\text{fix}} \in \Omega_0$. Also, σ_{fix} does not depend on the choice of v_1 from the edge e. Define $\varphi(\sigma) = (\sigma_{\text{fix}}, u, e)$, where $e = (v_1, v_2)$.

We claim that φ is injective. We just need to recover σ given $(\sigma_{\text{fix}}, u, e)$. We first figure out whether $S = V$. Notice that u' can be recovered as $u \to u'$ is in σ_{fix}. If $S \neq V$, then the edge (u, u') is a bridge under σ_{fix}, whereas if $S = V$, (u, u') is not.

In the first case, simply find the path between u and v_1, and reverse the 'repairing' to yield the original σ. In the second case, we remove (u, u') first, and then recover the unique path between u and v_1. The rest is the same as the first case.

Note that $|\sigma_{\text{fix}}| = n$, $u \to u' \in \sigma_{\text{fix}}$, and $e \in U(\sigma_{\text{fix}})$, but $(v_1, v_2) \neq (u, u')$. Thus, fixing σ_{fix}, there are n choices for u, and $(n-1)$ choices for $e = (v_1, v_2)$. Since φ is injective, we have that $|\Omega_E^{\text{var}}| = 2|\Omega_E| \leqslant 2n(n-1)|\Omega_0|$. □

Combining Lemma 3.2, Lemma 3.3 and Proposition 3.1, we have the following theorem.

Theorem 3.1. *Let $G = (V, E)$ be a connected graph, $n = |V|$, $m = |E|$ and $m \geqslant n$. The expected number of random variables sampled in Algorithm 1 on G is at most $2n^2 - n$.*

Algorithm 2 A random walk implementation of bicycle popping

```
$V_u \leftarrow V$;
$S \leftarrow \emptyset$;
while $V_u \neq \emptyset$ do
    $v \leftarrow$ an arbitrary vertex in $V_u$;
    Start a random walk from $v$, where in each step we move uniformly at random to a neighbour
        of the current vertex. Erase any cycle $C$ having length 2 or a wrong orientation, until some
        vertex in $V \setminus V_u$ is reached, or a good cycle $C$ is formed;
    Remove all vertices of the walk from $V_u$;
    Add all (undirected) edges along the walk to $S$;
end
return $S$
```

The bound in Theorem 3.1 is tight. Consider a cycle of length n. Clearly $|\Omega_0| = 1$ and $|\Omega^{\mathrm{var}}_{\mathrm{cycle}}| = n$ as there is only one cycle containing n edges. Moreover, $|\Omega_e| = n - 1$ for there are $n - 1$ choices of the missing edge. Thus $|\Omega^{\mathrm{var}}_E| = 2|\Omega_E| = 2n(n - 1)$ and the upper bound is achieved.

4. An implementation based on a loop-erasing random walk

In the execution of Algorithm 1, during each iteration, one needs to find all bad events, and a naive implementation may take up to $O(n)$ time for this task, giving another factor on top of the bound in Theorem 3.1. Here we provide an implementation that has expected run-time $O(n^2)$, similar to the loop-erasing random walk of Wilson [28]. A formal description is given in Algorithm 2.

Observe that, in Algorithm 2, once a cycle is orientated correctly, none of its associated arcs will be resampled ever again, and the same holds for any arc attached to it. We will call such arcs 'fixed'. Starting from an arbitrary vertex v, we assign a random arc from v to u, and continue this for u. So far this is just the normal random walk with memory. The difference is that whenever a cycle appears, we check whether it has length > 2 and the correct orientation. If not, then we erase it, and continue the random walk. Otherwise, we keep all random arcs leading towards this cycle, and mark them as fixed. Thus Algorithm 2 amounts to a loop-erasing random walk with a special erasing rule.

Once the first random walk stops with a correctly oriented cycle, we do the same for the next vertex that has not been fixed yet. Now the new walk has two possible terminating conditions. Namely it is fixed if it has reached some fixed vertex, or a correctly oriented cycle of length > 2 is formed. This process is repeated until all vertices are fixed.

Algorithm 1 specifies a particular order of resampling bad events, modulo the ordering of bad events within each iteration of the while-loop. However, bad events can be sampled in any order, without affecting correctness or the expected number of resampled variables. Although the proof of this key fact has appeared in the context of specific instances of partial rejection sampling, such as cycle popping [25] and sink popping [3], we are not aware that the argument has been presented in generality, so we do so presently. As a consequence of this key fact, Algorithm 2, which is sequential, has the same resulting distribution and expected number of resampled variables as Algorithm 1, which is parallel. In particular, the expected run-time of Algorithm 2 has the same order as the number of resampled variables, which is at most $O(n^2)$ by Theorem 3.1.

The correctness of Algorithm 2 is due to the aforementioned fact that the ordering of resamplings does not matter for extremal instances. We now formalize and verify this fact. Consider a generic partial rejection sampling algorithm that repeatedly locates an occurring bad event and resamples the variables on which it depends. A specific implementation will choose a particular order for resampling the bad events. We can represent the choices made as a path in a countably infinite, directed 'game graph' $\Gamma = (\Sigma, A)$. The vertex set Σ of Γ contains all multisets of bad events. We refer to these vertices as *states*. The arc set A is defined relative to a *resampling table*, as used in [12], following Moser and Tardos. As the algorithm proceeds, the 'frontier' in the table between used and fresh random variables advances; in the notation of [12], the frontier at time t is specified by the indices $(j_{i,t}: 1 \leqslant i \leqslant n)$. At time t, the implementation will have sampled a certain multiset $M \in \Sigma$ of bad events: an event B_* that has been resampled k times will occur k times in M. Note that M determines the number of times each variable has been resampled, and hence the frontier of the table. So, even though we do not know the order in which those bad events were resampled, we do know the occurring bad events at time t. For each $M \in \Sigma$ and each possible occurring bad event B_*, we add an arc in Γ from M to $M' = M + B_*$. A state with outdegree 0 is a *terminating state*. Given a fixed resampling table, an implementation of partial rejection sampling will generate a directed path in Γ starting at the state $\emptyset$. With probability 1 (over the choice of resampling table), this path will be finite, that is, end in a terminating state.

We now apply a lemma of Eriksson [5], which is similar in spirit to Newman's lemma, but which is both more elementary and better suited to our needs. Observe that if two bad events occur at time t then they can be resampled in either order without altering the result; this is a consequence of the fact that the events are on disjoint sets of variables. In the terminology of Eriksson [5], the game graph Γ has the *polygon property*. It follows from his Theorem 2.1 that Γ has the *strong convergence property*: if there exists a path starting at $\emptyset$ and terminating at M, then every path starting at $\emptyset$ will terminate at M in the same number of steps. Since a terminating path exists with probability 1, we see that both the output and the number of resampled variables is independent of the order in which the implementation decides to resample bad events. In other words, the correctness of Algorithm 2 follows from that of Algorithm 1, and the distribution of the number of resampled variables is identical in the two algorithms.

5. Approximating the number of bases

For completeness, we include a standard self-reduction to count the number of bases of a bicircular matroid, utilizing Algorithm 1.

Theorem 5.1. *There is an FPRAS for counting bases of a bicircular matroid, with time complexity* $O(n^3 m^2 \varepsilon^{-2})$.

Proof. Let $0 < \varepsilon < 1$ be a parameter expressing the desired accuracy. Also, let $N(G)$ be the number of bases of $B(G)$, the bicircular matroid associated with G.

The technique for reducing approximate counting to sampling is entirely standard [14, Chapter 3], but we include the details here for completeness. Fix any sequence of graphs $G = G_m, G_{m-1}, \dots, G_{n+1}, G_n$, where each graph G_{i-1} is obtained from the previous one G_i by removing a single edge e_i, and G_n is a disjoint union of unicyclic components. (Thus the edge set of G_n is a basis of $B(G)$.) Then, noting $N(G_n) = 1$,

$$N(G)^{-1} = N(G_m)^{-1} = \frac{N(G_{m-1})}{N(G_m)} \times \frac{N(G_{m-2})}{N(G_{m-1})} \times \cdots \times \frac{N(G_{n+1})}{N(G_{n+2})} \times \frac{N(G_n)}{N(G_{n+1})}. \tag{5.1}$$

Let X_i be the random variable resulting from the following trial: select, uniformly at random, a basis R from $B(G_i)$ and set

$$X_i = \begin{cases} 1 & \text{if } e_i \notin R, \\ 0 & \text{otherwise.} \end{cases}$$

Here we use Algorithm 2 to generate the uniform random basis R. For different i, we use fresh random sources so that all X_i are mutually independent. Note that $\mu_i = \mathbb{E}\, X_i = N(G_{i-1})/N(G_i)$, so that

$$N(G)^{-1} = \mathbb{E}\,(X_m X_{m-1} \ldots X_{n+2} X_{n+1}) = \mu_m \mu_{m-1} \ldots \mu_{n+2}\mu_{n+1}.$$

Now let $\overline{X}_i$ be obtained by taking the mean of t independent copies of the random variable X_i. Since $\mathbb{E}\,\overline{X}_i = \mu_i$, we have $N(G)^{-1} = \mathbb{E}\,Z$, where $Z = \overline{X}_m \overline{X}_{m-2} \cdots \overline{X}_{n+1}$. Also, $\operatorname{Var} \overline{X}_i = t^{-1} \operatorname{Var} X_i$, so if t is large enough the variance of Z will be small, and Z^{-1} will be a good estimate for $N(G)$. For this approach to yield a polynomial-time algorithm, we need that all the fractions appearing in the product (5.1) are not too small. In fact we will show that they are all bounded below by $1/2n$, which is sufficient.

For the moment, assume this claim, that is, $1/2n \leqslant \mu_i \leqslant 1$, for all $n < i \leqslant m$. Note that $\mathbb{E}\,X_i^2 = \mathbb{E}\,X_i = \mu_i$ since X_i is a 0,1-variable. Standard manipulations give

$$\mathbb{E}\,\overline{X}_i^2 = \operatorname{Var} \overline{X}_i + (\,\mathbb{E}\,\overline{X}_i)^2 = t^{-1} \operatorname{Var} X_i + \mu_i^2 = t^{-1}(\,\mathbb{E}\,X_i^2 - \mu_i^2) + \mu_i^2 \leqslant \mu_i^2\left(1 + \frac{1}{t\mu_i}\right),$$

whence

$$\mathbb{E}\,Z^2 = \mathbb{E}\,\overline{X}_m^2 \ldots \mathbb{E}\,\overline{X}_{n+1}^2 \leqslant \mu_m^2 \ldots \mu_{n+1}^2 \left(1 + \frac{2n}{t}\right)^m = (\,\mathbb{E}\,Z)^2\left(1 + \frac{2n}{t}\right)^m.$$

Setting $t = 40nm\varepsilon^{-2}$, we obtain

$$\mathbb{E}\,Z^2 = \exp\,(\varepsilon^2/20)(\,\mathbb{E}\,Z)^2 \leqslant (1 + \varepsilon^2/16)(\,\mathbb{E}\,Z)^2,$$

which implies $\operatorname{Var} Z \leqslant (\varepsilon^2/16)(\,\mathbb{E}\,Z)^2$. Thus, by Chebyshev's inequality,

$$\mathbb{P}\left[|Z - N(G)^{-1}| \leqslant \frac{1}{2}\varepsilon N(G)^{-1}\right] = \mathbb{P}\left[|Z - \mathbb{E}\,Z| \leqslant \frac{1}{2}\varepsilon\,\mathbb{E}\,Z\right] \geqslant \frac{3}{4}.$$

It follows that

$$\mathbb{P}\,[|Z^{-1} - N(G)| \leqslant \varepsilon N(G)] \geqslant \frac{3}{4}.$$

In other words, the algorithm that returns the estimate Z^{-1} satisfies the conditions for an FPRAS.

To complete the proof we just need to bound the ratio $\mu_i = N(G_{i-1})/N(G_i)$. Let R be a basis of $B(G_i)$ that contains the edge e_i, *i.e.* that is not a basis of $B(G_{i-1})$. Let R_0 be the unique basis in $B(G_n)$ and note that $e_i \notin R_0$. Since R_0 is also a basis of $B(G_i)$, the exchange axiom for matroids asserts that there is an edge $f \in R_0 \setminus R$ such that $R + f - e_i$ is a basis of $B(G_i)$ and hence of $B(G_{i-1})$. This exchange operation associates a basis in $B(G_{i-1})$ with each basis in $B(G_i)$ that is not a basis in $B(G_{i-1})$; furthermore, every basis in $B(G_{i-1})$ arises at most $|R_0| = n$ times in this way. It follows that $N(G_i) \leqslant (n+1)N(G_{i-1}) \leqslant 2nN(G_{i-1})$, as required.

Overall we need $O(nm\varepsilon^{-2})$ samples for $m - n$ estimators each. For each sample we use Algorithm 2, which has expected run-time $O(n^2)$ by Theorem 3.1, yielding the claimed time complexity. □

6. Faster approximate counting

Similar to [18], let Ω be the set of configurations consisting of directed edges so that every vertex is the tail of exactly one arc. Consider the following Gibbs distribution:

$$\rho_{\gamma_2,\gamma}(S) \propto \gamma_2^{C_2(S)}\gamma^{C(S)}, \tag{6.1}$$

where $S \in \Omega$, $\gamma_2, \gamma \geqslant 0$ are two parameters, and $C(S)$ (or $C_2(S)$) is the number of cycles of length greater than 2 (or 2-cycles) present in (V, S). We adopt the convention that $0^0 = 1$.

Algorithm 3 Sample $\rho_{\gamma_2,\gamma}$ in (6.1)

```
$V_u \leftarrow V$;
$S \leftarrow \emptyset$;
while $V_u \neq \emptyset$ do
    $v \leftarrow$ an arbitrary vertex in $V_u$;
    Start a random walk from $v$, and erase any cycle $C$ formed with probability $1-\gamma_2$ if the length
      is 2 or with probability $1-\gamma$ otherwise, until some vertex in $V \setminus V_u$ is reached, or a cycle $C$
      is accepted;
    Remove all vertices of the walk from $V_u$;
    Add all arcs along the walk to $S$;
end
return $S$
```

Then Algorithms 1 and 2 sample from the distribution $\mu_{0,0.5}$. We also define the corresponding partition function

$$Z_{\gamma_2,\gamma}(G) = \sum_{S \in \Omega} \gamma_2^{C_2(S)} \gamma^{C(S)}. \tag{6.2}$$

Then $Z_{0,0.5}(G) = |\mathcal{B}|$. Another interesting special case is $\rho_{1,1}$, with the corresponding partition function $Z_{1,1}(G) = \prod_{v \in V} \deg(v)$. This is because $\rho_{1,1}$ corresponds to choosing a random neighbour of v for each $v \in V$. Note that each cycle longer than 2 has two potential orientations.

In order to sample from $\rho_{\gamma_2,\gamma}$ in (6.1) where $\gamma_2, \gamma \in [0,1]$, we introduce the following variant of Algorithm 2. Again, this is very similar to the sampling algorithm of Kassel and Kenyon [18].

The correctness of Algorithm 3 also follows from [12, Theorem 8] and the argument in Section 4. We introduce an auxiliary variable for each cycle, which is false with probability $1-\gamma_2$ if the cycle has length 2, or with probability $1-\gamma$ if the cycle is longer. A cycle is 'bad' if and only if it is present and the auxiliary variable is false. Although there are exponentially many such auxiliary variables, we only reveal them when necessary. Every time a cycle is popped, the auxiliary variable is reset. By the same reason as for Algorithm 1, such an instance is *extremal* and the correctness follows.

Since the extremal condition holds, the run-time of Algorithm 3 can be analysed analogously to that of Algorithms 1 and 2. Let T be the number of resampled variables of Algorithm 3. We apply (3.2). First consider the case of $\gamma_2 = 0$. To bound $\sum_{e \in E} q_e |e| / q_\emptyset$, we use the injective mapping in Lemma 3.3, and to bound $\sum_{C \text{ is a cycle}} q_C |C| / q_\emptyset$, we use an injective mapping similar to the one in Lemma 3.2 by simply flipping the auxiliary variable. Observe that both mappings preserve all cycles other than the one repaired, and as a consequence, preserve all weights up to a factor $(1-\gamma)/\gamma$ in the latter case. It implies that when $\gamma_2 = 0$, the run-time can be bounded as follows:

$$\mathbb{E}\, T \leqslant 2n(n-1) + \frac{1-\gamma}{\gamma} \cdot n. \tag{6.3}$$

Otherwise $\gamma_2 > 0$. To bound $\sum_{e \in E} q_e |e| / q_\emptyset$, again, we need to resort to the injective mapping in Lemma 3.3. However, now the 'perfect' configurations and 'one-flaw' configurations allow more than one 2-cycles. Let $\Omega^{(k)} \subset \Omega$ be the set of configurations with k 2-cycles, for any $k \leqslant n/2$.

Lemma 6.1. *There is an injective mapping $\psi_k \colon \Omega^{(k)} \to \Psi^{(k-1)}$ for any $k \geqslant 1$, where*

$$\Psi^{(k-1)} := \{(\sigma, v, e) \mid \sigma \in \Omega^{(k-1)},\ v \in V,\ e \in U(\sigma)\}.$$

Moreover, ψ_k preserves all cycles except one with length 2.

Proof. The mapping is similar to the one in Lemma 3.3. Given $S \in \Omega^{(k)}$, we choose an arbitrary 2-cycle, and apply the 'fix' of the injective mapping in Lemma 3.3. It is straightforward to check that this operation will only destroy the chosen 2-cycle, and it is reversible given the auxiliary information. □

A consequence of Lemma 6.1 is that $\sum_{e \in E} q_e|e|/q_\emptyset \leqslant 2n^2$, similar to Lemma 3.3. This is because for any configuration with exactly one 'bad' 2-cycle, applying the mapping in Lemma 6.1 yields a configuration without the presence of the bad 2-cycle, and other cycle structures are all preserved. The overhead is to remember one vertex and one edge from the undirected version of the image.

To bound $\sum_{C \text{ is a cycle}} q_C|C|/q_\emptyset$, consider again the injective mapping similar to the one in Lemma 3.2, by simply flipping the value of the auxiliary variable. It implies that

$$\sum_{C \text{ is a cycle}} \frac{q_C|C|}{q_\emptyset} \leqslant \frac{1-\gamma}{\gamma} \cdot n.$$

Thus we have that for $\gamma_2 > 0$, the following also holds:

$$\mathbb{E}\, T \leqslant 2n^2 + \frac{1-\gamma}{\gamma} \cdot n. \tag{6.4}$$

Combining the two cases (6.3) and (6.4), we have the following corollary.

Corollary 6.2. *The run-time of Algorithm 3, in expectation, is $O(n^2)$ if $\gamma_2 \in [0, 1]$ and $\gamma \in [1/2, 1]$.*

The Gibbs formulation (6.1) allows us to utilize faster annealing algorithms to reduce approximate counting to sampling. See [27] and [13]; the current best algorithm is due to Kolmogorov [20].

In fact, we will need a slight generalization from [10] as follows. Let Ω be a finite set, and the generalized Gibbs distribution $\rho_\beta(\cdot)$ over Ω takes the following form:

$$\rho_\beta(X) = \frac{1}{Z(\beta)} \exp(-\beta H(X)) \cdot F(X), \tag{6.5}$$

where β is the *temperature*, $H(X) \geqslant 0$ is an integer function called the *Hamiltonian*, $F\colon \Omega \to \mathbb{R}^+$ is a non-negative function and, with a little abuse of notation,

$$Z(\beta) = \sum_{X \in \Omega} \exp(-\beta H(X)) \cdot F(X)$$

is the normalizing factor. We would like to turn the sampling algorithm into an approximation algorithm to $Z(\beta)$. Typically this involves calling the sampling oracle in a range of temperatures, which we denote by $[\beta_{\min}, \beta_{\max}]$. (This process is usually called simulated annealing.) Let

$$Q := \frac{Z(\beta_{\min})}{Z(\beta_{\max})}, \quad q = \log Q \quad \text{and} \quad N = \max_{X \in \Omega} H(X).$$

The following result is due to Kolmogorov [20, Theorem 8], as extended in [10, Lemma 8].

Proposition 6.3. *Suppose we have a sampling oracle from the distribution ρ_β for any $\beta \in [\beta_{\min}, \beta_{\max}]$. There is an algorithm to approximate Q within $1 \pm \varepsilon$ multiplicative errors using $O(q \log N/\varepsilon^2)$ oracle calls on average.*

Theorem 6.1. *There is an FPRAS for counting bases of a bicircular matroid, with time complexity $O(n^3(\log n)^2\varepsilon^{-2})$.*

Proof. We will do a two-stage annealing. Our starting point is $Z_{1,1}(G) = \prod_{v \in V} \deg(v)$. We first apply the annealing algorithm in Proposition 6.3 between $\gamma_2 = 1$ and $\gamma_2 = 0$. We only treat γ_2 as the temperature in this stage but not γ.[3] Clearly $H(S) = C_2(S) \leqslant m$ in this case, and $\log N = O(\log n)$. The more complicated estimate is

$$Q = \frac{Z_{1,1}(G)}{Z_{0,1}(G)}.$$

For any $S \in \Omega$, we apply the mapping in Lemma 6.1 at most $C_2(S) \leqslant n$ times to get S' where no 2-cycle is present. Given S', and a sequence of vertex-edge pair generated by this repairing sequence, we may uniquely recover S since the mapping in Lemma 6.1 is injective. Moreover, the longer cycles in S are all preserved in S'. It implies that

$$Q = \frac{Z_{1,1}(G)}{Z_{0,1}(G)} \leqslant (mn)^n \quad \text{and} \quad q = \log Q = O(n \log n).$$

Hence the number of samples required in this step is $O(n(\log n)^2 \varepsilon^{-2})$.

In the second stage, we apply Proposition 6.3 between $\gamma = 1$ and $\gamma = 0.5$, while $\gamma_2 = 0$ is fixed. Then $H(S) = C(S) \leqslant n$ in this case, and $\log N = O(\log n)$. Moreover

$$Q = \frac{Z_{0,1}(G)}{Z_{0,0.5}(G)} \leqslant 2^{\max_S C(S)} \leqslant 2^n.$$

Thus $q = \log Q = O(n)$. The number of samples required in this step is $O(n \log n \varepsilon^{-2})$.

Overall, the number of samples required is $O(n(\log n)^2 \varepsilon^{-2})$. Together with Corollary 6.2, this implies the claimed run-time. □

Acknowledgements

We thank Adrien Kassel for bringing reference [18] to our attention. We also thank Kun He for pointing out an improvement in Lemma 3.3. Part of the work was done while HG was visiting the Institute of Theoretical Computer Science, Shanghai University of Finance and Economics, and he would like to thank their hospitality.

References

[1] Anari, N., Liu, K., Oveis Gharan, S. and Vinzant, C. (2019) Log-concave polynomials II: high-dimensional walks and an FPRAS for counting bases of a matroid. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pp. 1–12. ACM.
[2] Chávez Lomelí, L. and Welsh, D. J. A. (1996) Randomized approximation of the number of bases. In *Matroid Theory (Seattle, WA, 1995)*, Vol. 197 of Contemporary Mathematics, pp. 371–376. AMS.
[3] Cohn, H., Pemantle, R. and Propp, J. G. (2002) Generating a random sink-free orientation in quadratic time. *Electron. J. Combin.* **9** #R10.
[4] Cryan, M., Guo, H. and Mousa, G. (2019) Modified log-Sobolev inequalities for strongly log-concave distributions. In *60th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2019)*, pp. 1358–1370. IEEE.
[5] Eriksson, K. (1996) Strong convergence and a game of numbers. *European J. Combin.* **17** 379–390.
[6] Feder, T. and Mihail, M. (1992) Balanced matroids. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC '92)*, pp. 26–38. ACM.
[7] Giménez, O., de Mier, A. and Noy, M. (2005) On the number of bases of bicircular matroids. *Ann. Combin.* **9** 35–45.
[8] Giménez, O. and Noy, M. (2006) On the complexity of computing the Tutte polynomial of bicircular matroids. *Combin. Probab. Comput.* **15** 385–395.
[9] Gorodezky, I. and Pak, I. (2014) Generalized loop-erased random walks and approximate reachability. *Random Struct. Algorithms* **44** 201–223.
[10] Guo, H. and He, K. (2018) Tight bounds for popping algorithms. *Random Struct. Algorithms*. doi:10.1002/rsa.20928.

[3] This explains why we need $F(X)$ in (6.5).

[11] Guo, H. and Jerrum, M. (2019) A polynomial-time approximation algorithm for all-terminal network reliability. *SIAM J. Comput.* **48** 964–978.
[12] Guo, H., Jerrum, M. and Liu, J. (2019) Uniform sampling through the Lovász local lemma. *J. Assoc. Comput. Mach.* **66** 18.
[13] Huber, M. (2015) Approximation algorithms for the normalizing constant of Gibbs distributions. *Ann. Appl. Probab.* **25** 974–985.
[14] Jerrum, M. (2003) *Counting, Sampling and Integrating: Algorithms and Complexity*, Lectures in Mathematics ETH Zürich. Birkhäuser.
[15] Jerrum, M. (2006) Two remarks concerning balanced matroids. *Combinatorica* **26** 733–742.
[16] Jerrum, M., Son, J.-B., Tetali, P. and Vigoda, E. (2004) Elementary bounds on Poincaré and log-Sobolev constants for decomposable Markov chains. *Ann. Appl. Probab.* **14** 1741–1765.
[17] Kassel, A. (2015) Learning about critical phenomena from scribbles and sandpiles. In *Modélisation Aléatoire et Statistique: Journées MAS 2014*, Vol. 51 of ESAIM Proc. Surveys, pp. 60–73. EDP Sciences, Les Ulis.
[18] Kassel, A. and Kenyon, R. (2017) Random curves on surfaces induced from the Laplacian determinant. *Ann. Probab.* **45** 932–964.
[19] Kolipaka, K. B. R. and Szegedy, M. (2011) Moser and Tardos meet Lovász. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC 2011)*, pp. 235–244. ACM.
[20] Kolmogorov, V. (2018) A faster approximation algorithm for the Gibbs partition function. In *Proceedings of the 31st Conference On Learning Theory (COLT)*, Vol. 75 of Proceedings of Machine Learning Research, pp. 228–249. PMLR.
[21] Matthews, L. R. (1977) Bicircular matroids. *Quart. J. Math. Oxford Ser. (2)* **28** 213–227.
[22] Maurer, S. B. (1976) Matrix generalizations of some theorems on trees, cycles and cocycles in graphs. *SIAM J. Appl. Math.* **30** 143–148.
[23] Motwani, R. and Raghavan, P. (1995) *Randomized Algorithms*. Cambridge University Press.
[24] Piff, M. J. and Welsh, D. J. A. (1971) The number of combinatorial geometries. *Bull. London Math. Soc.* **3** 55–56.
[25] Propp, J. G. and Wilson, D. B. (1998) How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *J. Algorithms* **27** 170–217.
[26] Snook, M. (2012) Counting bases of representable matroids. *Electron. J. Combin.* **19** P41.
[27] Štefankovič, D., Vempala, S. and Vigoda, E. (2009) Adaptive simulated annealing: a near-optimal connection between sampling and counting. *J. Assoc. Comput. Mach.* **56** 18.
[28] Wilson, D. B. (1996) Generating random spanning trees more quickly than the cover time. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC 1996)*, pp. 296–303. ACM.

Cite this article: Guo H and Jerrum M (2021). Approximately counting bases of bicircular matroids. *Combinatorics, Probability and Computing* **30**, 124–135. https://doi.org/10.1017/S0963548320000292