A Sublinear Local Access Implementation for the Chinese Restaurant Process

Peter Mörters
University of Cologne

Christian Sohler
University of Cologne

Stefan Walzer
University of Cologne

Abstract

The Chinese restaurant process is a stochastic process closely related to the Dirichlet process that groups sequentially arriving objects into a variable number of classes, such that within each class objects are cyclically ordered. A popular description involves a restaurant, where customers arrive one by one and either sit down next to a randomly chosen customer at one of the existing tables or open a new table. The full state of the process after n steps is given by a permutation of the n objects and cannot be represented in sublinear space. In particular, if we only need specific information about a few objects or classes it would be preferable to obtain the answers without simulating the process completely.

A recent line of research [15, 28, 5, 12] attempts to provide access to huge random objects without fully instantiating them. Such *local access implementations* provide answers to a sequence of queries about the random object, following the same distribution as if the object was fully generated. In this paper, we provide a local access implementation for a generalization of the Chinese restaurant process described above. Our implementation can be used to answer any sequence of adaptive queries about class affiliation of objects, number and sizes of classes at any time, position of elements within a class, or founding time of a class. The running time per query is polylogarithmic in the total size of the object, with high probability. Our approach relies on some ideas from the recent local access implementation for preferential attachment trees by Even et al. [12]. Such trees are related to the Chinese restaurant process in the sense that both involve a "rich-get-richer" phenomenon. A novel ingredient in our implementation is to embed the process in continuous time, in which the evolution of the different classes becomes stochastically independent [21]. This independence is used to keep the probabilistic structure manageable even if many queries have already been answered. As similar embeddings are available for a wide range of urn processes [2], we believe that our approach may be applicable more generally. Moreover, local access implementations for birth and death processes that we encounter along the way may be of independent interest.

2012 ACM Subject Classification Theory of computation → Generating random combinatorial structures; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Chinese restaurant process, Dirichlet process, sublinear time algorithm, random recursive tree, random permutation, random partition, Ewens distribution, simulation, local access implementation, continuous time embedding.

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2022.28

Category RANDOM

Funding *Stefan Walzer*: DFG grant WA 5025/1-1.

1 Introduction

Random objects are often used to model how data is generated. Examples include Gaussian mixture models, random graph models such as the preferential attachment model [3], Erdős-Renyi graphs [11] and the stochastic block model [19]. Usually, the process to generate such an object is fairly easy to describe and implement. However, if we think of these objects as modelling very large data sets it may be time and space consuming to generate an instance of the model. In particular, if we want to evaluate a sampling based algorithm on such a model or we want to study some local properties of the generated object, it would be interesting to provide local access to the object without fully instantiating it immediately. For example, can we easily determine the neighbourhood of a vertex in a webgraph taken from the preferential attachment model without first fully computing the graph? Can we compute the nearest neighbour of a point in a Gaussian mixture model without generating all data points? If we were able to provide consistent and efficient answers to such queries, we could, for example, run sampling algorithms on very large input graphs without fully generating them. Of course, such a local access must provide to all sequences of queries and answers the same distribution as if the object was immediately fully instantiated.

The challenge is that answers to queries must be correlated in the right way, i.e. the distribution of the answer for a query must be a conditional distribution that takes into account all answers given to previous queries. In a Gaussian mixture model, for instance, revealing the location x of one point makes it more likely that additional points are located close to x.

The Chinese restaurant process. An intuitive description of the *Chinese restaurant process* (CRP) as generalised in [6] involves a restaurant with round tables of unbounded capacity, corresponding to classes of objects, and customers, corresponding to the objects. We imagine that every dish has an objective tastiness and a distribution Φ on $(0, \infty)$ captures how tasty a random dish from the menu is. In every round one customer arrives, and the n-th customer has n options. She may sit down next to one of the $n-1$ earlier customers and order the same dish as him, or she sits at a new table and orders a random dish from the menu. She makes each choice with a probability proportional to how appealing it is. The appeal of sitting next to customer c is the tastiness of the dish of c and the appeal of sitting at a new table is 1. Applications in biology motivate us to speak of *fitness* rather than tastiness in the following. Instead of being assigned to dishes, we may instead assume that fitness values are assigned to the customers themselves or simply to the tables, as all customers at a table always order the same dish.

The CRP is closely related to Dirichlet processes and the Pólya urn model. Its easy iterative definition has contributed to its popularity as a model for clustering in Bayesian statistics, for example for gene expression data [27, 29] or in image analysis [25]. The random partition induced by the CRP is the Ewens distribution that describes the allelic partition of DNA in the infinite alleles models under assumptions of neutrality and no recombination [10, 8].

Our Contribution. In this paper we provide a local access implementation of the N-round CRP and several related processes. The number of steps required to compute the answer to a query is polylogarithmic in N. An informal version of our main theorem is as follows.

▶ **Theorem 1** (Main, informal). *Let Φ be a distribution on $(0, \infty)$ with some means of sampling from Φ, and let $N \in \mathbb{N}$. A local access implementation of the N-round CRP with parameter Φ can be achieved such that any (possibly adaptive) sequence of queries from the list below can be answered in* $\mathrm{polylog}(N)$ *time per query whp.*

- *Which customer sits right/left of customer c after $n \leq N$ rounds?*
- *What is the fitness of customer c's table?*
- *Who founded customer c's table?*
- *How many customers are at each of the tables after $n \leq N$ rounds?*

A formal version of this theorem can be found in Section 2.2.

1.1 Related Work

The problem of providing local access to a random object has appeared in several earlier works. Goldreich, Goldwasser and Nussboim [15] define a framework to study implementations of random objects. They assume the random object can be described as a function and require that queries to the function are answered in polylogarithmic time. They discuss *perfect implementations* (which have the same distribution as the original object), *close implementations* (which have approximately the same distribution) and *pseudoimplementations* (which cannot be efficiently distinguished from the original object). They also consider the truthfulness of implementations meaning that if every random object has property T then also every object in the implementation is required to have property T. They give several examples of implementations and further examples are found in [28]. Perfect implementations are also known as local access implementations or local access generators. These have been developed for preferential attachment trees by Even et al. [12], for Erdős-Renyi graphs and Dyck paths by Biswas et al. [5], and for random walks by Biswas et al. [4].

A related line of research includes partitioning oracles [17, 23, 22] that provide sublinear time access to a random partition of an input graph. While the partition is also a random object, randomness is used to guarantee that the partition cuts only few edges and that queries can be answered efficiently.

Yet another related direction is local computation algorithms developed by Rubinfeld et al. [30] and Alon et al. [1] that give sublinear time local access to the solution of a computational task. Examples for problems for which local computation algorithms are known include sparse spanning graphs [24], set cover [16], mechanism design [18], clustering [13] and maximum matching [26].

1.2 Technical Overview

In our pursuit of a local access implementation of the CRP we develop local access implementations of several related processes that may be of independent interest. An overview is given in Figure 1. Two aspects of the CRP are captured separately. In the following, we outline some difficulties in dealing with them as well as how we overcome these difficulties.

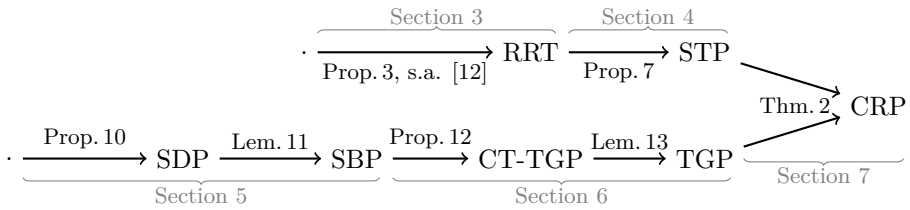

Figure 1 Random processes that capture aspects of the CRP. The picture indicates the theorem, proposition or lemma that establishes a local access implementation of these processes, the sections where they are found, as well as what each result builds upon.

The Single Table Process (STP). The simpler aspect concerns the order of customers at a single table, i.e. we have to answer who sits left or right of whom. Let us ignore other tables for now and simply assume customers $[n] := \{1, \dots, n\}$ join a table one by one, and each $i \in \{2, \dots, n\}$ sits down to the right of a customer sampled uniformly at random from $[i-1]$.

The final ordering of the customers is given by a uniformly random cyclic permutation $\pi : [n] \to [n]$, with $\pi(c)$ being c's final right neighbour. A local access implementation of π would be easy to come by but is insufficient for our purposes. This is because we permit queries regarding earlier points in time, i.e. a query may request the right neighbour of customer c after $n' \in \{c, \dots, n\}$ steps of the process. Within the sequence $(\pi(c), \pi^2(c), \pi^3(c), \dots)$ of customers towards the right of c in the final ordering, the query asks for the first element c' with $c' \leq n'$, i.e. the first customer who was already present at time n'. Doing this naively by generating the sequence may be too costly.

To allow computing c' quickly, we consider the **random recursive tree** (RRT), which is a rooted tree T with vertex set $[n]$ generated as customers arrive. We begin with just customer 1 as a root vertex. When customer $c \geq 2$ arrives and takes her seat to the right of customer p, then c is prepended to the list of children of vertex p in T. As it turns out, a depth first search of T then visits the vertices in the order $(1, \pi(1), \dots, \pi^{n-1}(1))$, reflecting the final ordering of customers. Since T has logarithmic depth and logarithmic maximum degree whp we can determine the predecessor and successor of a given vertex in the DFS ordering by issuing a logarithmic number of neighbourhood queries to T whp. Moreover – and crucially – information about the order of customers at any time step $n' < n$ is naturally contained in T within the subtree T' induced by vertex set $[n']$. In other words, to find the right neighbour of a vertex c at time n' we find its DFS successor in T'.

We therefore have to implement local access to a random recursive tree T. This has already been achieved by Even et al [12]. We simplify their implementation in two ways. Firstly, we always reveal the neighbourhood of any requested vertex in its entirety instead of revealing only the "next child" in its adjacency list. This simplifies the structure of the residual probability space without negatively impacting running time. Secondly, we employ what we call the "harmonic sampling trick" which allows sampling a set $X \subseteq [N]$ in $\mathcal{O}(|X|)$ steps that contains each $i \in [N]$ independently with probability $1/i$. The algorithm is quite simple: Sample x uniformly from $[N]$, then recursively sample a set $X' \subseteq [x-1]$ that contains each $i \in [x-1]$ independently with probability $1/i$, and return $X = \{x\} \cup X'$. The intuition why the trick is useful is that vertex 1 in T has vertices $2, 3, 4, \dots$ as children with probability $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \dots$, respectively. When combined with rejection sampling, the trick remains useful also when vertices other than 1 are concerned and when partial information about T has been revealed.

The Table Growth Process (TGP). The more difficult aspect concerns the number of customers at each of the tables over time. The growth of a single table T is correlated with the other tables. This is true for the obvious reason that after n steps all table sizes add to n, but also because the creation of a new table with high fitness can significantly hamper the ability of other tables to attract customers in the future. Even in the absence of table creations and with identical fitness values a considerable challenge remains: Assume we have revealed that at times n_1 and n_2 with $n_1 < n_2$ there are ℓ tables with fitness value 1 each and that the numbers of customers at these tables has risen from $a_1, a_2, \dots, a_\ell$ to $a_1+d_1, a_2+d_2, \dots, a_\ell+d_\ell$. Assume further that the customer counts $a_1 + d'_1, a_2 + d'_2, \dots, a_\ell + d'_\ell$ at an intermediate time $n \in \{n_1, \dots, n_2\}$ is requested. Then $(d'_1, \dots, d'_\ell)$ has a *multivariate hypergeometric distribution*, i.e. d'_i is the number of balls of colour i when drawing $n - n_1$ balls from an

urn without replacement where the urn contains d_j balls of colour j for each $j \in [\ell]$. The intimidating prospect of having to efficiently sample from such a distribution prompted us to choose a different path. While we have since been made aware that fast approximate sampling from multivariate hypergeometric distributions has been achieved in [5, Appendix D], we still do not know how table creations and fitness values could be incorporated into such an approach.

We dodge these complications by adopting a continuous time view of the table growth process where each table gains customers *independently* of all other tables. More precisely, every table gains additional customers with a *rate* proportional to both its fitness and size, and new tables are founded with a rate of 1. Properties of the exponential distribution ensure that the next customer to arrive will always be destined for a specific (new or old) table with a probability proportional to that table's rate, as required. A complication is that the times at which customers arrive are now *random*. In particular, if we are interested in the state of the process after n customers have arrived, we first have to locate a corresponding point in time using binary search in the time dimension.

In our continuous time table growth process (CT-TGP), the growth of a single table with fitness 1 is governed by a **simple birth process** (SBP, also called Yule process) that begins with a single element and from then on gains elements with a rate equal to its size. A fitness $\neq 1$ merely amounts to rescaling time. A SBP can be seen as a **simple death process** (SDP) played in reverse, where a SDP starts with some number N of elements and each element dies independently of the others with a rate of 1, i.e. it has an Exp(1)-distributed lifetime.

Our most fundamental problem is therefore that of providing a local access implementation to the SDP, which can answer for a given time $t \in \mathbb{R}_{\geq 0}$ how many elements die until time t. The rough idea for dealing with the first query is as follows. Let m be the median of the exponential distribution. The number of elements dying at a time in $[0, m]$ has distribution $N' \sim \mathrm{Bin}(N, \frac{1}{2})$ and if $t < m$ we need only obtain further information on those N' elements to answer the query. In that case, let $0 < m' < m$ be the median of the exponential distribution when conditioned on attaining values in $[0, m]$. The number N'' of elements dying within $[0, m']$ has distribution $N'' \sim \mathrm{Bin}(N', \frac{1}{2})$ and depending on whether $t \leq m'$ or $t > m'$ we would have to continue worrying about the precise death times of only N'' or $N' - N''$ elements – roughly half in expectation. To fully answer the query at hand and further queries like it we lazily construct a binary tree where inner nodes record numbers of elements (such as N' and N'') with death times falling within respective ranges. The elements themselves are represented in leafs which are at depth $\mathcal{O}(\log N)$ whp. To sample the necessary Binomial random variables in $\mathcal{O}(\log N)$ time whp we use a result by [7].

2 Preliminaries and formal statement of main result

2.1 The Chinese restaurant process with table fitness

The variant of the *Chinese restaurant process* considered in this paper was proposed in [6] and is parametrised by a distribution Φ on $(0, \infty)$. An outcome of the CRP is given by a (random) mapping

$$S \colon \mathbb{N} \to \mathbb{N} \text{ such that } S(n) \in [n]$$

where $n := \{1, \dots n\}$. It can be interpreted as follows. In a Chinese restaurant an unbounded number of circular tables are initially empty and customers arrive one by one. If $S(i) < i$ then the i-th customer takes her place at an existing table directly to the right of customer $S(i)$.

i	1	2	3	4	5	6	7
$S(i)$	1	2	2	1	1	6	4

c	1	2	3	4	5	6	7
$\pi_7(c)$	5	3	2	7	4	6	1

$\pi_7 = (1\,5\,4\,7)(2\,3)(6)$

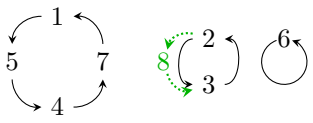

Figure 2 Some values for S, the permutation π_7 they give rise to, the cycle representation of π_7 and a visualisation. In the interpretation of the CRP the elements 1, 2 and 6 are founders and thereby representatives of three tables. In green we show the effect on the second table if $S(8) = 2$.

If $S(i) = i$ then customer i is understood to become her own right neighbour by becoming the *founder* of a new table (customer 1 is necessarily a founder). This process gives rise to a sequence $(\pi_n)_{n\in\mathbb{N}}$ of permutations π_n in the symmetric group Sym_n, where π_n indicates for a customer $i \in [n]$ the customer $\pi_n(i)$ sitting directly to the right of customer i at time n. An example is given in Figure 2. The figure also visualises the *cycles* of π_n as the connected components in the graph $([n], \{(i, \pi(i)) \mid i \in [n]\})$ and showcases the well-subscribed cycle notation.

The distribution of S involves fitness values $f_1, f_2, \ldots \in (0, \infty)$ assigned to customers (say the tastiness of their dish). Formally, when $n-1$ customers have entered the restaurant, and $S(1), \ldots, S(n-1)$, $f_1, \ldots, f_{n-1}$ and $\pi_1, \ldots, \pi_{n-1}$ are given, the n-th customer enters.

- She either chooses $S(n) = i \in [n-1]$ at random with probability

$$\mathbb{P}(S(n) = i) = \frac{f_i}{1+\sum_{j=1}^{n-1} f_j}.$$

  In this case $f_n = f_i$ (customer n orders the same dish as her neighbour) and $\pi_n(i) = n, \pi_n(n) = \pi_{n-1}(i)$ and $\pi_n(k) = \pi_{n-1}(k)$ for $k \in [n] \setminus \{i, n\}$.
- Otherwise she chooses $S(n) = n$ with probability

$$\mathbb{P}(S(n) = n) = \frac{1}{1+\sum_{j=1}^{n-1} f_j}.$$

  In this case customer n occupies a new table. We sample $f_n \sim \Phi$ and let $\pi_n(n) = n$ and $\pi_n(k) = \pi_{n-1}(k)$ for $k \in [n-1]$. This always occurs for customer 1.

Further quantities of interest are easily derived from S and (π_n). The set $\mathcal{F}$ of founders is precisely the set of fixed points of S, i.e.

$$\mathcal{F} = \{n \in \mathbb{N} \colon S(n) = n\}.$$

The founders $\mathcal{F} = \{n_1 < n_2 < \ldots\}$ can act as representatives of their tables, i.e. by the k-th table we mean the table at which n_k is placed. The number of non-empty tables at time n is $k_n = |\mathcal{F} \cap [n]|$, which is also the number of cycles in the permutation π_n. The customers at table n_k at time n are the elements of the cycle of π_n containing n_k. To find the customer $\mathrm{founder}(n)$ who founded the table at which customer n is sitting we can iteratively apply S starting with the argument n, until (after at most n iterations) we find a fixed point.

Related Objects. The classical single type Chinese restaurant process has only a single parameter $\theta > 0$, which is a weight for the probability of founding a new table. It arises as a special case of our variant when Φ is the trivial distribution putting all probability mass on the value θ^{-1}. Several aspects of the CRP are known under different names.

- If a_i denotes the number of tables of size i after n steps of the CRP with parameter θ then the distribution of $(a_1, \ldots, a_n)$ is known as the Ewens distribution.

- If we treat all fitness values as though they were 1 and reinterpret f_i as the dish ordered by customer i (with all dishes having the same quality) then $f_1, f_2, \dots$ is the Dirichlet process with base distribution Φ.
- If we initialise the CRP with some customers at some of the tables, all of which have the same fitness, and condition on the event that no further tables are created, then the number of customers at the tables over time is a Pólya urn model.
- Call a directed graph with vertex set $V = [n]$, where each $v \in V$ has exactly one outgoing edge towards $p(v) \leq v$ (loops are allowed) a *recursive forest*. If $\theta = 1$ then the graph $T = ([n], \{(i, S(i)) \mid i \in [n]\})$ generated from the outcome S of the CRP is a uniformly random recursive forest. If we condition on the event $\{\forall i > 1 : S(i) \neq i\}$ then T is a uniformly random recursive tree.

2.2 Formal statement of main result

In this paper, an event E_n occurs with high probability (whp) if for any constant $c > 0$ we have $1 - \mathbb{P}(E_n) = \mathcal{O}(n^{-c})$. We allow the constant hidden by $\mathcal{O}$-notation to depend on c. For a random variable X_n and a function g we say $X_n = \mathcal{O}(g(n))$ whp if

$$\forall c > 0 : \exists n_0, C : \forall n \geq n_0 : \mathbb{P}(X_n > C \cdot g(n)) \leq n^{-c},$$

in particular C may depend on c. Conveniently, whenever we have a polynomial number of events that individually occur whp, then these events jointly occur whp. We say X_n is $\mathrm{polylog}(n)$ whp if $X_n = \mathcal{O}(\log^b(n))$ whp for some constant $b > 0$.

Local Access Implementation. Our goal is known under several different names. We loosely follow the terminology of *local access implementation* by [5], which is a *stateful implementation* in the sense of [15] and a *random access generator* in the sense of [12].

Assume we are given a huge random object X by a distribution D on a set $\mathbb{X}$. A local access implementation of a family $F_1, \dots, F_q : \mathbb{X} \to R$ of attributes (random variables) with values in some set R is a data structure that answers a sequence of (possibly adaptive) queries $i_1, i_2, \dots$ with a sequence of results $r_1, r_2, \dots$ such that $(r_1, r_2, \dots)$ has the same distribution as $(F_{i_1}(X), F_{i_2}(X), \dots)$. If, for every $x \in \mathbb{X}$, the attributes $F_1(x), \dots, F_q(x)$ determine the object x completely we also speak of a local access implementation of X.

We can now formally state our main result.

▶ **Theorem 2** (Local Access Implementation of the CRP). *Let Φ be a distribution on $(0, \infty)$ with some means of sampling from Φ (e.g. using an oracle), and let $N \in \mathbb{N}$. A local access implementation of the N-round CRP with parameter Φ providing access to the attributes listed in Table 1 can be achieved such that each (possibly adaptive) query takes* $\mathrm{polylog}(N)$ *time whp.*

Note that the family of attributes grows with N. For instance, we can query the Chinese restaurant process for $\pi_n(c)$ for any $1 \leq c \leq n \leq N$. In the flat view used above this constitutes a separate attribute for every valid pair (n, c). Our implementation must be given a parameter $N \in \mathbb{N}$ beforehand and then one can query all attributes arising from rounds with index $n \leq N$.

Model of Computation. Since we use an embedding of the CRP in continuous time, our algorithms involve uniform sampling from $[0, 1] \subseteq \mathbb{R}$ and arithmetic operations on real numbers as would be allowed in the real RAM model. We suspect that our construction could be adapted for the word RAM model with moderate technical complications regarding the content of Section 6, but we do not pursue such a result.

attribute	parameters	interpretation
$\pi_n(c)$ and $\pi_n^{-1}(c)$	$1 \le c \le n \le N$	customer sitting right and left of customer c after n rounds
f_c	$1 \le c \le N$	fitness of customer c, i.e. weight for the probability with which customers are seated next to c.
founder(c)	$1 \le c \le N$	founder of customer c's table
tableSizes(n)	$1 \le n \le N$	number of customers at each of the k_n tables after n rounds in the order in which tables were founded
$\mathcal{F} \cap [N]$	—	set of all k_N founders within the first N rounds

Table 1 Attributes that can be queried by our local access implementation of the CRP and corresponding parameters.

3 Local access implementation of Random Recursive Trees

The *random recursive tree* T_N is the N-th element in a random sequence $(T_n)_{n\in\mathbb{N}}$ where T_n is a rooted tree with vertex set $[n]$ and T_{n+1} arises from T_n by assigning the new vertex $n+1$ a single neighbour parent$(n+1) \in [n]$ uniformly at random. In this section, we provide a random access generator for T_N in the following sense.

▶ **Proposition 3** (Local access implementation of the RRT). *Let $N \in \mathbb{N}$. A local access implementation of T_N that provides for any given vertex $v \in [N]$ access to all neighbours of v can be achieved such that each query takes* polylog(N) *time whp.*

A proof of this proposition is implicit in the work of Even et al. [12]. We still provide our own construction, which uses a slightly simplified invariant and exploits what we call the harmonic sampling trick. We begin with two facts that are also stated in [12].

▶ **Lemma 4.** **(i)** *The maximum degree of T_N is $\mathcal{O}(\log(N))$ whp.*
(ii) *The height of T_N is $\mathcal{O}(\log(N))$ whp.*

Proof. **(i)** Let c_i be the number of children of vertex i. We have $c_i = \sum_{j=i+1}^{N} X_j$ where X_j is the indicator that i is the parent of j. These indicators are independent and X_j is Bernoulli distributed with parameter $\frac{1}{j-1}$, hence $\mathbb{E}[c_i] = \mathcal{O}(\log(N))$. Moreover, c_i is *Poisson binomial* distributed and simple Chernoff bounds for this case suffice to show that $c_i = \mathcal{O}(\log(N))$ whp. A more fine-grained analysis is provided in [9, Thm 1].
(ii) A proof can be found in [14, Thm 6.32]. ◀

To prove Proposition 3 we need two simple ideas presented in the following two lemmas.

▶ **Lemma 5** (Harmonic Sampling Trick). *For $i \in [N]$ let $X_i \sim \mathrm{Ber}(\frac{1}{i})$ be independent Bernoulli random variables and $X = \{i \in [N] \mid X_i = 1\}$. The algorithm HST on the right samples X in $\mathcal{O}(\log N)$ time whp.*

Algorithm *HST($N \in \mathbb{N}$)*:
```
X ← ∅
while N ≥ 1 do
    sample i ∈ [N] uniformly
    X ← X ∪ {i}
    N ← i − 1
return X
```

Proof. First note that $\Pr[\max(X) = i] = \Pr[X_i = 1, X_{i+1} = \ldots = X_N = 0] = \frac{1}{i} \cdot \frac{i}{i+1} \cdot \ldots \cdot \frac{n-1}{n} = \frac{1}{n}$. Hence $\max(X)$ is uniformly distributed in $[N]$ and is correctly sampled in the first iteration of the while-loop. The remaining set $X \setminus \{\max(X)\} = \{j \in [\max(X) - 1] \mid X_j = 1\}$ can then be sampled using the same method since no information on $X_1, \ldots, X_{\max(X)-1}$ has been revealed.

To bound the running time of HST, we use a connection to random recursive trees. In T_{N+1}, the parent of a vertex $v \in [N+1] \setminus \{1\}$ is uniformly distributed in $[v-1]$. Therefore the set X' of ancestors of vertex $N+1$ in T_{N+1} can be sampled using HST and X' therefore has the same distribution as X. We can now use Lemma 4 **(ii)** to conclude that X has size $\mathcal{O}(\log N)$ whp and thus HST runs in time $\mathcal{O}(\log N)$ whp. ◀

▶ **Lemma 6** (Auxiliary Set Data Structure). *There is a data structure for representing a dynamic set $Q \subseteq [N]$ (initialised with $Q = \varnothing$) and its complement $\bar{Q} := [N] \setminus Q$ that uses $\mathcal{O}((|Q|+1)\log N)$ bits and supports the following operations in $\mathcal{O}(\log N)$ time.*

- **(i)** *Deciding for $v \in [N]$ whether $v \in Q$.*
- **(ii)** *Adding an element $v \in \bar{Q}$ to Q.*
- **(iii)** *Computing for $v \in \bar{Q}$ the number $\operatorname{rank}_{\bar{Q}}(v) = |\bar{Q} \cap [v]|$.*
- **(iv)** *Selecting for $r \in [N]$ the unique element $v \in \bar{Q}$ with $\operatorname{rank}_{\bar{Q}}(v) = r$, if it exists.*

Proof. Simply store Q in a balanced search tree data structure (e.g. an AVL tree) where subtrees are annotated with the number of elements they contain (see e.g. [31]). Implementing the operations as stated is then straightforward. ◀

Proof of Proposition 3. The tree T_N is determined by a family $(\text{parent}(v))_{2 \le v \le N}$ of independent random variables. Whenever we reveal the neighbourhood of some $v_0 \in [N]$ then this fully reveals $\text{parent}(v_0)$ and $\text{parent}(c_1), \dots, \text{parent}(c_k)$ for the children $c_1, \dots, c_k$ of v_0. It also reveals that $\text{parent}(v) \neq v_0$ for all $v \in \{v_0+1, \dots, N\} \setminus \{c_1, \dots, c_k\}$. Since every piece of information relates only to one random variable, the family $(\text{parent}(v))_{2 \le v \le N}$ is still independent conditioned on that information (this would not be true if we only revealed $\deg(v_0)$ for instance).

In contrast to Even et al. [12] we always reveal the entire neighbourhood of a requested vertex v_0. At any point in time let Q be the set of vertices that were previously queried and $\bar{Q} := [N] \setminus Q$. Our invariant is very simple:

> Conditioned on the information we have revealed, the family $(\text{parent}(v))_{2 \le v \le N}$ is still independent. For each $2 \le v \le N$, either $\text{parent}(v)$ is known (and not random any more) or uniformly distributed in $\bar{Q} \cap [v-1]$.

We use the data structure from Lemma 6 to store Q and $\bar{Q}$. Moreover we store all edges that were previously revealed.

We now explain how a query for $v \in [N]$ is handled. If $v \in Q$ then we simply reproduce the answer previously returned for v. Now assume $v \in \bar{Q}$. If $\text{parent}(v)$ is not yet revealed we have to select it uniformly from $\bar{Q} \cap [v-1]$. To this end we compute $\operatorname{rank}_{\bar{Q}}(v) = |\bar{Q} \cap [v-1]|$, then sample r' uniformly from $[\operatorname{rank}_{\bar{Q}}(v) - 1]$ and select as $\text{parent}(v)$ the element v' with $\operatorname{rank}_{\bar{Q}}(v') = r'$ using the $\mathcal{O}(\log N)$ time operations from Lemma 6. Some vertices from Q may already be known children of v. Moreover, every vertex $v' \in \{v+1, \dots, N\} \cap \bar{Q}$ where $\text{parent}(v')$ is not yet known has a probability of $\frac{1}{\operatorname{rank}(v')-1}$ to have v as its parent. Call such vertices *potential children* of v. It is important to note that two potential children $v'_1 < v'_2$ have distinct unit fractions from $\{\frac{1}{1}, \frac{1}{2}, \dots, \frac{1}{N}\}$ as a probability for having v as parent because the potential parents of v'_2 include all potential parents of v'_1 and v'_1 itself. We sample $X \subseteq [N]$ using Lemma 5, which contains each $i \in [N]$ with probability $\frac{1}{i}$. We then make a potential child v' of v an actual child of v if $\operatorname{rank}(v') - 1 \in X$. To do this quickly, we iterate over the (small) set X and check for each $i \in X$ with a select-query whether a corresponding potential child of v exists (and ignoring i if this is not the case). As a last step we add v to Q and return v's parent and children.

Note that (adaptive) queries can affect the way in which T_N is sampled but not its distribution. Since a query for $v \in [N]$ takes time $\deg(v) \cdot \mathcal{O}(\log n)$ its running time is $\mathrm{polylog}(N)$ whp by Lemma 4 **(i)**. ◀

4 Local access implementation of the Single Table Process

The *single table process* (STP) is the CRP conditioned on the event $\{\mathcal{F} = \{1\}\}$, i.e. no tables are founded after the first. This makes the parameter Φ irrelevant. In simple terms, the STP generates a sequence $(\tau_n)_{n\in\mathbb{N}}$ of cyclic permutations where $\tau_1 = (1)$ and where $\tau_{n+1} \in \mathrm{Sym}_{n+1}$ is obtained from τ_n by inserting $n+1$ into the cycle at a uniformly random position. Clearly this makes τ_n a uniformly random cyclic permutation. We now implement fast random access to this process in the following sense by exploiting a close correspondence between the STP and the random recursive trees $(T_n)_{1\le n\le N}$ from the previous section.

▶ **Proposition 7** (Local access implementation of the STP). *Let $N \in \mathbb{N}$. A local access implementation of the N-round STP that provides access to $\tau_n(c)$ and $\tau_n^{-1}(c)$ for any given $1 \le c \le n \le N$ can be achieved such that queries take* $\mathrm{polylog}(N)$ *time whp.*

Proof of Proposition 7. While T_n is defined as an unordered tree, we may assume that the children of a vertex are implicitly decreasingly ordered. The unique depth first search traversal of T_n produces a vertex list $L_n = (v_1 = 1, v_2, \dots, v_n)$. We can associate this list with a cyclic permutation τ'_n via

$$\tau'_n(v_i) = v_{i+1} \text{ for } i \in [n-1] \text{ and } \tau'_n(v_n) = v_1.$$

By definition T_{n+1} is obtained from T_n by prepending the vertex $n+1$ to the child list of a uniformly random vertex from $[n]$. Thus, the list L_{n+1} is obtained from L_n by inserting $n+1$ after a uniformly random element, and τ'_{n+1} is obtained from τ'_n by inserting $n+1$ into a uniformly random position of the cycle. So clearly the sequence $(\tau'_n)_{1\le n\le N}$ has the same distribution as the sequence $(\tau_n)_{1\le n\le N}$ from the STP. A query for $\tau_n(i)$ and $\tau_n^{-1}(i)$ concerning the STP corresponds to a query for the preorder successor and preorder predecessor of the vertex i in T_n (where v_1 is regarded as the successor of v_n in T_n).

By Proposition 3 we have access to adjacencies in T_N in polylogarithmic time. This also gives us access to adjacencies in the subtree T_n of T_N simply by ignoring any incidences to vertices $v > n$. To determine preorder successors and predecessors in T_n as required, we need only follow a path in T_n which comes with an additional $\mathcal{O}(\log(N))$ factor by Lemma 4 **(ii)**. ◀

5 Local access implementation of the Simple Birth Process

In this section we provide a random access generator for simple death processes. By reversing time, we then extend the result to (bounded) simple *birth* processes.

Simple Death Process. In a *simple death process* (SDP) with N elements and parameter $f > 0$, each element $i \in [N]$ is assigned a lifetime $\ell_i \sim \mathrm{Exp}(f)$ independently. The SDP is then $(Y_t)_{t\ge 0}$ where $Y_t := |\{i \in [N] \mid \ell_i \ge t\}|$ is the number of elements surviving until time t and the *duration* τ of the process is defined as $\tau := \inf\{t \ge 0 \mid Y_t = 0\}$.

Recall four simple facts abouts exponentially distributed random variables.

▶ **Fact 8.** **(i)** $\mathbb{P}(X \ge t) = \mathrm{e}^{-\lambda\cdot t}$ *for* $\lambda > 0$, $t \ge 0$ *and* $X \sim \mathrm{Exp}(\lambda)$.
(ii) $\mathbb{P}(X \ge t \mid X \ge t_0) = \mathbb{P}(X \ge t - t_0)$ *for* $\lambda > 0$, $t \ge t_0 \ge 0$ *and* $X \sim \mathrm{Exp}(\lambda)$.
This property is known as the memorylessness *of the exponential distribution.*

(iii) *For $k \in \mathbb{N}$, $\lambda_1, \dots, \lambda_k > 0$ and independent $X_i \sim \mathrm{Exp}(\lambda_i)$ for $i \in [k]$ we have*

$$\min_{i\in[k]} X_i \sim \mathrm{Exp}(\sum_{i\in[k]} \lambda_i) \text{ and } \mathbb{P}(j = \arg\min_{i\in[k]} X_i) = \frac{\lambda_j}{\sum_{i\in[k]} \lambda_i} \text{ for } j \in [k].$$

In the lemma that follows we need to quickly sample binomial random variables and use the following theorem from [7], which even works in the word RAM model.

▶ **Theorem 9** ([7, Section A.2 Thm 5]). *On a word RAM with word size $\Omega(\log N)$ it is possible to sample from $\mathrm{Bin}(N, \frac{1}{2})$ in expected time $\mathcal{O}(1)$ and whp in time $\mathcal{O}(\log N)$.*

▶ **Proposition 10** (Local access implementation of the SDP). *Let $N \in \mathbb{N}$ and $f > 0$. A local access implementation of a SDP $(Y_t)_{t\geq 0}$ with parameters N and f that provides access to τ and Y_t for given $t \in [0, \tau]$ can be achieved such that queries take $\mathrm{polylog}(N)$ time whp.*

Proof. If $(Y_t)_{t\geq 0}$ and $(Y'_t)_{t\geq 0}$ are SDPs with parameters 1 and f, respectively, and both with N elements, then $(Y_{ft})_{t\geq 0} \stackrel{d}{=} (Y'_t)_{t\geq 0}$. In other words, the parameter f only rescales time so we may assume $f = 1$ without loss of generality. The proof idea is to first describe how an outcome of a SDP can be represented using a binary tree. Second, we show how queries can be answered quickly using the tree. Lastly, we argue that the tree can be lazily generated.

Rather than sampling the lifetimes $(\ell_i)_{i\in[N]}$ from $\mathrm{Exp}(1)$, we sample $(\ell'_i)_{i\in[N]}$ uniformly from the interval $(0, 1)$ and define $\ell_i := -\ln(\ell'_i)$. This works because for $t \geq 0$

$$\mathbb{P}(\ell_i > t) = \mathbb{P}(-\ln(\ell'_i) > t) = \mathbb{P}(\ln(\ell'_i) < -t) = \mathbb{P}(\ell'_i < \mathrm{e}^{-t}) = \mathrm{e}^{-t}, \tag{1}$$

meaning $\ell_i \sim \mathrm{Exp}(1)$ as desired. We may assume that the set $L = \{\ell'_1, \dots, \ell'_N\}$ has size N (values are pairwise distinct). Instead of L we consider a binary tree $T = T(L)$ that is defined recursively. The root of T is *responsible* for $[0, 1)$. If a vertex v of T is responsible for an interval $[a, b) \subseteq [0, 1)$ then the subtree rooted at v stores $L \cap [a, b)$. Let $s(v) := |L \cap [a, b)|$. If $s(v) \geq 2$ then v has two children v_1 and v_2 responsible for $[a, \frac{a+b}{2})$ and $[\frac{a+b}{2}, b)$, respectively. If $s(v) \leq 1$ then v is a leaf and if $s(v) = 1$ then v is annotated with the unique element from $L \cap [a, b)$.

Note that we need not store the values a and b because they are implicit in the location of v in T. More precisely we have $a = \frac{i}{2^d}$ and $b = \frac{i+1}{2^d}$ where d is the depth of v in T and i is the binary number obtained when encoding the path from the root of T to v as a sequence of left (0) and right (1) choices. We do however store the value $s(v)$ explicitly in v.

It is quite clear that T has height $\mathcal{O}(\log N)$ whp because an inner vertex at depth d means that two values $x, y \in L$ fall within the same interval of size 2^{-d}. The expected number of such pairs is $\mathcal{O}(N^2 2^{-d})$, which is $\mathcal{O}(N^{-c})$ for $d \geq (c + 2) \log_2 N$. It is also clear that T allows us to answer any query in time proportional to the depth of T: To compute τ it suffices to find the left-most leaf of τ (with minimal ℓ'_i, hence corresponding to maximal ℓ_i). To compute Y_t, observe that

$$Y_t = |\{i \in [N] \mid \ell_i \geq t\}| = |\{i \in [N] \mid -\ln(\ell'_i) \geq t\}| = |\{i \in [N] \mid \ell'_i \leq \mathrm{e}^{-t}\}|.$$

It therefore suffices to locate where e^{-t} would be in T and compute, using the values $s(v)$ along the path, the number of leafs that lie left of the path towards e^{-t}.

We now describe how T can be generated on the fly (or more precisely a tree with the same distribution). We generate the children of a vertex only when the vertex is first looked at. Initially there is only the root r annotated with $s(r) = N$. Now assume a vertex is

reached for the first time. If $s(v) = 0$ then there is nothing to do. If $s(v) \geq 2$ then we create its two children v_1 and v_2. Since each child is responsible for exactly half the interval that v is responsible for, we have $s(v_1) \sim \mathrm{Bin}(s(v), \frac{1}{2})$ and $s(v_2) = s(v) - s(v_1)$. We can sample $s(v_1)$ in $\mathcal{O}(\log N)$ time whp by Theorem 9. If $s(v) = 1$, then we instantiate the single value $\ell \in L$ that is represented in v by sampling it uniformly from the interval that v is responsible for.

During a single query one descending path in T has to be generated in this way, which takes $\mathcal{O}(\log(N) \cdot \mathrm{height}(T)) = \mathcal{O}(\log^2(N))$ time whp. Note again that the (possibly adaptive) queries may affect the way in which T is generated, but not its distribution. ◀

Simple Birth Process. Consider a Markov process $(X_t)_{t\geq 0}$ with $X_t \in \mathbb{N}_0$ for $t \in \mathbb{R}_{\geq 0}$ that is monotonic in t. Let $t_i := \inf\{t \geq 0 \mid X_t \geq i+1\}$ be the time when it jumps from $\leq i$ to $\geq i+1$ for $i \in \mathbb{N}_0$. If $X_0 = 1$ and the waiting times $\Delta_i := t_i - t_{i-1}$ are independent random variables with distribution $\mathrm{Exp}(i \cdot f)$ for some $f > 0$, then $(X_t)_{t\geq 0}$ is called a *simple birth process* (SBP) or *Yule-process* with parameter f. Intuitively a SBP is just a SDP that is run in reverse. The following lemma makes this precise.

▶ **Lemma 11.** *Let $N \in \mathbb{N}$ and $f > 0$. Let $(X_t)_{t\geq 0}$ be a SBP with parameter f and $(Y_t)_{t\geq 0}$ a SDP with parameters N and f. Let τ denote the random duration of $(Y_t)_{t\geq 0}$ and $t_N := \inf\{t \geq 0 \mid X_t > N\}$. Then*

$$(\tau, (Y_t)_{t>0}) \stackrel{d}{=} (t_N, (X_{t_N - t})_{t>0}) \text{ where we define } X_t = 0 \text{ for } t < 0.$$

Proof. The amount of time that $(Y_t)_{t\geq 0}$ lingers at value N is $\min_{i\in[N]} \ell_i \sim \mathrm{Exp}(N \cdot f)$ by Fact 8 **(iii)**. Using the memorylessness of the exponential distribution (Fact 8 **(ii)**) and induction we can say more generally that $(Y_t)_{t\geq 0}$ lingers at value i for an $\mathrm{Exp}(i \cdot f)$-distributed time, independently for each $i \in [N]$. By definition, the same is true for $(X_t)_{t\geq 0}$, except in reverse order and for all $i \in \mathbb{N}$. The claimed distributional equality follows easily. Note that we had to remove the exceptional case $t = 0$ due to $Y_0 = N \neq N + 1 = X_\tau$. ◀

6 Local access implementation of the Table Growth Process

In this section we capture the aspects of the CRP that remain if customers are indistinguishable, i.e. those aspects related only to the sizes of tables.

The Table Growth Process. The *table growth process* (TGP) is, like the CRP, parametrised by a distribution Φ on $(0, \infty)$. For $n \in \mathbb{N}_0$, the n-th state S_n has the form

$$S_n = \left((a_n^{(1)}, \ldots, a_n^{(k_n)}),\ (f^{(1)}, \ldots, f^{(k_n)})\right)$$

where $k_n \leq n$ is a number of tables, $(a_n^{(1)}, \ldots, a_n^{(k_n)})$ are table sizes with $\sum_{j=1}^{k_n} a_n^{(j)} = n$, and $(f^{(1)}, \ldots, f^{(k_n)})$ are table fitness values. The process begins with $k_0 = 0$, i.e. $S_0 = ((), ())$. Given the n-th state, there are $k_n + 1$ possibilities for the $(n+1)$-th state: Either for some $j \in [k_n]$ the j-th table gains a customer or a new table is created. With the normalisation factor $Z = 1 + \sum_{j=1}^{k_n} a_n^{(j)} f^{(j)}$ we have: With probability $a_n^{(j)} f^{(j)}/Z$ the j-th table grows to size $a_{n+1}^{(j)} = a_n^{(j)} + 1$ while all other table sizes are unchanged. With the remaining probability $1/Z$ a new table is founded, meaning that $k_{n+1} = k_n + 1$, $a_{n+1}^{(k_n+1)} = 1$ and $f^{(k_{n+1})}$ is sampled from Φ. We prove the following.

▶ **Proposition 12** (Local access implementation of the TGP)**.** *Let Φ be a distribution on $(0, \infty)$ with some means of sampling from Φ, and let $N \in \mathbb{N}$. A local access implementation of the*

N-round TGP that provides access to S_n for any given $1 \le n \le N$ can be achieved such that queries take polylog(N) *time whp.*

Note that this implies that the total number k_N of tables is at most polylog(N) whp simply because a query's output size is a lower bound on its running time.

Continuous Time TGP. We now describe the *continuous-time table growth process* (CT-TGP), which is closely linked to the TGP with parameter Φ. Let $\delta_j \sim \text{Exp}(1)$ for $j \in \mathbb{N}$ be independent random variables, let $s_j := \sum_{i=1}^{j} \delta_j$ be the *creation time* of the j-th table, let $f^{(j)} \sim \Phi$ be the fitness of the j-th table and let $(X_t^{(j)})_{t \ge 0}$ be a SBP with parameter $f^{(j)}$ that describes the size $X_{t-s_j}^{(j)}$ of the j-th table at any time $t \ge s_j$. The state $S'(t)$ of the CT-TGP at time $t \ge 0$ describes the fitness and sizes of all tables at time t, formally defined as

$$S'(t) := \Big((X_{t-s_1}^{(1)}, \dots, X_{t-s_{k(t)}}^{(k(t))}),\ (f^{(1)}, \dots, f^{(k(t))}) \Big)$$

where $k(t) := \max\{j \in \mathbb{N}_0 \mid s_j \le t\}$ is the number of tables created at time t. The number $\text{count}(t) := \sum_{j=1}^{k(t)} X_{t-s_j}^{(j)}$ of customers at time t is clearly monotone in t. Let us define the n-th state S'_n of the CT-TGP as $S'_n = S(t_n)$ where $t_n := \inf\{t \ge 0 \mid \text{count}(t) \ge n\}$. With probability 1 all t_n are distinct and $\text{count}(t_n) = n$ for all $n \in \mathbb{N}_0$.

We now show that the sequence of states in the CT-TGP and in the TGP have the same distribution. This type of argument, known as Athreya-Karlin embedding in the literature on urn processes, is well suited for the study of processes with type or fitness dependent progression rules, see for example [20].

▶ **Lemma 13.** *Let Φ be as in Proposition 12. The sequences $(S_n)_{n \in \mathbb{N}_0}$ and $(S'_n)_{n \in \mathbb{N}_0}$ of states traversed by the TGP and the CT-TGP, respectively, have the same distribution.*

Proof. The sequences of fitness values are independently sampled from Φ in both cases, so it suffices to show that the distribution of the table sizes coincide for the TGP and the CT-TGP, when both processes are conditioned on using any fixed sequence $(f^{(j)})_{j \in \mathbb{N}}$ of fitness values. We may then suppress fitness values when writing states, i.e. we write $S_n = (a_n^{(1)}, \dots, a_n^{(k_n)})$ and $S'_n = (X_{t-s_1}^{(1)}, \dots, X_{t-s_{k(t)}}^{(k(t))})$. We shall consider the probabilities for all possible state transitions. Let therefore $Y = (y^{(1)}, \dots, y^{(k)}) \in \mathbb{N}^k$ be any state with $k \in \mathbb{N}$ and $\sum_{j \in [k]} y^{(j)} = n$. Morever let $Y_j := (y^{(1)}, \dots, y^{(j)} + 1, \dots, y^{(k)})$ for $j \in [k]$ and $Y_0 := (y^{(1)}, \dots, y^{(k)}, 1)$ be possible successor states of Y. Since both the TGP and the CT-TGP are Markov processes starting with $S_0 = S'_0 = ()$, it suffices to show that

$$\mathbb{P}(S_{n+1} = Y_j \mid S_n = Y) = \mathbb{P}(S'_{n+1} = Y_j \mid S'_n = Y) \text{ for all } Y \text{ and all } j = 0, \dots, k. \tag{2}$$

In the TGP we have with $Z = 1 + \sum_{j \in [k]} y^{(j)} \cdot f^{(j)}$ by definition

$$\mathbb{P}(S_{n+1} = Y_j \mid S_n = Y) = \begin{cases} y^{(j)} \cdot f^{(j)} / Z & \text{for } j \in [k] \\ 1/Z & \text{for } j = 0. \end{cases}$$

For the CT-TGP the situation is similar: Conditioned on $S'_n = Y$ being the state at time t_n the delay until the next time $t^{(j)} > t_n$ when table $j \in [k]$ grows has distribution $t^{(j)} - t_n \sim \text{Exp}(y^{(j)} \cdot f^{(j)})$ by the definition of SBPs and by the memorylessness of the exponential distribution (see Fact 8 **(ii)**). Similarly the delay until the time $t^{(0)} = s_{k+1}$ when the $(k+1)$-th table is founded has distribution $t^{(0)} - t_n \sim \text{Exp}(1)$. Whichever of these $k+1$

events occurs first determines S'_{n+1}, i.e.

$$\mathbb{P}(S'_{n+1} = Y_j \mid S'_n = Y) = \mathbb{P}(\operatorname*{arg\,min}_{i\in\{0,\dots,k\}} t^{(i)} - t_n = j \mid S'_n = Y)$$
$$\overset{\text{Fact 8(iii)}}{=} \begin{cases} y^{(j)} \cdot f^{(j)}/Z & \text{for } j \in [k] \\ 1/Z & \text{for } j = 0. \end{cases}$$

This establishes Equation (2). ◀

This equivalence is the first crucial step for local access to the TGP.

Proof of Proposition 12. We promised a local access implementation of the TGP with parameters Φ and N. By Lemma 13 we may instead give a local access implementation of the CT-TGP that provides access to S'_n for given $n \in [N]$.

We begin by describing the *setup phase* where we determine the parameters of all relevant tables, i.e. those tables that are created before the sum of table sizes exceeds N. Using the same trick as in Equation (1) on Page 11, we sample $\delta'_1, \delta'_2, \dots$ uniformly from $[0, 1)$ and define the delays $\delta_1, \delta_2, \dots$ between table creations as $\delta_k = -\ln(\delta'_k)$, which ensures $\delta_1, \delta_2, \dots \sim \mathrm{Exp}(1)$. We can then compute the creation times $s_1, s_2, \dots$ of tables as $s_k := \sum_{j=1}^{k} \delta_j$ and sample the fitness values $f^{(1)}, f^{(2)}, \dots \sim \Phi$. For the k-th table we would instantiate, by definition of the CT-TGP a SBP with parameter $f^{(k)}$. However, we are interested in this SBP only until its size reaches N and may, by Lemma 11, instead instantiate a SDP with parameters $f^{(k)}$ and $N + 1$. Let τ_k be the duration of the SDP. The size of table k for any time $t \in [s_k, s_k + \tau_k)$ can be assessed by querying the SDP for time $s_k + \tau_k - t$.

To decide after the creation of the first k tables whether a $(k + 1)$-th table is needed, we consider its designated birth time s_{k+1}. If $s_{k+1} \geq s_j + \tau_j$ for some $j \in [k]$ then the birth of the $(k + 1)$-th table would occur after the j-th table has grown to size $N + 1$, so it is not needed. Otherwise we determine the size of the first k tables at time s_{k+1}. Let N_k be the sum of these sizes. If $N_k < N$ then the $(k + 1)$-th table is created, otherwise it is not needed.

We now argue that at most a polylogarithmic number of tables is created whp. This implies that the setup just described can be carried out in $\mathrm{polylog}(N)$ time by Proposition 10. We begin with a simple tail bound on the duration τ of a SDP with parameters f and N (recall that $\ell_i \sim \mathrm{Exp}(f)$ is the lifetime of the i-th element):

$$\mathbb{P}(\tau > t) = \mathbb{P}(\max_{i\in[N]} \ell_i > t) \leq N \cdot \mathbb{P}(\ell_1 > t) = N \cdot \mathrm{e}^{-ft}. \tag{3}$$

In particular $\tau = \mathcal{O}(\frac{\log N}{f})$ whp. Since we made no assumptions on Φ, we may occasionally see very small fitness values, but since we do not permit Φ to depend on N there is a constant $\varepsilon > 0$ such that $\mathbb{P}_{f\sim\Phi}(f \geq \varepsilon) \geq \frac{1}{2}$. Therefore there is whp at least one table j with parameter $f^{(j)} \geq \varepsilon$ among the first $\mathcal{O}(\log N)$ tables, which guarantees $\tau_j = \mathcal{O}(\log N)$ whp. The total number of tables is then at most $j + X$ where X is the number of tables scheduled for creation within $[s_j, s_j + \tau_j)$. Though we have not yet stated it in this way, table creation is governed by a Poisson process with parameter 1 and hence $X \sim \mathrm{Po}(\tau_j)$. A simple concentration argument implies that X is $\mathcal{O}(\log N)$ whp. Hence, the total number of tables is $j + X = \mathcal{O}(\log N)$ whp.

We now describe how queries to the CT-TGP are answered. Reporting fitness values is straightforward as all of them have been determined during setup. The challenge is to report for any given $n \in [N]$ the sizes of each table at some point in time when the sum of these sizes is n. The idea is quite simple: First determine the number of tables $k = k(n)$ that exist at such times by iterating over the numbers $N_0 = 0, N_1, N_2, \dots$ encountered during

setup. Typically, k is characterised by $N_{k-1} < n \leq N_k$ and we should focus on the time interval $I = [s_k, s_{k+1})$ when k tables exist. In the special case where $n > N_{k-1}$ and the $(k+1)$-th table was never created, we may not have computed N_k and we use the time interval $I = [s_k, \min_{j\in[k]} s_j + \tau_j)$. In both cases we use binary search to find a time point $t \in I$ where table sizes add up to n and answer the query with the corresponding state $S'(t)$ of the CT-TGP.

It should be clear that this approach yields the correct result. Moreover, we have already argued that the relevant number of tables is $k = \mathcal{O}(\log N)$ whp and by Proposition 10 a single table's size can be determined in polylog(N) time whp for any $t \in I$. To obtain polylog(N) running time overall whp the only thing left to consider is the number of rounds needed by the binary search. For this let $\hat{f} := \max_{j\in[k]} f^{(j)}$. Because $I \subseteq [s_j, s_j + \tau_j]$ for every $j \in [k]$ and using $\tau_j = \mathcal{O}(\frac{\log N}{f^{(j)}})$ by Equation (3), the size $|I|$ of I satisfies

$$|I| \leq \min_{j\in[k]} \tau_j = \mathcal{O}(\min_{j\in[k]} \tfrac{\log N}{f^{(j)}}) = \mathcal{O}(\tfrac{\log N}{\hat{f}}) \text{ whp.}$$

We also need a lower bound on the delays $\Delta_i = t_{i+1} - t_i$ between the arrival times of two consecutive customers. Recall that in the "flat view" of the CRP f_c for $c \in [N]$ is the fitness of customer c. Given the state at time t_i we have $\Delta_i \sim \text{Exp}(r_i)$ where $r_i = 1 + \sum_{c\in[i]} f_c$. We say the i-th delay is p-long if $\Delta_i \geq \frac{p}{r_i}$, which is the case with probability $\Pr[\Delta_i \geq \frac{p}{r_i}] = \Pr[\text{Exp}(r_i) \geq \frac{p}{r_i}] = e^{-p} \geq 1 - p$. In particular, for any $c > 0$ any $i \in [N-1]$ the i-th delay is N^{-c}-long with probability $1 - N^{-c}$. By union bound, all $N-1$ delays are (simultaneously) N^{-c-1}-long with probability $1 - N^{-c}$. In this sense all delays are $N^{-\mathcal{O}(1)}$-long whp. Let now $\hat{\Delta}$ be the smallest delay between any two consecutive arrivals within I. Assuming that all delays are $N^{-\mathcal{O}(1)}$-long we have $\hat{\Delta} \geq N^{-\mathcal{O}(1)}/\hat{r}$ where $\hat{r}$ is the largest arrival rate (r_i above) that occurs within I. Since the number of customers is bounded by N and the fitness of any customer is bounded by $\hat{f}$ we have $\hat{r} \leq N\hat{f}$. Hence

$$\hat{\Delta} = \tfrac{N^{-\mathcal{O}(1)}}{\hat{r}} = \tfrac{N^{-\mathcal{O}(1)}}{N\hat{f}} = \tfrac{N^{-\mathcal{O}(1)}}{\hat{f}}.$$

Using our bounds on $|I|$ and $\hat{\Delta}$ we conclude that the binary search takes at most

$$\log_2(|I|/\hat{\Delta}) = \log_2\left(\frac{\log N}{N^{-\mathcal{O}(1)}}\right) = \log_2(N^{\mathcal{O}(1)}) = \mathcal{O}(\log N)$$

steps whp as desired.

So far we have not explicitly worried about the fact that queries can be adaptive. Could an attacker, after collecting some information, concoct a specific query that is – while generally fast whp – exceptionally difficult conditioned on the information that has been revealed? Luckily this worry can be dispelled for the reason that the number of distinct attributes is small enough: Our bounds on query time relate only to circumstances regarding the random processes that the attacker does not control. Since these circumstances are favourable whp for any *fixed* query, they are – using our definition of "whp" – also simultaneously favourable for each of the polynomially many possible queries. This concludes the argument. ◀

7 Local access implementation of the Chinese Restaurant Process

We are finally ready to prove Theorem 2 based on Propositions 7 and 12.

Proof of Theorem 2. The setup is straightforward: To provide a local access implementation of the N-round CRP with parameter Φ, we use our local access implementation the N-round

TGP with parameter Φ, which determines how customers are distributed to tables. This yields the correct distribution because customers that share a table in the CRP process also share the same fitness value, so the way in which fitness is assigned to tables in the TGP is adequate. We query the TGP for $n = N$ to learn the final table sizes $(a_N^{(1)}, \dots, a_N^{(k_N)})$ and instantiate k_N copies of the STP with these sizes as parameters. The j-th STP provides access to a sequence $(\tau_n^{(j)})_{1 \le j \le a_N^{(j)}}$ of permutations and is responsible for the ordering of the customers at the j-th table. Note that fitness parameters are not needed since whenever a customer joins a specific table in the CRP all positions at that table are equally likely due to shared fitness values. Any query to the TGP takes polylog(N) time whp by Proposition 12 and any query to a STP takes polylog(N) time whp by Proposition 7. In particular the described setup takes polylog(N) time whp.

To answer CRP queries (see below), we have to translate between the two distinct ways in which customers are referenced by the TGP and the STPs. If the c-th customer overall joins the j-th table and is the c'-th customer at that table, then we call (c', j) the *local identity* of the customer while c is her *global identity*. Given the global identity c of a customer we can obtain her local identity by querying the TGP for round $c-1$ and round c. Let j be the index of the unique table that either grew in size or was newly created in round c. Then $(a_c^{(j)}, j)$ is the local identity of c. Conversely, given the local identity (c', j) of a customer, her global identity is the unique number c with $a_c^{(j)} = c'$ and $a_{c-1}^{(j)} = c' - 1$ (assuming $a_c^{(j)}$ has an implicit value of 0 when $k_c < j$). This number can be determined with binary search in $\mathcal{O}(\log N)$ queries to the TGP since $a_c^{(j)}$ is monotonic in c. Any translation operation takes polylog(N) time whp.

We now show how any query to the CRP can be answered by issuing an at most polylogarithmic number of queries to the $k_N + 1$ processes we have instantiated. In that context, we may freely translate between local and global identities as explained in the previous paragraph.

$\pi_n(c), \pi_n^{-1}(c)$. Let (c', j) be the local identity of customer c and $n' = a_n^{(j)}$ the size of her table after round n. The query asks for the global identities of the customers with local identities $(\tau_{n'}^{(j)}(c'), j)$ and $((\tau_{n'}^{(j)})^{-1}(c'), j)$, respectively. We obtain the values $\tau_{n'}^{(j)}(c')$ and $(\tau_{n'}^{(j)})^{-1}(c')$ by querying the j-th STP.

f_c. Simply return $f^{(j)}$ where (c', j) is the local identity of customer c.

founder(c). Let (c', j) be the local identity of customer c. Then founder(c) is the global identity of the customer with local identity $(1, j)$.

$\mathcal{F} \cap [N]$. This asks for the global identities of the customers with local identities $(1, j)$ for all $1 \le j \le k_N$.

tableSizes(n). This query can simply be forwarded to the TGP, the correct answer being $(a_n^{(1)}, \dots, a_n^{(k_n)})$. ◀

For reasons already discussed in the proof of Proposition 12, the fact that queries can be chosen adaptively poses no problem.

References

1 Alon, N., Rubinfeld, R., Vardi, S., and Xie, N. Space-efficient local computation algorithms. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA* (2012), SIAM, pp. 1132–1139.

2 Athreya, K. B., and Karlin, S. Embedding of urn schemes into continuous time markov branching processes and related limit theorems. *The Annals of Mathematical Statistics 39*, 6 (1968), 1801–1817.

3 Barabási, A.-L., and Albert, R. Emergence of scaling in random networks. *Science 286*, 5439 (1999), 509–512.

4 Biswas, A. S., Pyne, E., and Rubinfeld, R. Local access to random walks. In *13th Innovations in Theoretical Computer Science Conference, ITCS* (2022), M. Braverman, Ed., vol. 215 of *LIPIcs*, pp. 24:1–24:22.

5 Biswas, A. S., Rubinfeld, R., and Yodpinyanee, A. Local access to huge random objects through partial sampling. In *11th ITCS* (2020), vol. 151 of *LIPIcs*, Dagstuhl Publishing, pp. 27:1–27:65.

6 Björnberg, J., Mailler, C., Mörters, P., and Ueltschi, D. The weighted chinese restaurant process condenses in at most two tables. *In progress.* (2022).

7 Bringmann, K., Kuhn, F., Panagiotou, K., Peter, U., and Thomas, H. Internal DLA: efficient simulation of a physical growth model - (extended abstract). In *41st ICALP* (2014), vol. 8572 of *LNCS*, Springer, pp. 247–258.

8 Crane, H. The Ubiquitous Ewens Sampling Formula. *Statistical Science 31*, 1 (2016), 1 – 19.

9 Drmota, M. *Random Trees: An Interplay between Combinatorics and Probability*, 1st ed. Springer, 2009.

10 Durrett, R. *Probability models for DNA sequence evolution*. Springer, 2008.

11 Erdős, P., and Rényi, A. On random graphs. I. *Publ. Math. 6* (1959), 290–297.

12 Even, G., Levi, R., Medina, M., and Rosén, A. Sublinear random access generators for preferential attachment graphs. *ACM Trans. Algorithms 17*, 4 (2021), 28:1–28:26.

13 Gluch, G., Kapralov, M., Lattanzi, S., Mousavifar, A., and Sohler, C. Spectral clustering oracles in sublinear time. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms, SODA* (2021), SIAM, pp. 1598–1617.

14 Goh, W., and Schmutz, E. Limit distribution for the maximum degree of a random recursive tree. *Journal of Computational and Applied Mathematics 142*, 1 (2002), 61–82. Probabilistic Methods in Combinatorics and Combinatorial Optimization.

15 Goldreich, O., Goldwasser, S., and Nussboim, A. On the implementation of huge random objects. *SIAM J. Comput. 39*, 7 (2010), 2761–2822.

16 Grunau, C., Mitrovic, S., Rubinfeld, R., and Vakilian, A. Improved local computation algorithm for set cover via sparsification. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms, SODA* (2020), S. Chawla, Ed., SIAM, pp. 2993–3011.

17 Hassidim, A., Kelner, J. A., Nguyen, H. N., and Onak, K. Local graph partitions for approximation and testing. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009* (2009), pp. 22–31.

18 Hassidim, A., Mansour, Y., and Vardi, S. Local computation mechanism design. *ACM Trans. Economics and Comput. 4*, 4 (2016), 21:1–21:24.

19 Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social networks 5*, 2 (1983), 109–137.

20 Janson, S. Functional limit theorems for multitype branching processes and generalized pólya urns. *Stochastic Processes and their Applications 110*, 2 (2004), 177–245.

21 Joyce, P., and Tavaré, S. Cycles, permutations and the structure of the yule process with immigration. *Stochastic Processes and their Applications 25* (1987), 309–314.

22 Kumar, A., Seshadhri, C., and Stolman, A. Random walks and forbidden minors III: poly(d/ϵ)-time partition oracles for minor-free graph classes. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS* (2021), IEEE, pp. 257–268.

23 Levi, R., and Ron, D. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Trans. Algorithms 11*, 3 (2015), 24:1–24:13.

24 Levi, R., Ron, D., and Rubinfeld, R. Local algorithms for sparse spanning graphs. *Algorithmica 82*, 4 (2020), 747–786.

25 Li, M., and Ghosal, S. Bayesian Multiscale Smoothing of Gaussian Noised Images. *Bayesian Analysis 9*, 3 (2014), 733 – 758.

26 Mansour, Y., and Vardi, S. A local computation approximation scheme to maximum matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX, and 17th International Workshop, RANDOM* (2013), vol. 8096 of *Lecture Notes in Computer Science*, Springer, pp. 260–273.

27 Medvedovic, M., and Sivaganesan, S. Bayesian infinite mixture model based clustering of gene expression profiles. *Bioinformatics 18* (2002), 1194–1206.

28 Naor, M., and Nussboim, A. Implementing huge sparse random graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th International Workshop, APPROX, and 11th International Workshop, RANDOM* (2007), vol. 4627 of *Lecture Notes in Computer Science*, pp. 596–608.

29 Qin, Z. S. Clustering microarray gene expression data using weighted Chinese restaurant process. *Bioinformatics 22*, 16 (2006), 1988–1997.

30 Rubinfeld, R., Tamir, G., Vardi, S., and Xie, N. Fast local computation algorithms. In *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings* (2011), B. Chazelle, Ed., Tsinghua University Press, pp. 223–238.

31 Sanders, P., Mehlhorn, K., Dietzfelbinger, M., and Dementiev, R. *Sequential and Parallel Algorithms and Data Structures - The Basic Toolbox*. Springer, 2019.