Distributed Distinct Elements

Emin Karayel

September 13, 2023

Abstract

This entry formalizes a randomized cardinality estimation data structure with asymptotically optimal space usage. It is inspired by the streaming algorithm presented by Błasiok [3] in 2018. His work closed the gap between the best-known lower bound and upper bound after a long line of research started by Flajolet and Martin [4] in 1984 and was to first to apply expander graphs (in addition to hash families) to the problem. The formalized algorithm has two improvements compared to the algorithm by Błasiok. It supports operation in parallel mode, and it relies on a simpler pseudo-random construction avoiding the use of code based extractors.

Contents

1 Introduction

The algorithm is described as functional data strucutures, given a seed which needs to be choosen uniformly from a initial segment of the natural numbers and globally, there are three functions:

- single - given the seed and an element from the universe computes a sketch for that singleton set
- merge - computes a sketch based on two input sketches and returns a sketch representing the union set
- estimate - computes an estimate for the cardinality of the set represented by a sketch

The main point is that a sketch requires $\mathcal{O}(\delta^{-2}\ln(\varepsilon^{-1}) + \ln n)$ space where n is the universe size, δ is the desired relative accuracy and ε is the desired failure probability. Note that it is easy to see that an exact solution would necessarily require $\mathcal{O}(n)$ bits.

The algorithm is split into two parts an inner algorithm, described in Section 6, which itself is already a full cardinality estimation algorithm, however its space usage is below optimal. The outer algorithm is introduced in Section 10, which runs mutiple copies of the inner algorithm with carefully chosen inner parameters.

As mentioned in the abstract the algorithm is inspired by the solution to the streaming version of the problem by Błasiok [3] in 2020. His work builds on a long line of reasarch starting in 1985 [4, 1, 2, 8, 12, 5].

In an earlier AFP entry [10] I have formalized an earlier cardinality estimation algorithm based on the work by Bar-Yossef et al. [2] in 2002. Since then I have addressed the existence of finite fields for higher prime powers and expander graphs [9, 11]. Building on these results, the formalization of this more advanced solution presented here became possible.

The solution described here improves on the algorithms described by Błasiok in two ways (without comprising its optimal space usage). It can be used in a parallel mode of operation. Moreover the pseudo-random construction used is simpler than the solution described by Błasiok — who uses an extractor based on Parvaresh-Vardy codes [7] to sample random walks in an expander graph, which are then sub-sampled and then the walks are used to sample seeds for hash functions. In the solution presented here neither the sub-sampling step nor the extractor is needed, instead a two-stage expander construction is used, this means that the nodes of the first expander correspond to the walks in a second expander graph. The latters nodes correspond to seeds of hash functions (as in Błasiok's solution).

The modification needed to support a parallel mode of operation is a change in the failure strategy of the solution presented in Kane et al., which is the event when the data in the sketch reequires too much space. The main issue is that in the parallel case the number of states the algorithm might reach is not bounded by the universe size and thus an estimate they make for the probability of the failure event does not transfer to the parallel case. To solve that the algorithm in this work is more conservative. Instead of failing out-right it instead increases a cutoff threshold. For which it is then possible to show an upper estimate independent of the number of reached states.

2 Preliminary Results

This section contains various short preliminary results used in the sections below.

theory *Distributed-Distinct-Elements-Preliminary*
  imports
    Frequency-Moments.Frequency-Moments-Preliminary-Results
    Frequency-Moments.Product-PMF-Ext
    Median-Method.Median
    Expander-Graphs.Extra-Congruence-Method
    Expander-Graphs.Constructive-Chernoff-Bound
    Frequency-Moments.Landau-Ext
    Stirling-Formula.Stirling-Formula
begin

unbundle *intro-cong-syntax*

Simplified versions of measure theoretic results for pmfs:

lemma *measure-pmf-cong*:
  assumes $\bigwedge x.\ x \in \textit{set-pmf}\ p \Longrightarrow x \in P \longleftrightarrow x \in Q$
  shows $\textit{measure}\ (\textit{measure-pmf}\ p)\ P = \textit{measure}\ (\textit{measure-pmf}\ p)\ \ Q$
  ⟨*proof*⟩

lemma *pmf-mono*:
  assumes $\bigwedge x.\ x \in \textit{set-pmf}\ p \Longrightarrow x \in P \Longrightarrow x \in Q$
  shows $\textit{measure}\ (\textit{measure-pmf}\ p)\ P \leq \textit{measure}\ (\textit{measure-pmf}\ p)\ Q$
⟨*proof*⟩

lemma *pmf-rev-mono*:
  assumes $\bigwedge x.\ x \in \textit{set-pmf}\ p \Longrightarrow x \notin Q \Longrightarrow x \notin P$
  shows $\textit{measure}\ p\ P \leq \textit{measure}\ p\ Q$
  ⟨*proof*⟩

lemma *pmf-exp-mono*:
  fixes $f\ g :: {'a} \Rightarrow \textit{real}$
  assumes $\textit{integrable}\ (\textit{measure-pmf}\ p)\ f\ \textit{integrable}\ (\textit{measure-pmf}\ p)\ g$
  assumes $\bigwedge x.\ x \in \textit{set-pmf}\ p \Longrightarrow f\ x \leq g\ x$
  shows $\textit{integral}^L\ (\textit{measure-pmf}\ p)\ f \leq \textit{integral}^L\ (\textit{measure-pmf}\ p)\ g$
  ⟨*proof*⟩

lemma *pmf-markov*:
  assumes $\textit{integrable}\ (\textit{measure-pmf}\ p)\ f\ c > 0$
  assumes $\bigwedge x.\ x \in \textit{set-pmf}\ p \Longrightarrow f\ x \geq 0$
  shows $\textit{measure}\ p\ \{\omega.\ f\ \omega \geq c\} \leq (\int \omega.\ f\ \omega\ \partial p)/\ c$ (**is** $?L \leq ?R$)
⟨*proof*⟩

lemma *pmf-add*:
  assumes $\bigwedge x.\ x \in P \Longrightarrow x \in \textit{set-pmf}\ p \Longrightarrow x \in Q \vee x \in R$
  shows $\textit{measure}\ p\ P \leq \textit{measure}\ p\ Q + \textit{measure}\ p\ R$
⟨*proof*⟩

lemma *pair-pmf-prob-left*:
  $\textit{measure-pmf.prob}\ (\textit{pair-pmf}\ A\ B)\ \{\omega.\ P\ (\textit{fst}\ \omega)\} = \textit{measure-pmf.prob}\ A\ \{\omega.\ P\ \omega\}$ (**is** $?L = ?R$)
⟨*proof*⟩

lemma *pmf-exp-of-fin-function*:
  assumes $\textit{finite}\ A\ g\ `\ \textit{set-pmf}\ p \subseteq A$
  shows $(\int \omega.\ f\ (g\ \omega)\ \partial p) = (\sum y \in A.\ f\ y * \textit{measure}\ p\ \{\omega.\ g\ \omega = y\})$
    (**is** $?L = ?R$)
⟨*proof*⟩

Cardinality rules for distinct/ordered pairs of a set without the finiteness constraint - to use in simplification:

lemma *card-distinct-pairs*:
  $\textit{card}\ \{x \in B \times B.\ \textit{fst}\ x \neq \textit{snd}\ x\} = \textit{card}\ B\hat{\ }2 - \textit{card}\ B$ (**is** $\textit{card}\ ?L = ?R$)
⟨*proof*⟩
  include *intro-cong-syntax*
  ⟨*proof*⟩

lemma *card-ordered-pairs'*:
  fixes $M :: ({'a} :: \textit{linorder})\ \textit{set}$
  shows $\textit{card}\ \{(x,y) \in M \times M.\ x < y\} = \textit{card}\ M * (\textit{card}\ M - 1)\ /\ 2$
⟨*proof*⟩

The following are versions of the mean value theorem, where the interval endpoints may be reversed.

lemma *MVT-symmetric*:
assumes $\bigwedge x.\ [\![min\ a\ b \leq x;\ x \leq max\ a\ b]\!] \Longrightarrow DERIV\ f\ x :> f'\ x$
shows $\exists\ z{::}real.\ min\ a\ b \leq z \wedge z \leq max\ a\ b \wedge (f\ b - f\ a = (b - a) * f'\ z)$
$\langle proof \rangle$

lemma *MVT-interval*:
fixes $I :: real\ set$
assumes $interval\ I\ a \in I\ b \in I$
assumes $\bigwedge x.\ x \in I \Longrightarrow DERIV\ f\ x :> f'\ x$
shows $\exists\ z.\ z \in I \wedge (f\ b - f\ a = (b - a) * f'\ z)$
$\langle proof \rangle$

Ln is monotone on the positive numbers and thus commutes with min and max:

lemma *ln-min-swap*:
$x > (0{::}real) \Longrightarrow (y > 0) \Longrightarrow ln\ (min\ x\ y) = min\ (ln\ x)\ (ln\ y)$
$\langle proof \rangle$

lemma *ln-max-swap*:
$x > (0{::}real) \Longrightarrow (y > 0) \Longrightarrow ln\ (max\ x\ y) = max\ (ln\ x)\ (ln\ y)$
$\langle proof \rangle$

Loose lower bounds for the factorial fuction:.

lemma *fact-lower-bound*:
$sqrt(2*pi*n)*(n/exp(1))\hat{}n \leq fact\ n$ (**is** $?L \leq ?R$)
$\langle proof \rangle$

lemma *fact-lower-bound-1*:
assumes $n > 0$
shows $(n/exp\ 1)\hat{}n \leq fact\ n$ (**is** $?L \leq ?R$)
$\langle proof \rangle$

Rules to handle O-notation with multiple variables, where some filters may be towards zero:

lemma *real-inv-at-right-0-inf*:
$\forall_F\ x\ in\ at\text{-}right\ (0{::}real).\ c \leq 1\ /\ x$
$\langle proof \rangle$

lemma *bigo-prod-1*:
assumes $(\lambda x.\ f\ x) \in O[F](\lambda x.\ g\ x)\ \ G \neq bot$
shows $(\lambda x.\ f\ (fst\ x)) \in O[F \times_F G](\lambda x.\ g\ (fst\ x))$
$\langle proof \rangle$

lemma *bigo-prod-2*:
assumes $(\lambda x.\ f\ x) \in O[G](\lambda x.\ g\ x)\ \ F \neq bot$
shows $(\lambda x.\ f\ (snd\ x)) \in O[F \times_F G](\lambda x.\ g\ (snd\ x))$
$\langle proof \rangle$

lemma *eventually-inv*:
fixes $P :: real \Rightarrow bool$
assumes $eventually\ (\lambda x.\ P\ (1/x))\ at\text{-}top$
shows $eventually\ (\lambda x.\ P\ x)\ (at\text{-}right\ 0)$
$\langle proof \rangle$

lemma *bigo-inv*:
fixes $f\ g :: real \Rightarrow real$

assumes $(\lambda x.\ f\ (1/x)) \in O(\lambda x.\ g\ (1/x))$
shows $f \in O[\textit{at-right } 0](g)$
⟨*proof*⟩

unbundle *no-intro-cong-syntax*

end

3 Combinators for Pseudo-random Objects

This section introduces a combinator library for pseudo-random objects. Each object can be described as a sample space, a function from an initial segment of the natural numbers that selects a value (or data structure.) Semantically they are multisets with the natural interpretation as a probability space (each element is selected with a probability proportional to its occurence count in the multiset). Operationally the selection procedure describes an algorithm to sample from the space.

After general definitions and lemmas basic sample spaces, such as chosing a natural uniformly in an initial segment, a product construction the main pseudo-random objects: hash families and expander graphs are introduced. In both cases the range is itself an arbitrary sample space, such that it is for example possible to construct a pseudo-random object that samples seeds for hash families using an expander walk.

The definitions Ψ in Section 6 and Θ in Section 10 are good examples.

A nice introduction into such constructions has been published by Goldreich [6].

3.1 Definitions and General Lemmas

theory *Pseudorandom-Combinators*
imports
Finite-Fields.Card-Irreducible-Polynomials
Universal-Hash-Families.Carter-Wegman-Hash-Family
Frequency-Moments.Product-PMF-Ext
Distributed-Distinct-Elements-Preliminary
Expander-Graphs.Expander-Graphs-Strongly-Explicit
begin

unbundle *intro-cong-syntax*
hide-const *Quantum.T*
hide-const *Discrete-Topology.discrete*
hide-const *Polynomial.order*
no-notation *Digraph.dominates* $(- \rightarrow_1 -\ [100,100]\ 40)$

record *'a sample-space* =
size :: *nat*
sample-space-select :: *nat* ⇒ *'a*

definition *sample-pmf*
where *sample-pmf S* = *map-pmf* (*sample-space-select S*) (*pmf-of-set* {..<*size S*})

definition *sample-space S* ≡ *size S* > *0*

definition *select S k* = (*sample-space-select S* (*if k* < *size S then k else 0*))

definition *sample-set S* = *select S* ‘ {..<*size S*}

lemma *sample-space-imp-ne*:

assumes *sample-space S*
shows $\{..<$*size S*$\} \neq \{\}$
⟨*proof*⟩

lemma *sample-pmf-alt*:
assumes *sample-space S*
shows *sample-pmf S* = *map-pmf* (*select S*) (*pmf-of-set* $\{..<$*size S*$\}$)
⟨*proof*⟩

lemma *sample-space-alt*:
assumes *sample-space S*
shows *sample-set S* = *set-pmf* (*sample-pmf S*)
⟨*proof*⟩

lemma *sample-set-alt*:
assumes *sample-space S*
shows *sample-set S* = *sample-space-select S* ‘ $\{..<$*size S*$\}$
⟨*proof*⟩

lemma *select-range*:
assumes *sample-space S*
shows *select S i* ∈ *sample-set S*
⟨*proof*⟩

declare [[*coercion sample-pmf*]]

lemma *integrable-sample-pmf*[*simp*]:
fixes *f* :: *'a* ⇒ *'c*::{*banach*, *second-countable-topology*}
assumes *sample-space S*
shows *integrable* (*measure-pmf* (*sample-pmf S*)) *f*
⟨*proof*⟩

3.2 Basic sample spaces

Sample space for uniformly selecting a natural number less than a given bound:

definition *nat-sample-space* :: *nat* ⇒ *nat sample-space* ($[\text{-}]_S$)
where *nat-sample-space n* = ⦇ *size* = *n*, *select* = *id* ⦈

lemma *nat-sample-pmf*:
sample-pmf ($[x]_S$) = *pmf-of-set* $\{..<x\}$
⟨*proof*⟩

lemma *nat-sample-space*[*simp*]:
assumes $n > 0$
shows *sample-space* $[n]_S$
⟨*proof*⟩

Sample space for the product of two sample spaces:

definition *prod-sample-space* ::
'a sample-space ⇒ *'b sample-space* ⇒ (*'a* × *'b*) *sample-space* (**infixr** $\times_S$ *65*)
where
prod-sample-space s t =
⦇ *size* = *size s* ∗ *size t*,
select = (λ*i*. (*select s* (*i mod* (*size s*)), *select t* (*i div* (*size s*)))) ⦈

lemma *split-pmf-mod-div'*:
assumes *a* > (*0*::*nat*)

assumes $b > 0$
shows *map-pmf* ($\lambda x.$ (x *mod* a, x *div* a)) (*pmf-of-set* $\{..<a * b\}$) = *pmf-of-set* ($\{..<a\} \times \{..<b\}$)
⟨*proof*⟩

lemma *pmf-of-set-prod-eq*:
assumes $A \neq \{\}$ *finite A*
assumes $B \neq \{\}$ *finite B*
shows *pmf-of-set* ($A \times B$) = *pair-pmf* (*pmf-of-set A*) (*pmf-of-set B*)
⟨*proof*⟩

lemma *split-pmf-mod-div*:
assumes $a > (0::nat)$
assumes $b > 0$
shows *map-pmf* ($\lambda x.$ (x *mod* a, x *div* a)) (*pmf-of-set* $\{..<a * b\}$) =
pair-pmf (*pmf-of-set* $\{..<a\}$) (*pmf-of-set* $\{..<b\}$)
⟨*proof*⟩

lemma *split-pmf-mod*:
assumes $a > (0::nat)$
assumes $b > 0$
shows *map-pmf* ($\lambda x.$ x *mod* a) (*pmf-of-set* $\{..<a * b\}$) = *pmf-of-set* $\{..<a\}$
⟨*proof*⟩

lemma *prod-sample-pmf*:
assumes *sample-space S*
assumes *sample-space T*
shows *sample-pmf* ($S \times_S T$) = *pair-pmf* (*sample-pmf S*) (*sample-pmf T*) (**is** *?L = ?R*)
⟨*proof*⟩

lemma *prod-sample-space*[*simp*]:
assumes *sample-space S sample-space T*
shows *sample-space* ($S \times_S T$)
⟨*proof*⟩

lemma *prod-sample-set*:
assumes *sample-space S*
assumes *sample-space T*
shows *sample-set* ($S \times_S T$) = *sample-set S* $\times$ *sample-set T* (**is** *?L = ?R*)
⟨*proof*⟩

3.3 Hash Families

lemma *indep-vars-map-pmf*:
assumes *prob-space.indep-vars* (*measure-pmf p*) (λ-. *discrete*) ($\lambda i\ \omega.\ X'\ i\ (f\ \omega)$) I
shows *prob-space.indep-vars* (*measure-pmf* (*map-pmf f p*)) (λ-. *discrete*) X' I
⟨*proof*⟩

lemma *k-wise-indep-vars-map-pmf*:
assumes *prob-space.k-wise-indep-vars* (*measure-pmf p*) k (λ-. *discrete*) ($\lambda i\ \omega.\ X'\ i\ (f\ \omega)$) I
shows *prob-space.k-wise-indep-vars* (*measure-pmf* (*map-pmf f p*)) k (λ-. *discrete*) X' I
⟨*proof*⟩

lemma (**in** *prob-space*) *k-wise-indep-subset*:
assumes $J \subseteq I$
assumes *k-wise-indep-vars* k M' X' I
shows *k-wise-indep-vars* k M' X' J
⟨*proof*⟩

lemma (**in** *prob-space*) *k-wise-indep-vars-reindex*:
  assumes *inj-on f I*
  assumes *k-wise-indep-vars k M' X'* (*f ' I*)
  shows *k-wise-indep-vars k* ($M' \circ f$) ($\lambda k\ \omega.\ X'$ (*f k*) ω) *I*
⟨*proof*⟩

definition *GF* :: *nat* ⇒ *int set list set ring*
  where *GF n* = (*SOME F. finite-field F* ∧ *order F* = *n*)

definition *is-prime-power* :: *nat* ⇒ *bool*
  where *is-prime-power n* ⟷ (∃ *p k. Factorial-Ring.prime p* ∧ *k* > *0* ∧ *n* = *p^k*)

lemma
  assumes *is-prime-power n*
  shows *GF*: *finite-field* (*GF n*) *order* (*GF n*) = *n*
⟨*proof*⟩

lemma *is-prime-power*: *Factorial-Ring.prime p* ⟹ *k* > *0* ⟹ *is-prime-power* (*p^k*)
  ⟨*proof*⟩

definition *split-prime-power* :: *nat* ⇒ (*nat* × *nat*)
  where *split-prime-power n* = (*THE* (*p*, *k*). *p^k* = *n* ∧ *Factorial-Ring.prime p* ∧ *k* > *0*)

lemma *split-prime-power*:
  assumes *Factorial-Ring.prime p*
  assumes *k* > *0*
  shows *split-prime-power* (*p^k*) = (*p*,*k*)
⟨*proof*⟩

definition $\mathcal{H}$ :: *nat* ⇒ *nat* ⇒ *'a sample-space* ⇒ (*nat* ⇒ *'a*) *sample-space*
  where $\mathcal{H}$ *k d R* = (
    let (*p*,*n*) = *split-prime-power* (*size R*);
      m = (*LEAST j. d* ≤ *p^j* ∧ *j* ≥ *n*);
      f = *from-nat-into* (*carrier* (*GF* (*p^m*)));
      f' = *to-nat-on* (*carrier* (*GF* (*p^m*)));
      g = *from-nat-into* (*bounded-degree-polynomials* (*GF* (*p^m*)) *k*) *in*
    (| *size* = *p^*(*m***k*), *select* = (λ*i x. select R* ((*f'* (*ring.hash* (*GF* (*p^m*)) (*f x*) (*g i*))) *mod p^n*))|))

locale *hash-sample-space* =
  fixes *k d p n* :: *nat*
  fixes *R* :: *'a sample-space*
  assumes *p-prime*: *Factorial-Ring.prime p*
  assumes *size-R*: *size R* = *p ^ n*
  assumes *k-gt-0*: *k* > *0*
  assumes *n-gt-0*: *n* > *0*
begin

abbreviation *S* **where** *S* ≡ $\mathcal{H}$ *k d R*

lemma *p-n-def*: (*p*,*n*) = *split-prime-power* (*size R*)
  ⟨*proof*⟩

definition *m* **where** *m* = (*LEAST j. d* ≤ *p^j* ∧ *j* ≥ *n*)
definition *f* **where** *f* = *from-nat-into* (*carrier* (*GF* (*p^m*)))
definition *f'* **where** *f'* = *to-nat-on* (*carrier* (*GF* (*p^m*)))

lemma *n-lt-m*: *n* ≤ *m* **and** *d-lt-p-m*: *d* ≤ *p^m*
⟨*proof*⟩

lemma
is-field: *finite-field* (*GF* (*p*^*m*)) (**is** *?A*) **and**
field-order: *order* (*GF*(*p*^*m*)) = *p*^*m* (**is** *?B*)
⟨*proof*⟩

interpretation *cw*: *carter-wegman-hash-family GF* (*p*^*m*) *k*
⟨*proof*⟩

lemma *field-size*: *cw.field-size* = *p*^*m*
⟨*proof*⟩

lemma *f-bij*: *bij-betw f* {..<*p*^*m*} (*carrier* (*GF* (*p*^*m*)))
⟨*proof*⟩

definition *g* **where** *g* = *from-nat-into cw.space*

lemma *p-n-gt-0*: *p*^*n* > *0*
⟨*proof*⟩

lemma *p-m-gt-0*: *p*^*m* > *0*
⟨*proof*⟩

lemma *S-eq*: *S* = (| *size* = *p*^(*m*∗*k*), *sample-space-select* = (λ *i x*. *select R* (*f'* (*cw.hash* (*f x*) (*g i*)) *mod p*^*n*)) |)
⟨*proof*⟩

lemma $\mathcal{H}$-*size*: *size S* > *0*
⟨*proof*⟩

lemma *sample-space*: *sample-space S*
⟨*proof*⟩

lemma *sample-space-R*: *sample-space R*
⟨*proof*⟩

lemma *range*: *range* (*select S i*) ⊆ *sample-set R*
⟨*proof*⟩

lemma *cw-space*: *map-pmf g* (*pmf-of-set* {..<*p*^(*m*∗*k*)}) = *pmf-of-set cw.space*
⟨*proof*⟩

lemma *single*:
assumes *x* < *d*
shows *map-pmf* (λω. ω *x*) (*sample-pmf S*) = *sample-pmf R* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *indep*:
prob-space.k-wise-indep-vars (*sample-pmf S*) *k* (λ-. *discrete*) (λ*i* ω. ω *i*) {..<*d*}
⟨*proof*⟩

lemma *size*:
fixes *m* :: *nat*
assumes *d* > *0*
defines *m-altdef*: *m* ≡ *max n* (*nat* ⌈*log p d*⌉)
shows *size S* = *p*^(*m*∗*k*)
⟨*proof*⟩

end

Sample space with a geometric distribution

fun *count-zeros* :: *nat* ⇒ *nat* ⇒ *nat* **where**
  count-zeros 0 k = *0* |
  count-zeros (*Suc n*) *k* = (*if odd k then 0 else 1* + *count-zeros n* (*k div 2*))

lemma *count-zeros-iff*: $j \le n \Longrightarrow$ *count-zeros n k* $\ge j \longleftrightarrow 2\hat{}j$ *dvd k*
⟨*proof*⟩

lemma *count-zeros-max*:
  count-zeros n k $\le n$
  ⟨*proof*⟩

definition $\mathcal{G}$ :: *nat* ⇒ *nat sample-space* **where**
  $\mathcal{G}$ *n* = (| *size* = $2\hat{}n$, *sample-space-select* = *count-zeros n* |)

lemma $\mathcal{G}$*-sample-space*[*simp*]: *sample-space* ($\mathcal{G}$ *n*)
  ⟨*proof*⟩

lemma $\mathcal{G}$*-range*: *sample-set* ($\mathcal{G}$ *n*) ⊆ {..*n*}
  ⟨*proof*⟩

lemma $\mathcal{G}$*-prob*:
  measure (*sample-pmf* ($\mathcal{G}$ *n*)) {ω. ω ≥ *j*} = *of-bool* (*j* ≤ *n*) / $2\hat{}j$ (**is** *?L* = *?R*)
⟨*proof*⟩

lemma $\mathcal{G}$*-prob-single*:
  measure (*sample-pmf* ($\mathcal{G}$ *n*)) {*j*} ≤ *1* / $2\hat{}j$ (**is** *?L* ≤ *?R*)
⟨*proof*⟩

3.4 Expander Walks

definition $\mathcal{E}$ :: *nat* ⇒ *real* ⇒ *'a sample-space* ⇒ (*nat* ⇒ *'a*) *sample-space*
  where $\mathcal{E}$ *l* Λ *S* = (*let e* = *see-standard* (*size S*) Λ *in*
    (| *size* = *see-size e* ∗ *see-degree e*^(*l*−*1*),
      sample-space-select = (λ*i j*. *select S* (*see-sample-walk e* (*l*−*1*) *i* ! *j*)) |))

locale *expander-sample-space* =
  fixes *l* :: *nat*
  fixes Λ :: *real*
  fixes *S* :: *'a sample-space*
  assumes *l-gt-0*: *l* > *0*
  assumes Λ*-gt-0*: Λ > *0*
  assumes *sample-space-S*: *sample-space S*
begin

definition *e* **where** *e* = *see-standard* (*size S*) Λ

lemma *size-S-gt-0*: *size S* > *0*
  ⟨*proof*⟩

lemma $\mathcal{E}$*-alt*: ($\mathcal{E}$ *l* Λ *S*) =
  (| *size* = *see-size e* ∗ *see-degree e*^(*l*−*1*),
    sample-space-select = (λ*i j*. *select S* (*see-sample-walk e* (*l*−*1*) *i* ! *j*)) |)
  ⟨*proof*⟩

lemmas *see-standard* = *see-standard*[*OF size-S-gt-0* Λ*-gt-0*]

sublocale *E*: *regular-graph graph-of e*
⟨*proof*⟩

lemma *e-deg-gt-0*: *see-degree e* > *0*
⟨*proof*⟩

lemma *e-size-gt-0*: *see-size e* > *0*
⟨*proof*⟩

lemma *sample-space*: *sample-space* ($\mathcal{E}$ *l* Λ *S*)
⟨*proof*⟩

lemma *range*: *select* ($\mathcal{E}$ *l* Λ *S*) *i j* ∈ *sample-set S*
⟨*proof*⟩

lemma *sample-set*: *sample-set* ($\mathcal{E}$ *l* Λ *S*) ⊆ (*UNIV* → *sample-set S*)
⟨*proof*⟩

lemma *walks*:
defines *R* ≡ *map-pmf* (λ*xs i*. *select S* (*xs* ! *i*)) (*pmf-of-multiset* (*walks* (*graph-of e*) *l*))
shows *sample-pmf* ($\mathcal{E}$ *l* Λ *S*) = *R*
⟨*proof*⟩

lemma *uniform-property*:
assumes *i* < *l*
shows *map-pmf* (λ*w*. *w i*) ($\mathcal{E}$ *l* Λ *S*) = *sample-pmf S* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *size*:
size ($\mathcal{E}$ *l* Λ *S*) = *size S* ∗ (*16* ^ ((*l*−*1*) ∗ *nat* ⌈*ln* Λ / *ln* (*19* / *20*)⌉)) (**is** *?L* = *?R*)
⟨*proof*⟩

end

end

4 Balls and Bins

The balls and bins model describes the probability space of throwing r balls into b bins. This section derives the expected number of bins hit by at least one ball, as well as the variance in the case that each ball is thrown independently. Further, using an approximation argument it is then possible to derive bounds for the same measures in the case when the balls are being thrown only *k*-wise independently. The proofs follow the reasoning described in [8, §A.1] but improve on the constants, as well as constraints.

theory *Distributed-Distinct-Elements-Balls-and-Bins*
imports
Distributed-Distinct-Elements-Preliminary
Discrete-Summation.Factorials
HOL−*Combinatorics.Stirling*
HOL−*Computational-Algebra.Polynomial*
HOL−*Decision-Procs.Approximation*
begin

hide-fact *Henstock-Kurzweil-Integration.integral-sum*
hide-fact *Henstock-Kurzweil-Integration.integral-mult-right*

hide-fact *Henstock-Kurzweil-Integration.integral-nonneg*
hide-fact *Henstock-Kurzweil-Integration.integral-cong*
unbundle *intro-cong-syntax*

lemma *sum-power-distrib*:
fixes $f :: 'a \Rightarrow real$
assumes *finite R*
shows $(\sum i{\in}R.\ f\ i)\ \hat{}\ s = (\sum xs \mid set\ xs \subseteq R \wedge length\ xs = s.\ (\prod x \leftarrow xs.\ f\ x))$
⟨*proof*⟩

lemma *sum-telescope-eq*:
fixes $f :: nat \Rightarrow 'a :: \{comm\text{-}ring\text{-}1\}$
shows $(\sum k{\in}\{Suc\ m..n\}.\ f\ k - f\ (k - 1)) = of\text{-}bool(m \leq n) *(f\ n - f\ m)$
⟨*proof*⟩

An improved version of *diff-power-eq-sum*.

lemma *power-diff-sum*:
fixes $a\ b :: 'a :: \{comm\text{-}ring\text{-}1,power\}$
shows $a\hat{}k - b\hat{}k = (a-b) * (\sum i = 0..<k.\ a\ \hat{}\ i * b\ \hat{}\ (k - 1 - i))$
⟨*proof*⟩

lemma *power-diff-est*:
assumes $(a :: real) \geq b$
assumes $b \geq 0$
shows $a\hat{}k - b\hat{}k \leq (a-b) * k * a\hat{}(k-1)$
⟨*proof*⟩

lemma *power-diff-est-2*:
assumes $(a :: real) \geq b$
assumes $b \geq 0$
shows $a\hat{}k - b\hat{}k \geq (a-b) * k * b\hat{}(k-1)$
⟨*proof*⟩

lemma *of-bool-prod*:
assumes *finite R*
shows $(\prod\ j \in R.\ of\text{-}bool(f\ j)) = (of\text{-}bool(\forall j \in R.\ f\ j) :: real)$
⟨*proof*⟩

Additional results about falling factorials:

lemma *ffact-nonneg*:
fixes $x :: real$
assumes $k - 1 \leq x$
shows $ffact\ k\ x \geq 0$
⟨*proof*⟩

lemma *ffact-pos*:
fixes $x :: real$
assumes $k - 1 < x$
shows $ffact\ k\ x > 0$
⟨*proof*⟩

lemma *ffact-mono*:
fixes $x\ y :: real$
assumes $k-1 \leq x\ x \leq y$
shows $ffact\ k\ x \leq ffact\ k\ y$
⟨*proof*⟩

lemma *ffact-of-nat-nonneg*:

fixes x :: $'a$ :: {*comm-ring-1*, *linordered-nonzero-semiring*}
assumes $x \in \mathbb{N}$
shows *ffact* k $x \geq 0$
⟨*proof*⟩

lemma *ffact-suc-diff*:
fixes x :: ($'a$ :: *comm-ring-1*)
shows *ffact* k x − *ffact* k $(x-1)$ = *of-nat* k ∗ *ffact* $(k-1)$ $(x-1)$ (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *ffact-bound*:
ffact k (n::*nat*) ≤ n^k
⟨*proof*⟩

lemma *fact-moment-binomial*:
fixes n :: *nat* **and** α :: *real*
assumes $\alpha \in \{0..1\}$
defines $p \equiv$ *binomial-pmf* n α
shows ($\int \omega$. *ffact* s (*real* ω) ∂p) = *ffact* s (*real* n) ∗ α^s (**is** *?L* = *?R*)
⟨*proof*⟩

The following describes polynomials of a given maximal degree as a subset of the functions, similar to the subsets $\mathbb{Z}$ or $\mathbb{Q}$ as subsets of larger number classes.

definition *Polynomials* ($\mathbb{P}$)
where *Polynomials* k = {f. $\exists p$. f = *poly* p ∧ *degree* $p \leq k$}

lemma *Polynomials-mono*:
assumes $s \leq t$
shows $\mathbb{P}$ $s \subseteq \mathbb{P}$ t
⟨*proof*⟩

lemma *Polynomials-addI*:
assumes $f \in \mathbb{P}$ k $g \in \mathbb{P}$ k
shows ($\lambda\omega$. f ω + g ω) $\in \mathbb{P}$ k
⟨*proof*⟩

lemma *Polynomials-diffI*:
fixes f g :: $'a$ :: *comm-ring* ⇒ $'a$
assumes $f \in \mathbb{P}$ k $g \in \mathbb{P}$ k
shows (λx. f x − g x) $\in \mathbb{P}$ k
⟨*proof*⟩

lemma *Polynomials-idI*:
(λx. x) ∈ ($\mathbb{P}$ *1* :: ($'a$::*comm-ring-1* ⇒ $'a$) *set*)
⟨*proof*⟩

lemma *Polynomials-constI*:
(λx. c) $\in \mathbb{P}$ k
⟨*proof*⟩

lemma *Polynomials-multI*:
fixes f g :: $'a$ :: {*comm-ring*} ⇒ $'a$
assumes $f \in \mathbb{P}$ s $g \in \mathbb{P}$ t
shows (λx. f x ∗ g x) $\in \mathbb{P}$ $(s+t)$
⟨*proof*⟩

lemma *Polynomials-composeI*:
fixes f g :: $'a$ :: {*comm-semiring-0*, *semiring-no-zero-divisors*} ⇒ $'a$

assumes $f \in \mathbb{P}\ s$ $g \in \mathbb{P}\ t$
shows $(\lambda x.\ f\ (g\ x)) \in \mathbb{P}\ (s*t)$
⟨*proof*⟩

lemma *Polynomials-const-left-multI*:
fixes $c :: 'a :: \{comm\text{-}ring\}$
assumes $f \in \mathbb{P}\ k$
shows $(\lambda x.\ c * f\ x) \in \mathbb{P}\ k$
⟨*proof*⟩

lemma *Polynomials-const-right-multI*:
fixes $c :: 'a :: \{comm\text{-}ring\}$
assumes $f \in \mathbb{P}\ k$
shows $(\lambda x.\ f\ x * c) \in \mathbb{P}\ k$
⟨*proof*⟩

lemma *Polynomials-const-divI*:
fixes $c :: 'a :: \{field\}$
assumes $f \in \mathbb{P}\ k$
shows $(\lambda x.\ f\ x\ /\ c) \in \mathbb{P}\ k$
⟨*proof*⟩

lemma *Polynomials-ffact*: $(\lambda x.\ ffact\ s\ (x - y)) \in (\mathbb{P}\ s :: ('a :: comm\text{-}ring\text{-}1 \Rightarrow 'a)\ set)$
⟨*proof*⟩

lemmas *Polynomials-intros* =
Polynomials-const-divI
Polynomials-composeI
Polynomials-const-left-multI
Polynomials-const-right-multI
Polynomials-multI
Polynomials-addI
Polynomials-diffI
Polynomials-idI
Polynomials-constI
Polynomials-ffact

definition C_2 :: *real* **where** $C_2 = 7.5$
definition C_3 :: *real* **where** $C_3 = 16$

A locale fixing the sets of balls and bins

locale *balls-and-bins-abs* =
fixes $R :: 'a\ set$ **and** $B :: 'b\ set$
assumes *fin-B*: *finite B* **and** *B-ne*: $B \neq \{\}$
assumes *fin-R*: *finite R*
begin

Independent balls and bins space:

definition Ω
where $\Omega = prod\text{-}pmf\ R\ (\lambda\text{-}.\ pmf\text{-}of\text{-}set\ B)$

lemma *set-pmf-Ω*: $set\text{-}pmf\ \Omega = R \rightarrow_E B$
⟨*proof*⟩

lemma *card-B-gt-0*: $card\ B > 0$
⟨*proof*⟩

lemma *card-B-ge-1*: $card\ B \geq 1$

⟨*proof*⟩

definition $Z\ j\ \omega = real\ (card\ \{i.\ i \in R \wedge \omega\ i = (j::'b)\})$
definition $Y\ \omega = real\ (card\ (\omega\ `\ R))$
definition $\mu = real\ (card\ B) * (1 - (1-1/real\ (card\ B))\hat{}card\ R)$

Factorial moments for the random variable describing the number of times a bin will be hit:

lemma *fact-moment-balls-and-bins*:
  assumes $J \subseteq B\ J \neq \{\}$
  shows $(\int \omega.\ \mathit{ffact}\ s\ (\sum j \in J.\ Z\ j\ \omega)\ \partial\Omega) =$
    $\mathit{ffact}\ s\ (real\ (card\ R)) * (real\ (card\ J)\ /\ real\ (card\ B))\hat{}s$
    (**is** $?L = ?R$)
⟨*proof*⟩

Expectation and variance for the number of distinct bins that are hit by at least one ball in the fully independent model. The result for the variance is improved by a factor of 4 w.r.t. the paper.

lemma
  shows *exp-balls-and-bins*: $measure\text{-}pmf.expectation\ \Omega\ Y = \mu$ (**is** $?AL = ?AR$)
    and *var-balls-and-bins*: $measure\text{-}pmf.variance\ \Omega\ Y \leq card\ R * (real\ (card\ R) - 1)\ /\ card\ B$
      (**is** $?BL \leq ?BR$)
⟨*proof*⟩

definition $lim\text{-}balls\text{-}and\text{-}bins\ k\ p = ($
   $prob\text{-}space.k\text{-}wise\text{-}indep\text{-}vars\ (measure\text{-}pmf\ p)\ k\ (\lambda\text{-}.\ discrete)\ (\lambda x\ \omega.\ \omega\ x)\ R \wedge$
  $(\forall x.\ x \in R \longrightarrow map\text{-}pmf\ (\lambda\omega.\ \omega\ x)\ p = pmf\text{-}of\text{-}set\ B))$

lemma *indep*:
  assumes $lim\text{-}balls\text{-}and\text{-}bins\ k\ p$
  shows $prob\text{-}space.k\text{-}wise\text{-}indep\text{-}vars\ (measure\text{-}pmf\ p)\ k\ (\lambda\text{-}.\ discrete)\ (\lambda x\ \omega.\ \omega\ x)\ R$
  ⟨*proof*⟩

lemma *ran*:
  assumes $lim\text{-}balls\text{-}and\text{-}bins\ k\ p\ x \in R$
  shows $map\text{-}pmf\ (\lambda\omega.\ \omega\ x)\ p = pmf\text{-}of\text{-}set\ B$
  ⟨*proof*⟩

lemma *Z-integrable*:
  fixes $f :: real \Rightarrow real$
  assumes $lim\text{-}balls\text{-}and\text{-}bins\ k\ p$
  shows $integrable\ p\ (\lambda\omega.\ f\ (Z\ i\ \omega)\ )$
  ⟨*proof*⟩

lemma *Z-any-integrable-2*:
 fixes $f :: real \Rightarrow real$
  assumes $lim\text{-}balls\text{-}and\text{-}bins\ k\ p$
  shows $integrable\ p\ (\lambda\omega.\ f\ (Z\ i\ \omega + Z\ j\ \omega))$
⟨*proof*⟩

lemma *hit-count-prod-exp*:
  assumes $j1 \in B\ j2 \in B\ s+t \leq k$
  assumes $lim\text{-}balls\text{-}and\text{-}bins\ k\ p$
  defines $L \equiv \{(xs,ys).\ set\ xs \subseteq R \wedge set\ ys \subseteq R \wedge$
   $(set\ xs \cap set\ ys = \{\} \vee j1 = j2) \wedge length\ xs = s \wedge length\ ys = t\}$
  shows $(\int \omega.\ Z\ j1\ \omega\hat{}s * Z\ j2\ \omega\hat{}t\ \partial p) =$
   $(\sum (xs,ys) \in L.\ (1/real\ (card\ B))\hat{}(card\ (set\ xs \cup set\ ys)))$

(**is** $?L = ?R$)
⟨*proof*⟩

lemma *hit-count-prod-pow-eq*:
assumes $i \in B$ $j \in B$
assumes *lim-balls-and-bins* k p
assumes *lim-balls-and-bins* k q
assumes $s+t \leq k$
shows $(\int \omega.\ (Z\ i\ \omega)\hat{}s * (Z\ j\ \omega)\hat{}t\ \partial p) = (\int \omega.\ (Z\ i\ \omega)\hat{}s * (Z\ j\ \omega)\hat{}t\ \partial q)$
⟨*proof*⟩

lemma *hit-count-sum-pow-eq*:
assumes $i \in B$ $j \in B$
assumes *lim-balls-and-bins* k p
assumes *lim-balls-and-bins* k q
assumes $s \leq k$
shows $(\int \omega.\ (Z\ i\ \omega + Z\ j\ \omega)\hat{}s\ \partial p) = (\int \omega.\ (Z\ i\ \omega + Z\ j\ \omega)\hat{}s\ \partial q)$
(**is** $?L = ?R$)
⟨*proof*⟩

lemma *hit-count-sum-poly-eq*:
assumes $i \in B$ $j \in B$
assumes *lim-balls-and-bins* k p
assumes *lim-balls-and-bins* k q
assumes $f \in \mathbb{P}\ k$
shows $(\int \omega.\ f\ (Z\ i\ \omega + Z\ j\ \omega)\ \partial p) = (\int \omega.\ f\ (Z\ i\ \omega + Z\ j\ \omega)\ \partial q)$
(**is** $?L = ?R$)
⟨*proof*⟩

lemma *hit-count-poly-eq*:
assumes $b \in B$
assumes *lim-balls-and-bins* k p
assumes *lim-balls-and-bins* k q
assumes $f \in \mathbb{P}\ k$
shows $(\int \omega.\ f\ (Z\ b\ \omega)\ \partial p) = (\int \omega.\ f\ (Z\ b\ \omega)\ \partial q)$ (**is** $?L = ?R$)
⟨*proof*⟩

lemma *lim-balls-and-bins-from-ind-balls-and-bins*:
lim-balls-and-bins k Ω
⟨*proof*⟩

lemma *hit-count-factorial-moments*:
assumes $a{:}j \in B$
assumes $s \leq k$
assumes *lim-balls-and-bins* k p
shows $(\int \omega.\ \mathit{ffact}\ s\ (Z\ j\ \omega)\ \partial p) = \mathit{ffact}\ s\ (\mathit{real}\ (\mathit{card}\ R)) * (1\ /\ \mathit{real}\ (\mathit{card}\ B))\hat{}s$
(**is** $?L = ?R$)
⟨*proof*⟩

lemma *hit-count-factorial-moments-2*:
assumes $a{:}i \in B$ $j \in B$
assumes $i \neq j$ $s \leq k$ $\mathit{card}\ R \leq \mathit{card}\ B$
assumes *lim-balls-and-bins* k p
shows $(\int \omega.\ \mathit{ffact}\ s\ (Z\ i\ \omega + Z\ j\ \omega)\ \partial p) \leq 2\hat{}s$
(**is** $?L \leq ?R$)
⟨*proof*⟩

lemma *balls-and-bins-approx-helper*:

fixes x :: *real*
assumes $x \geq 2$
assumes *real* $k \geq 5{*}x$ / *ln* x
shows $k \geq 2$
and $2\hat{}(k+3)$ / *fact* $k \leq (1/exp\ x)\hat{}2$
and 2 / *fact* $k \leq 1$ / (*exp* $1 *$ *exp* x)
⟨*proof*⟩

Bounds on the expectation and variance in the k-wise independent case. Here the indepedence assumption is improved by a factor of two compared to the result in the paper.

lemma
assumes *card* $R \leq$ *card* B
assumes $\bigwedge c$. *lim-balls-and-bins* $(k+1)$ $(p\ c)$
assumes $\varepsilon \in \{0{<}..1/exp(2)\}$
assumes $k \geq 5 *$ *ln* (*card* B / ε) / *ln* (*ln* (*card* B / ε))
shows
exp-approx: |*measure-pmf.expectation* (p *True*) Y − *measure-pmf.expectation* (p *False*) Y| ≤ $\varepsilon *$ *real* (*card* R) (**is** *?A*) **and**
var-approx: |*measure-pmf.variance* (p *True*) Y − *measure-pmf.variance* (p *False*) Y| ≤ ε^2 (**is** *?B*)
⟨*proof*⟩

lemma
assumes *card* $R \leq$ *card* B
assumes *lim-balls-and-bins* $(k+1)$ p
assumes $k \geq 7.5 *$ (*ln* (*card* B) + 2)
shows *exp-approx-2*: |*measure-pmf.expectation* p Y − μ| ≤ *card* R / *sqrt* (*card* B) (**is** *?AL* ≤ *?AR*)
and *var-approx-2*: *measure-pmf.variance* p Y ≤ *real* (*card* R)$\hat{}2$ / *card* B (**is** *?BL* ≤ *?BR*)
⟨*proof*⟩

lemma *devitation-bound*:
assumes *card* $R \leq$ *card* B
assumes *lim-balls-and-bins* k p
assumes *real* $k \geq C_2 *$ *ln* (*real* (*card* B)) + C_3
shows *measure* p {ω. |$Y\ \omega - \mu$| > $9 *$ *real* (*card* R) / *sqrt* (*real* (*card* B))} ≤ 1 / $2\hat{}6$ (**is** *?L* ≤ *?R*)
⟨*proof*⟩

end

unbundle *no-intro-cong-syntax*

end

5 Tail Bounds for Expander Walks

theory *Distributed-Distinct-Elements-Tail-Bounds*
imports
Distributed-Distinct-Elements-Preliminary
Expander-Graphs.Expander-Graphs-Definition
Expander-Graphs.Expander-Graphs-Walks
HOL−Decision-Procs.Approximation
Pseudorandom-Combinators
begin

This section introduces tail estimates for random walks in expander graphs, specific to the verification of this algorithm (in particular to two-stage expander graph sampling and obtained tail bounds for subgaussian random variables). They follow from the more fundamental results *regular-graph.kl-chernoff-property* and *regular-graph.uniform-property* which are verified in the AFP entry for expander graphs [11].

hide-fact *Henstock-Kurzweil-Integration.integral-sum*

unbundle *intro-cong-syntax*

lemma *x-ln-x-min*:
  assumes $x \geq (0::real)$
  shows $x * ln\ x \geq -exp\ (-1)$
⟨*proof*⟩

theorem (**in** *regular-graph*) *walk-tail-bound*:
  assumes $l > 0$
  assumes $S \subseteq verts\ G$
  defines $\mu \equiv real\ (card\ S)\ /\ card\ (verts\ G)$
  assumes $\gamma < 1\ \mu + \Lambda_a \leq \gamma$
  shows $measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ real\ (card\ \{i \in \{..<l\}.\ w\ !\ i \in S\}) \geq \gamma * l\}$
    $\leq exp\ (-\ real\ l * (\gamma * ln\ (1/(\mu+\Lambda_a)) - 2 * exp(-1)))$ (**is** $?L \leq ?R$)
⟨*proof*⟩

theorem (**in** *regular-graph*) *walk-tail-bound-2*:
  assumes $l > 0\ \Lambda_a \leq \Lambda\ \Lambda > 0$
  assumes $S \subseteq verts\ G$
  defines $\mu \equiv real\ (card\ S)\ /\ card\ (verts\ G)$
  assumes $\gamma < 1\ \mu + \Lambda \leq \gamma$
  shows $measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ real\ (card\ \{i \in \{..<l\}.\ w\ !\ i \in S\}) \geq \gamma * l\}$
    $\leq exp\ (-\ real\ l * (\gamma * ln\ (1/(\mu+\Lambda)) - 2 * exp(-1)))$ (**is** $?L \leq ?R$)
⟨*proof*⟩

lemma (**in** *expander-sample-space*) *tail-bound*:
  fixes T
  assumes $l > 0\ \Lambda > 0$
  defines $\mu \equiv measure\ (sample\text{-}pmf\ S)\ \{w.\ T\ w\}$
  assumes $\gamma < 1\ \mu + \Lambda \leq \gamma$
  shows $measure\ (\mathcal{E}\ l\ \Lambda\ S)\ \{w.\ real\ (card\ \{i \in \{..<l\}.\ T\ (w\ i)\}) \geq \gamma * l\}$
    $\leq exp\ (-\ real\ l * (\gamma * ln\ (1/(\mu+\Lambda)) - 2 * exp(-1)))$ (**is** $?L \leq ?R$)
⟨*proof*⟩

definition C_1 :: *real* **where** $C_1 = exp\ 2 + exp\ 3 + (exp\ 1 - 1)$

lemma (**in** *regular-graph*) *deviation-bound*:
  fixes $f :: 'a \Rightarrow real$
  assumes $l > 0$
  assumes $\Lambda_a \leq exp\ (-real\ l * ln\ (real\ l)\hat{\ }3)$
  assumes $\bigwedge x.\ x \geq 20 \Longrightarrow measure\ (pmf\text{-}of\text{-}set\ (verts\ G))\ \{v.\ f\ v \geq x\} \leq exp\ (-x * ln\ x\hat{\ }3)$
  shows $measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ (\sum i \leftarrow w.\ f\ i) \geq C_1 * l\} \leq exp\ (-\ real\ l)$
    (**is** $?L \leq ?R$)
⟨*proof*⟩

lemma (**in** *expander-sample-space*) *deviation-bound*:
  fixes $f :: 'a \Rightarrow real$
  assumes $l > 0$
  assumes $\Lambda \leq exp\ (-real\ l * ln\ (real\ l)\hat{\ }3)$
  assumes $\bigwedge x.\ x \geq 20 \Longrightarrow measure\ (sample\text{-}pmf\ S)\ \{v.\ f\ v \geq x\} \leq exp\ (-x * ln\ x\hat{\ }3)$
  shows $measure\ (\mathcal{E}\ l\ \Lambda\ S)\ \{\omega.\ (\sum i<l.\ f\ (\omega\ i)) \geq C_1 * l\} \leq exp\ (-\ real\ l)$ (**is** $?L \leq ?R$)

⟨*proof*⟩

unbundle *no-intro-cong-syntax*

end

6 Inner Algorithm

This section introduces the inner algorithm (as mentioned it is already a solution to the cardinality estimation with the caveat that, if ε is too small it requires to much space. The outer algorithm in Section 10 resolved this problem.

The algorithm makes use of the balls and bins model, more precisely, the fact that the number of hit bins can be used to estimate the number of balls thrown (even if there are collusions). I.e. it assigns each universe element to a bin using a k-wise independent hash function. Then it counts the number of bins hit.

This strategy however would only work if the number of balls is roughly equal to the number of bins, to remedy that the algorithm performs an adaptive sub-sampling strategy. This works by assigning each universe element a level (using a second hash function) with a geometric distribution. The algorithm then selects a level that is appropriate based on a rough estimate obtained using the maximum level in the bins.

To save space the algorithm drops information about small levels, whenever the space usage would be too high otherwise. This level will be called the cutoff-level. This is okey as long as the cutoff level is not larger than the sub-sampling threshold. A lot of the complexity in the proof is devoted to verifying that the cutoff-level will not cross it, it works by defining a third value s_M that is both an upper bound for the cutoff level and a lower bound for the subsampling threshold simultaneously with high probability.

theory *Distributed-Distinct-Elements-Inner-Algorithm*
  imports
    Pseudorandom-Combinators
    Distributed-Distinct-Elements-Preliminary
    Distributed-Distinct-Elements-Balls-and-Bins
    Distributed-Distinct-Elements-Tail-Bounds
    Prefix-Free-Code-Combinators.Prefix-Free-Code-Combinators
begin

unbundle *intro-cong-syntax*
hide-const *Abstract-Rewriting.restrict*

definition C_4 :: *real* **where** $C_4 = 3\hat{}2*2\hat{}23$
definition C_5 :: *int* **where** $C_5 = 33$
definition C_6 :: *real* **where** $C_6 = 4$
definition C_7 :: *nat* **where** $C_7 = 2\hat{}5$

locale *inner-algorithm* =
  fixes n :: *nat*
  fixes δ :: *real*
  fixes ε :: *real*
  assumes *n-gt-0*: $n > 0$
  assumes *δ-gt-0*: $\delta > 0$ **and** *δ-lt-1*: $\delta < 1$
  assumes *ε-gt-0*: $\varepsilon > 0$ **and** *ε-lt-1*: $\varepsilon < 1$
begin

definition *b-exp* **where** *b-exp* = *nat* $\lceil log\ 2\ (C_4\ /\ \varepsilon\hat{}2)\rceil$
definition b :: *nat* **where** $b = 2\hat{}b\text{-}exp$

definition *l* **where** $l = nat \lceil C_6 * ln\ (2/\ \delta)\rceil$
definition *k* **where** $k = nat \lceil C_2 * ln\ b + C_3\rceil$
definition Λ :: *real* **where** $\Lambda = min\ (1/16)\ (exp\ (-l * ln\ l\hat{}3))$
definition ϱ :: *real* $\Rightarrow$ *real* **where** $\varrho\ x = b * (1 - (1-1/b)\ powr\ x)$
definition ϱ*-inv* :: *real* $\Rightarrow$ *real* **where** $\varrho\text{-}inv\ x = ln\ (1-x/b)\ /\ ln\ (1-1/b)$

lemma *l-lbound*: $C_6 * ln\ (2\ /\ \delta) \leq l$
⟨*proof*⟩

lemma *k-min*: $C_2 * ln\ (real\ b) + C_3 \leq real\ k$
⟨*proof*⟩

lemma Λ*-gt-0*: $\Lambda > 0$
⟨*proof*⟩

lemma Λ*-le-1*: $\Lambda \leq 1$
⟨*proof*⟩

lemma *l-gt-0*: $l > 0$
⟨*proof*⟩

lemma *l-ubound*: $l \leq C_6 * ln(1\ /\ \delta) + C_6 * ln\ 2 + 1$
⟨*proof*⟩

lemma *b-exp-ge-26*: $b\text{-}exp \geq 26$
⟨*proof*⟩

lemma *b-min*: $b \geq 2\hat{}26$
⟨*proof*⟩

lemma *k-gt-0*: $k > 0$
⟨*proof*⟩

lemma *b-ne*: $\{..<b\} \neq \{\}$
⟨*proof*⟩

lemma *b-lower-bound*: $C_4\ /\ \varepsilon\hat{}2 \leq real\ b$
⟨*proof*⟩

definition *n-exp* **where** $n\text{-}exp = max\ (nat\ \lceil log\ 2\ n\rceil)\ 1$

lemma *n-exp-gt-0*: $n\text{-}exp > 0$
⟨*proof*⟩

abbreviation Ψ_1 **where** $\Psi_1 \equiv \mathcal{H}\ 2\ n\ (\mathcal{G}\ n\text{-}exp)$
abbreviation Ψ_2 **where** $\Psi_2 \equiv \mathcal{H}\ 2\ n\ [C_7 * b^2]_S$
abbreviation Ψ_3 **where** $\Psi_3 \equiv \mathcal{H}\ k\ (C_7 * b^2)\ [b]_S$

definition Ψ **where** $\Psi = \Psi_1 \times_S \Psi_2 \times_S \Psi_3$

abbreviation Ω **where** $\Omega \equiv \mathcal{E}\ l\ \Lambda\ \Psi$

type-synonym *state* = (*nat* $\Rightarrow$ *nat* $\Rightarrow$ *int*) $\times$ (*nat*)

fun *is-too-large* :: (*nat* $\Rightarrow$ *nat* $\Rightarrow$ *int*) $\Rightarrow$ *bool* **where**
is-too-large $B = ((\sum\ (i,j) \in \{..<l\} \times \{..<b\}.\ \lfloor log\ 2\ (max\ (B\ i\ j)\ (-1) + 2)\rfloor) > C_5 * b * l)$

fun *compress-step* :: *state* $\Rightarrow$ *state* **where**

compress-step $(B,q) = (\lambda\ i\ j.\ max\ (B\ i\ j - 1)\ (-1),\ q+1)$

function *compress* :: *state* ⇒ *state* **where**
compress (B,q) = (
if is-too-large B
then (*compress* (*compress-step* (B,q)))
else (B,q))
⟨*proof*⟩

fun *compress-termination* :: *state* ⇒ *nat* **where**
compress-termination $(B,q) = (\sum\ (i,j) \in \{..<l\} \times \{..<b\}.\ nat\ (B\ i\ j + 1))$

lemma *compress-termination*:
assumes *is-too-large B*
shows *compress-termination* (*compress-step* (B,q)) < *compress-termination* (B,q)
⟨*proof*⟩

termination *compress*
⟨*proof*⟩

fun *merge1* :: *state* ⇒ *state* ⇒ *state* **where**
merge1 $(B1,q_1)\ (B2,\ q_2)$ = (
let $q = max\ q_1\ q_2$ *in* $(\lambda\ i\ j.\ max\ (B1\ i\ j + q_1 - q)\ (B2\ i\ j + q_2 - q),\ q))$

fun *merge* :: *state* ⇒ *state* ⇒ *state* **where**
merge x y = *compress* (*merge1 x y*)

type-synonym *seed* = *nat* ⇒ (*nat* ⇒ *nat*) × (*nat* ⇒ *nat*) × (*nat* ⇒ *nat*)

fun *single1* :: *seed* ⇒ *nat* ⇒ *state* **where**
single1 ω x = (λ *i j*.
let $(f,g,h) = \omega\ i$ *in* (
if $h\ (g\ x) = j \wedge i < l$ *then int* $(f\ x)$ *else* $(-1)),\ 0)$

fun *single* :: *seed* ⇒ *nat* ⇒ *state* **where**
single ω x = *compress* (*single1* ω x)

fun *estimate1* :: *state* ⇒ *nat* ⇒ *real* **where**
estimate1 (B,q) i = (
let $s = max\ 0\ (Max\ ((B\ i)\ `\ \{..<b\}) + q - \lfloor log\ 2\ b \rfloor + 9)$;
$p = card\ \{\ j.\ j \in \{..<b\} \wedge B\ i\ j + q \geq s\ \}$ *in*
2 powr $s * ln\ (1-p/b)\ /\ ln(1-1/b))$

fun *estimate* :: *state* ⇒ *real* **where**
estimate x = *median l* (*estimate1 x*)

6.1 History Independence

fun τ_0 :: ((*nat* ⇒ *nat*) × (*nat* ⇒ *nat*) × (*nat* ⇒ *nat*)) ⇒ *nat set* ⇒ *nat* ⇒ *int*
where $\tau_0\ (f,g,h)\ A\ j = Max\ (\{\ int\ (f\ a)\ |\ a\ .\ a \in A \wedge h\ (g\ a) = j\ \} \cup \{-1\})$

definition τ_1 :: ((*nat* ⇒ *nat*) × (*nat* ⇒ *nat*) × (*nat* ⇒ *nat*)) ⇒ *nat set* ⇒ *nat* ⇒ *nat* ⇒ *int*
where $\tau_1\ \psi\ A\ q\ j = max\ (\tau_0\ \psi\ A\ j - q)\ (-1)$

definition τ_2 :: *seed* ⇒ *nat set* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int*
where $\tau_2\ \omega\ A\ q\ i\ j$ = (*if* $i < l$ *then* $\tau_1\ (\omega\ i)\ A\ q\ j$ *else* $(-1))$

definition τ_3 :: *seed* ⇒ *nat set* ⇒ *nat* ⇒ *state*

where $\tau_3\ \omega\ A\ q = (\tau_2\ \omega\ A\ q,\ q)$

definition q :: *seed* ⇒ *nat set* ⇒ *nat*
where $q\ \omega\ A = (LEAST\ q\ .\ \neg(\textit{is-too-large}\ (\tau_2\ \omega\ A\ q)))$

definition τ :: *seed* ⇒ *nat set* ⇒ *state*
where $\tau\ \omega\ A = \tau_3\ \omega\ A\ (q\ \omega\ A)$

lemma τ_2*-step*: $\tau_2\ \omega\ A\ (x{+}y) = (\lambda i\ j.\ max\ (\tau_2\ \omega\ A\ x\ i\ j - y)\ (-\ 1))$
⟨*proof*⟩

lemma τ_3*-step*: *compress-step* $(\tau_3\ \omega\ A\ x) = \tau_3\ \omega\ A\ (x{+}1)$
⟨*proof*⟩

sublocale Ψ_1: *hash-sample-space 2 n 2 n-exp* $\mathcal{G}$ *n-exp*
⟨*proof*⟩

sublocale Ψ_2: *hash-sample-space 2 n 2 5 + b-exp*∗*2* $[(C_7 * b^2)]_S$
⟨*proof*⟩

sublocale Ψ_3: *hash-sample-space k* $C_7 * b^2$ *2 b-exp* $[b]_S$
⟨*proof*⟩

lemma *sample-pmf-*Ψ: *sample-pmf* Ψ = *pair-pmf* Ψ_1 (*pair-pmf* Ψ_2 Ψ_3)
⟨*proof*⟩

lemma *sample-set-*Ψ:
sample-set Ψ = *sample-set* Ψ_1 × *sample-set* Ψ_2 × *sample-set* Ψ_3
⟨*proof*⟩

lemma *sample-space-*Ψ: *sample-space* Ψ
⟨*proof*⟩

lemma *f-range*:
assumes $(f,g,h) \in$ *sample-set* Ψ
shows $f\ x \leq$ *n-exp*
⟨*proof*⟩

lemma *g-range-1*:
assumes $g \in$ *sample-set* Ψ_2
shows $g\ x < C_7 * b\hat{}2$
⟨*proof*⟩

lemma *h-range-1*:
assumes $h \in$ *sample-set* Ψ_3
shows $h\ x < b$
⟨*proof*⟩

lemma *g-range*:
assumes $(f,g,h) \in$ *sample-set* Ψ
shows $g\ x < C_7 * b\hat{}2$
⟨*proof*⟩

lemma *h-range*:
assumes $(f,g,h) \in$ *sample-set* Ψ
shows $h\ x < b$
⟨*proof*⟩

lemma *fin-f*:
  assumes $(f,g,h) \in$ *sample-set* Ψ
  shows *finite* { *int* (*f a*) | *a*. *P a* } (**is** *finite ?M*)
⟨*proof*⟩

lemma *Max-int-range*: $x \leq (y::int) \Longrightarrow Max\ \{x..y\} = y$
  ⟨*proof*⟩

sublocale Ω: *expander-sample-space* $l\ \Lambda\ \Psi$
  ⟨*proof*⟩

lemma *max-q-1*:
  assumes $\omega \in$ *sample-set* Ω
  shows $\tau_2\ \omega\ A\ (nat\ \lceil log\ 2\ n\rceil{+}2)\ i\ j = (-1)$
⟨*proof*⟩

lemma *max-q-2*:
  assumes $\omega \in$ *sample-set* Ω
  shows $\neg$ (*is-too-large* $(\tau_2\ \omega\ A\ (nat\ \lceil log\ 2\ n\rceil{+}2))$)
  ⟨*proof*⟩

lemma *max-s-3*:
  assumes $\omega \in$ *sample-set* Ω
  shows $q\ \omega\ A \leq (nat\ \lceil log\ 2\ n\rceil{+}2)$
  ⟨*proof*⟩

lemma *max-mono*: $x \leq (y::'a::linorder) \Longrightarrow max\ x\ z \leq max\ y\ z$
  ⟨*proof*⟩

lemma *max-mono-2*: $y \leq (z::'a::linorder) \Longrightarrow max\ x\ y \leq max\ x\ z$
  ⟨*proof*⟩

lemma τ_0*-mono*:
  assumes $\psi \in$ *sample-set* Ψ
  assumes $A \subseteq B$
  shows $\tau_0\ \psi\ A\ j \leq \tau_0\ \psi\ B\ j$
⟨*proof*⟩

lemma τ_2*-mono*:
  assumes $\omega \in$ *sample-set* Ω
  assumes $A \subseteq B$
  shows $\tau_2\ \omega\ A\ x\ i\ j \leq \tau_2\ \omega\ B\ x\ i\ j$
⟨*proof*⟩

lemma *is-too-large-antimono*:
  assumes $\omega \in$ *sample-set* Ω
  assumes $A \subseteq B$
  assumes *is-too-large* $(\tau_2\ \omega\ A\ x)$
  shows *is-too-large* $(\tau_2\ \omega\ B\ x)$
⟨*proof*⟩

lemma *q-compact*:
  assumes $\omega \in$ *sample-set* Ω
  shows $\neg$ (*is-too-large* $(\tau_2\ \omega\ A\ (q\ \omega\ A))$)
  ⟨*proof*⟩

lemma *q-mono*:
  assumes $\omega \in$ *sample-set* Ω

assumes $A \subseteq B$
shows $q\ \omega\ A \leq q\ \omega\ B$
⟨*proof*⟩

lemma *lt-s-too-large*: $x < q\ \omega\ A \Longrightarrow$ *is-too-large* $(\tau_2\ \omega\ A\ x)$
⟨*proof*⟩

lemma *compress-result-1*:
assumes $\omega \in$ *sample-set* Ω
shows *compress* $(\tau_3\ \omega\ A\ (q\ \omega\ A - i)) = \tau\ \omega\ A$
⟨*proof*⟩

lemma *compress-result*:
assumes $\omega \in$ *sample-set* Ω
assumes $x \leq q\ \omega\ A$
shows *compress* $(\tau_3\ \omega\ A\ x) = \tau\ \omega\ A$
⟨*proof*⟩

lemma τ_0*-merge*:
assumes $(f,g,h) \in$ *sample-set* Ψ
shows $\tau_0\ (f,g,h)\ (A \cup B)\ j = max\ (\tau_0\ (f,g,h)\ A\ j)\ (\tau_0\ (f,g,h)\ B\ j)$ (**is** *?L* = *?R*)
⟨*proof*⟩

lemma τ_2*-merge*:
assumes $\omega \in$ *sample-set* Ω
shows $\tau_2\ \omega\ (A \cup B)\ x\ i\ j = max\ (\tau_2\ \omega\ A\ x\ i\ j)\ (\tau_2\ \omega\ B\ x\ i\ j)$
⟨*proof*⟩

lemma *merge1-result*:
assumes $\omega \in$ *sample-set* Ω
shows *merge1* $(\tau\ \omega\ A)\ (\tau\ \omega\ B) = \tau_3\ \omega\ (A \cup B)\ (max\ (q\ \omega\ A)\ (q\ \omega\ B))$
⟨*proof*⟩

lemma *merge-result*:
assumes $\omega \in$ *sample-set* Ω
shows *merge* $(\tau\ \omega\ A)\ (\tau\ \omega\ B) = \tau\ \omega\ (A \cup B)$ (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *single1-result*: *single1* $\omega\ x = \tau_3\ \omega\ \{x\}\ 0$
⟨*proof*⟩

lemma *single-result*:
assumes $\omega \in$ *sample-set* Ω
shows *single* $\omega\ x = \tau\ \omega\ \{x\}$ (**is** *?L* = *?R*)
⟨*proof*⟩

6.2 Encoding states of the inner algorithm

definition *is-state-table* :: $(nat \times nat \Rightarrow int) \Rightarrow bool$ **where**
is-state-table $g = (range\ g \subseteq \{-1..\} \wedge g\ `\ (-(\{..<l\} \times \{..<b\})) \subseteq \{-1\})$

Encoding for state table values:

definition V_e :: *int encoding*
where $V_e\ x = (if\ x \geq -1\ then\ N_e\ (nat\ (x+1))\ else\ None)$

Encoding for state table:

definition $T_e{}'$ :: $(nat \times nat \Rightarrow int)$ *encoding* **where**
$T_e{}'\ g = ($

if is-state-table g
then (*List.product* [*0..<l*] [*0..<b*] $\rightarrow_e$ V_e) (*restrict g* ({..<*l*}×{..<*b*}))
else None)

definition T_e :: (*nat* ⇒ *nat* ⇒ *int*) *encoding*
where T_e *f* = $T_e{}'$ (*case-prod f*)

definition *encode-state* :: *state encoding*
where *encode-state* = T_e $\times_e$ Nb_e (*nat* ⌈*log 2 n*⌉+*3*)

lemma *inj-on-restrict*:
assumes *B* ⊆ {*f*. *f* ‘ (− *A*) ⊆ {*c*}}
shows *inj-on* (λ*x*. *restrict x A*) *B*
⟨*proof*⟩

lemma *encode-state*: *is-encoding encode-state*
⟨*proof*⟩

lemma *state-bit-count*:
assumes ω ∈ *sample-set* Ω
shows *bit-count* (*encode-state* (τ ω *A*)) ≤ *2^36* ∗ (*ln*(*1*/δ)+*1*)/ ε^*2* + *log 2* (*log 2 n* + *3*)
(**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *random-bit-count*:
size Ω ≤ *2 powr* (*4* ∗ *log 2 n* + *48* ∗ (*log 2* (*1* / ε) + *16*)^*2* + (*55* + *60* ∗ *ln* (*1* / δ))^*3*)
(**is** *?L* ≤ *?R*)
⟨*proof*⟩

end

unbundle *no-intro-cong-syntax*

end

7 Accuracy without cutoff

This section verifies that each of the l estimate have the required accuracy with high probability assuming that there was no cut-off, i.e., that $s = 0$. Section 9 will then show that this remains true as long as the cut-off is below $t\ f$ the subsampling threshold.

theory *Distributed-Distinct-Elements-Accuracy-Without-Cutoff*
imports
Distributed-Distinct-Elements-Inner-Algorithm
Distributed-Distinct-Elements-Balls-and-Bins
begin

no-notation *Polynomials.var* (X_1)

locale *inner-algorithm-fix-A* = *inner-algorithm* +
fixes *A*
assumes *A-range*: *A* ⊆ {..<*n*}
assumes *A-nonempty*: {} ≠ *A*
begin

definition *X* :: *nat* **where** *X* = *card A*

definition *q-max* **where** *q-max* = *nat* (⌈*log 2 X*⌉ − *b-exp*)

definition t :: $(nat \Rightarrow nat) \Rightarrow int$
  where $t\ f = int\ (Max\ (f\ `\ A)) - b\text{-}exp + 9$

definition s :: $(nat \Rightarrow nat) \Rightarrow nat$
  where $s\ f = nat\ (t\ f)$

definition R :: $(nat \Rightarrow nat) \Rightarrow nat\ set$
  where $R\ f = \{a.\ a \in A \wedge f\ a \geq s\ f\}$

definition r :: $nat \Rightarrow (nat \Rightarrow nat) \Rightarrow nat$
  where $r\ x\ f = card\ \{a.\ a \in A \wedge f\ a \geq x\}$

definition p **where** $p = (\lambda(f,g,h).\ card\ \{j\in \{..<b\}.\ \tau_1\ (f,g,h)\ A\ 0\ j \geq s\ f\})$

definition Y **where** $Y = (\lambda(f,g,h).\ 2\ \hat{}\ s\ f * \varrho\text{-}inv\ (p\ (f,g,h)))$

lemma *fin-A*: *finite A*
  ⟨*proof*⟩

lemma *X-le-n*: $X \leq n$
⟨*proof*⟩

lemma *X-ge-1*: $X \geq 1$
  ⟨*proof*⟩

lemma *of-bool-square*: $(of\text{-}bool\ x)^2 = ((of\text{-}bool\ x)::real)$
  ⟨*proof*⟩

lemma *r-eq*: $r\ x\ f = (\sum\ a \in A.(\ of\text{-}bool(\ x \leq f\ a) :: real))$
  ⟨*proof*⟩

lemma
  shows
    r-exp: $(\int \omega.\ real\ (r\ x\ \omega)\ \partial\ \Psi_1) = real\ X * (of\text{-}bool\ (x \leq max\ (nat\ \lceil log\ 2\ n\rceil)\ 1)\ /\ 2\hat{}x)$ **and**
    r-var: $measure\text{-}pmf.variance\ \Psi_1\ (\lambda\omega.\ real\ (r\ x\ \omega)) \leq (\int \omega.\ real\ (r\ x\ \omega)\ \partial\ \Psi_1)$
⟨*proof*⟩

definition E_1 **where** $E_1 = (\lambda(f,g,h).\ 2\ powr\ (-t\ f) * X \in \{b/2\hat{}16..b/2\})$

lemma *t-low*:
  $measure\ \Psi_1\ \{f.\ of\text{-}int\ (t\ f) < log\ 2\ (real\ X) + 1 - b\text{-}exp\} \leq 1/2\hat{}7$ (**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *t-high*:
  $measure\ \Psi_1\ \{f.\ of\text{-}int\ (t\ f) > log\ 2\ (real\ X) + 16 - b\text{-}exp\} \leq 1/2\hat{}7$ (**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *e-1*: $measure\ \Psi\ \{\psi.\ \neg E_1\ \psi\} \leq 1/2\hat{}6$
⟨*proof*⟩

definition E_2 **where** $E_2 = (\lambda(f,g,h).\ |card\ (R\ f) - X\ /\ 2\hat{}(s\ f)| \leq \varepsilon/3 * X\ /\ 2\hat{}(s\ f))$

lemma *e-2*: $measure\ \Psi\ \{\psi.\ E_1\ \psi \wedge \neg E_2\ \psi\} \leq 1/2\hat{}6$ (**is** *?L* ≤ *?R*)
⟨*proof*⟩

definition E_3 **where** $E_3 = (\lambda(f,g,h).\ inj\text{-}on\ g\ (R\ f))$

lemma *R-bound*:
  fixes *f g h*
  assumes E_1 (*f*,*g*,*h*)
  assumes E_2 (*f*,*g*,*h*)
  shows *card* (*R f*) $\leq$ *2/3* $*$ *b*
⟨*proof*⟩

lemma *e-3*: *measure* Ψ $\{\psi.\ E_1\ \psi \wedge E_2\ \psi \wedge \neg E_3\ \psi\} \leq 1/2\hat{}6$ (**is** *?L* $\leq$ *?R*)
⟨*proof*⟩

definition E_4 **where** $E_4 = (\lambda(f,g,h).\ |p\ (f,g,h) - \varrho\ (card\ (R\ f))| \leq\ \varepsilon/12 * card\ (R\ f))$

lemma *e-4-h*: $9\ /\ sqrt\ b \leq \varepsilon\ /\ 12$
⟨*proof*⟩

lemma *e-4*: *measure* Ψ $\{\psi.\ E_1\ \psi \wedge E_2\ \psi \wedge E_3\ \psi \wedge \neg E_4\ \psi\} \leq 1/2\hat{}6$ (**is** *?L* $\leq$ *?R*)
⟨*proof*⟩

lemma *ϱ-inverse*: $\varrho\text{-}inv\ (\varrho\ x) = x$
⟨*proof*⟩

lemma *rho-mono*:
  assumes $x \leq y$
  shows $\varrho\ x \leq \varrho\ y$
⟨*proof*⟩

lemma *rho-two-thirds*: $\varrho\ (2/3 * b) \leq 3/5 *b$
⟨*proof*⟩

definition $\varrho\text{-}inv'$ :: *real* $\Rightarrow$ *real*
  where $\varrho\text{-}inv'\ x = -1\ /\ (real\ b * (1-x\ /\ real\ b) * ln\ (1 - 1\ /\ real\ b))$

lemma *ϱ-inv′-bound*:
  assumes $x \geq 0$
  assumes $x \leq 59/90{*}b$
  shows $|\varrho\text{-}inv'\ x| \leq 4$
⟨*proof*⟩

lemma *ϱ-inv′*:
  fixes *x* :: *real*
  assumes $x < b$
  shows *DERIV* $\varrho\text{-}inv\ x$:> $\varrho\text{-}inv'\ x$
⟨*proof*⟩

lemma *accuracy-without-cutoff*:
  measure Ψ $\{(f,g,h).\ |Y\ (f,g,h) - real\ X| > \varepsilon * X \vee s\ f < q\text{-}max\} \leq 1/2\hat{}4$
  (**is** *?L* $\leq$ *?R*)
⟨*proof*⟩

end

end

8 Cutoff Level

This section verifies that the cutoff will be below *q-max* with high probability. The result will be needed in Section 9, where it is shown that the estimates will be accurate for any cutoff below *q-max*.

theory *Distributed-Distinct-Elements-Cutoff-Level*
imports
Distributed-Distinct-Elements-Accuracy-Without-Cutoff
Distributed-Distinct-Elements-Tail-Bounds
begin

hide-const *Quantum.Z*

unbundle *intro-cong-syntax*

lemma *mono-real-of-int*: *mono real-of-int*
⟨*proof*⟩

lemma *Max-le-Sum*:
fixes $f :: 'a \Rightarrow int$
assumes *finite A*
assumes $\bigwedge a.\ a \in A \Longrightarrow f\ a \geq 0$
shows $Max\ (insert\ 0\ (f\ `\ A)) \leq (\sum a \in A\ .f\ a)$ (**is** $?L \leq ?R$)
⟨*proof*⟩

context *inner-algorithm-fix-A*
begin

The following inequality is true for base e on the entire domain ($x > 0$). It is shown in *ln-add-one-self-le-self*. In the following it is established for base 2, where it holds for $x \geq 1$.

lemma *log-2-estimate*:
assumes $x \geq (1::real)$
shows $log\ 2\ (1+x) \leq x$
⟨*proof*⟩

lemma *cutoff-eq-7*:
$real\ X * 2\ powr\ (-real\ q\text{-}max)\ /\ b \leq 1$
⟨*proof*⟩

lemma *cutoff-eq-6*:
fixes k
assumes $a \in A$
shows $(\int f.\ real\text{-}of\text{-}int\ (max\ 0\ (int\ (f\ a) - int\ k))\ \partial\Psi_1) \leq 2\ powr\ (-real\ k)$ (**is** $?L \leq ?R$)
⟨*proof*⟩

lemma *cutoff-eq-5*:
assumes $x \geq (-1 :: real)$
shows $real\text{-}of\text{-}int\ \lfloor log\ 2\ (x+2) \rfloor \leq (real\ c+2) + max\ (x - 2\hat{}c)\ 0$ (**is** $?L \leq ?R$)
⟨*proof*⟩

lemma *cutoff-level*:
$measure\ \Omega\ \{\omega.\ q\ \omega\ A > q\text{-}max\} \leq \delta/2$ (**is** $?L \leq ?R$)
⟨*proof*⟩

end

unbundle *no-intro-cong-syntax*

end

9 Accuracy with cutoff

This section verifies that each of the l estimate have the required accuracy with high probability assuming as long as the cutoff is below *q-max*, generalizing the result from Section 7.

theory *Distributed-Distinct-Elements-Accuracy*
  imports
    Distributed-Distinct-Elements-Accuracy-Without-Cutoff
    Distributed-Distinct-Elements-Cutoff-Level
begin

unbundle *intro-cong-syntax*

lemma (**in** *semilattice-set*) *Union*:
  assumes $\mathit{finite}\ I\ I \neq \{\}$
  assumes $\bigwedge i.\ i \in I \Longrightarrow \mathit{finite}\ (Z\ i)$
  assumes $\bigwedge i.\ i \in I \Longrightarrow Z\ i \neq \{\}$
  shows $F\ (\bigcup\ (Z\ `\ I)) = F\ ((\lambda i.\ (F\ (Z\ i)))\ `\ I)$
  ⟨*proof*⟩

This is similar to the existing *hom-Max-commute* with the crucial difference that it works even if the function is a homomorphism between distinct lattices. An example application is $\mathit{Max}\ (\mathit{int}\ `\ A) = \mathit{int}\ (\mathit{Max}\ A)$.

lemma *hom-Max-commute'*:
  assumes $\mathit{finite}\ A\ A \neq \{\}$
  assumes $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow \mathit{max}\ (f\ x)\ (f\ y) = f\ (\mathit{max}\ x\ y)$
  shows $\mathit{Max}\ (f\ `\ A) = f\ (\mathit{Max}\ A)$
  ⟨*proof*⟩

context *inner-algorithm-fix-A*
begin

definition t_c
  where $t_c\ \psi\ \sigma = (\mathit{Max}\ ((\lambda j.\ \tau_1\ \psi\ A\ \sigma\ j + \sigma)\ `\ \{..<b\})) - \mathit{b\text{-}exp} + 9$

definition s_c
  where $s_c\ \psi\ \sigma = \mathit{nat}\ (t_c\ \psi\ \sigma)$

definition p_c
  where $p_c\ \psi\ \sigma = \mathit{card}\ \{j\in \{..<b\}.\ \tau_1\ \psi\ A\ \sigma\ j + \sigma \geq s_c\ \psi\ \sigma\}$

definition Y_c
  where $Y_c\ \psi\ \sigma = 2\ \hat{}\ s_c\ \psi\ \sigma * \varrho\text{-}\mathit{inv}\ (p_c\ \psi\ \sigma)$

lemma s_c*-eq-s*:
  assumes $(f,g,h) \in \mathit{sample\text{-}set}\ \Psi$
  assumes $\sigma \leq s\ f$
  shows $s_c\ (f,g,h)\ \sigma = s\ f$
⟨*proof*⟩

lemma p_c*-eq-p*:
  assumes $(f,g,h) \in \mathit{sample\text{-}set}\ \Psi$
  assumes $\sigma \leq s\ f$
  shows $p_c\ (f,g,h)\ \sigma = p\ (f,g,h)$
⟨*proof*⟩

lemma Y_c*-eq-Y*:

assumes $(f,g,h) \in$ *sample-set* Ψ
assumes $\sigma \leq s\ f$
shows $Y_c\ (f,g,h)\ \sigma = Y\ (f,g,h)$
⟨*proof*⟩

lemma *accuracy-single*: *measure* Ψ $\{\psi.\ \exists \sigma \leq$ *q-max*. $|Y_c\ \psi\ \sigma -$ *real* $X| > \varepsilon * X\} \leq 1/2\hat{}4$
(**is** $?L \leq ?R$)
⟨*proof*⟩

lemma *estimate1-eq*:
assumes $j < l$
shows *estimate1* $(\tau_2\ \omega\ A\ \sigma,\ \sigma)\ j = Y_c\ (\omega\ j)\ \sigma$ (**is** $?L = ?R$)
⟨*proof*⟩

lemma *estimate-result-1*:
measure Ω $\{\omega.\ (\exists \sigma \leq$*q-max*. $\varepsilon * X <$ |*estimate* $(\tau_2\ \omega\ A\ \sigma,\sigma) - X|)\ \} \leq \delta/2$ (**is** $?L \leq ?R$)
⟨*proof*⟩

theorem *estimate-result*:
measure Ω $\{\omega.$ |*estimate* $(\tau\ \omega\ A) - X| > \varepsilon * X\} \leq \delta$
(**is** $?L \leq ?R$)
⟨*proof*⟩

end

lemma (**in** *inner-algorithm*) *estimate-result*:
assumes $A \subseteq \{..<n\}$ $A \neq \{\}$
shows *measure* Ω $\{\omega.$ |*estimate* $(\tau\ \omega\ A) -$ *real* (*card* A)| $> \varepsilon *$ *real* (*card* A)$\} \leq \delta$ (**is** $?L \leq ?R$)
⟨*proof*⟩

unbundle *no-intro-cong-syntax*

end

10 Outer Algorithm

This section introduces the final solution with optimal size space usage. Internally it relies on the inner algorithm described in Section 6, dependending on the paramaters n, ε and δ it either uses the inner algorithm directly or if ε^{-1} is larger than $\ln n$ it runs $\frac{\varepsilon^{-1}}{\ln \ln n}$ copies of the inner algorithm (with the modified failure probability $\frac{1}{\ln n}$) using an expander to select its seeds. The theorems below verify that the probability that the relative accuracy of the median of the copies is too large is below ε.

theory *Distributed-Distinct-Elements-Outer-Algorithm*
imports
Distributed-Distinct-Elements-Accuracy
Prefix-Free-Code-Combinators.Prefix-Free-Code-Combinators
Frequency-Moments.Landau-Ext
Landau-Symbols.Landau-More
begin

unbundle *intro-cong-syntax*

The following are non-asymptotic hard bounds on the space usage for the sketches and seeds repsectively. The end of this section contains a proof that the sum is asymptotically in $\mathcal{O}(\ln(\varepsilon^{-1})\delta^{-1} + \ln n)$.

definition *state-space-usage* $= (\lambda(n,\varepsilon,\delta).\ 2\hat{}40 * (ln(1/\delta)+1)/\ \varepsilon\hat{}2 + log\ 2\ (log\ 2\ n + 3))$

definition *seed-space-usage* = $(\lambda(n,\varepsilon,\delta).\ 2\hat{\ }30{+}2\hat{\ }23{*}ln\ n{+}48{*}(log\ 2(1/\varepsilon){+}16)^2{+}336{*}ln\ (1/\delta))$

locale *outer-algorithm* =
  fixes *n* :: *nat*
  fixes δ :: *real*
  fixes ε :: *real*
  assumes *n-gt-0*: *n* > *0*
  assumes δ*-gt-0*: $\delta > 0$ **and** δ*-lt-1*: $\delta < 1$
  assumes ε*-gt-0*: $\varepsilon > 0$ **and** ε*-lt-1*: $\varepsilon < 1$
begin

definition n_0 **where** $n_0 = max\ (real\ n)\ (exp\ (exp\ 5))$
definition *stage-two* **where** *stage-two* = $(\delta < (1/ln\ n_0))$
definition δ_i :: *real* **where** δ_i = (*if stage-two then* $(1/ln\ n_0)$ *else* δ)
definition *m* :: *nat* **where** *m* = (*if stage-two then nat* $\lceil 4 * ln\ (1/\ \delta)/ln\ (ln\ n_0)\rceil$ *else 1*)
definition α **where** α = (*if stage-two then* $(1/ln\ n_0)$ *else 1*)

lemma *m-lbound*:
  assumes *stage-two*
  shows $m \geq 4 * ln\ (1/\ \delta)/ln(ln\ n_0)$
⟨*proof*⟩

lemma *n-lbound*:
  $n_0 \geq exp\ (exp\ 5)\ ln\ n_0 \geq exp\ 5\ 5 \leq ln\ (ln\ n_0)\ ln\ n_0 > 1\ n_0 > 1$
⟨*proof*⟩

lemma δ*1-gt-0*: $0 < \delta_i$
  ⟨*proof*⟩

lemma δ*1-lt-1*: $\delta_i < 1$
  ⟨*proof*⟩

lemma *m-gt-0-aux*:
  assumes *stage-two*
  shows $1 \leq ln\ (1\ /\ \delta)\ /\ ln\ (ln\ n_0)$
⟨*proof*⟩

lemma *m-gt-0*: *m* > *0*
⟨*proof*⟩

lemma α*-gt-0*: $\alpha > 0$
  ⟨*proof*⟩

lemma α*-le-1*: $\alpha \leq 1$
  ⟨*proof*⟩

sublocale *I*: *inner-algorithm n* δ_i ε
  ⟨*proof*⟩

abbreviation Θ **where** Θ ≡ $\mathcal{E}$ *m* α *I*.Ω

sublocale Θ: *expander-sample-space m* α *I*.Ω
  ⟨*proof*⟩

type-synonym *state* = *inner-algorithm.state list*

fun *single* :: *nat* ⇒ *nat* ⇒ *state* **where**
  single ϑ *x* = *map* (λ*j*. *I.single* (*select* Θ ϑ *j*) *x*) [*0*..<*m*]

fun *merge* :: *state* ⇒ *state* ⇒ *state* **where**
  merge x y = *map* (λ(*x*,*y*). *I.merge x y*) (*zip x y*)

fun *estimate* :: *state* ⇒ *real* **where**
  estimate x = *median m* (λ*i*. *I.estimate* (*x* ! *i*))

definition ν :: *nat* ⇒ *nat set* ⇒ *state*
  where ν ϑ *A* = *map* (λ*i*. *I*.τ (*select* Θ ϑ *i*) *A*) [*0*..<*m*]

The following three theorems verify the correctness of the algorithm. The term τ is a mathematical description of the sketch for a given subset, while *local.single*, *local.merge* are the actual functions that compute the sketches.

theorem *merge-result*: *merge* (ν ω *A*) (ν ω *B*) = ν ω (*A* ∪ *B*) (**is** *?L* = *?R*)
⟨*proof*⟩

theorem *single-result*: *single* ω *x* = ν ω {*x*} (**is** *?L* = *?R*)
⟨*proof*⟩

theorem *estimate-result*:
  assumes *A* ⊆ {..<*n*} *A* ≠ {}
  defines *p* ≡ (*pmf-of-set* {..<*size* Θ})
  shows *measure p* {ω. |*estimate* (ν ω *A*)− *real* (*card A*)| > ε ∗ *real* (*card A*)} ≤ δ (**is** *?L* ≤ *?R*)
⟨*proof*⟩

The function *encode-state* can represent states as bit strings. This enables verification of the space usage.

definition *encode-state*
  where *encode-state* = Lf_e *I.encode-state m*

lemma *encode-state*: *is-encoding encode-state*
  ⟨*proof*⟩

lemma *state-bit-count*:
  bit-count (*encode-state* (ν ω *A*)) ≤ *state-space-usage* (*real n*, ε, δ)
    (**is** *?L* ≤ *?R*)
⟨*proof*⟩

Encoding function for the seeds which are just natural numbers smaller than *sample-space.size* Θ.

definition *encode-seed*
  where *encode-seed* = Nb_e (*size* Θ)

lemma *encode-seed*:
  is-encoding encode-seed
  ⟨*proof*⟩

lemma *random-bit-count*:
  assumes ω < *size* Θ
  shows *bit-count* (*encode-seed* ω) ≤ *seed-space-usage* (*real n*, ε, δ)
    (**is** *?L* ≤ *?R*)
⟨*proof*⟩

The following is an alternative form expressing the correctness and space usage theorems. If *x* is expression formed by *local.single* and *local.merge* operations. Then *x* requires *state-space-usage* (*real n*, ε, δ) bits to encode and *estimate x* approximates the count of the distinct universe elements in the expression.

For example:

$estimate\ (local.merge\ (local.single\ \omega\ 1)\ (local.merge\ (local.single\ \omega\ 5)\ (local.single\ \omega\ 1)))$ approximates the cardinality of $\{1,\ 5,\ 1\}$ i.e. 2.

datatype *sketch-tree* = *Single nat* | *Merge sketch-tree sketch-tree*

fun *eval* :: *nat* ⇒ *sketch-tree* ⇒ *state*
  where
    eval ω (*Single x*) = *single* ω *x* |
    eval ω (*Merge x y*) = *merge* (*eval* ω *x*) (*eval* ω *y*)

fun *sketch-tree-set* :: *sketch-tree* ⇒ *nat set*
  where
    sketch-tree-set (*Single x*) = {*x*} |
    sketch-tree-set (*Merge x y*) = *sketch-tree-set x* ∪ *sketch-tree-set y*

theorem *correctness*:
  fixes *X*
  assumes *sketch-tree-set t* ⊆ {..<*n*}
  defines *p* ≡ *pmf-of-set* {..<*size* Θ}
  defines *X* ≡ *real* (*card* (*sketch-tree-set t*))
  shows *measure p* {ω. |*estimate* (*eval* ω *t*) − *X*| > ε ∗ *X*} ≤ δ (**is** *?L* ≤ *?R*)
⟨*proof*⟩

theorem *space-usage*:
  assumes ω < *size* Θ
  shows
    bit-count (*encode-state* (*eval* ω *t*)) ≤ *state-space-usage* (*real n*, ε, δ) (**is** *?A*)
    bit-count (*encode-seed* ω) ≤ *seed-space-usage* (*real n*, ε, δ) (**is** *?B*)
⟨*proof*⟩

end

The functions *state-space-usage* and *seed-space-usage* are exact bounds on the space usage for the state and the seed. The following establishes asymptotic bounds with respect to the limit $n, \delta^{-1}, \varepsilon^{-1} \to \infty$.

context
begin

Some local notation to ease proofs about the asymptotic space usage of the algorithm:

private definition *n-of* :: *real* × *real* × *real* ⇒ *real* **where** *n-of* = (λ(*n*, ε, δ). *n*)
private definition *δ-of* :: *real* × *real* × *real* ⇒ *real* **where** *δ-of* = (λ(*n*, ε, δ). δ)
private definition *ε-of* :: *real* × *real* × *real* ⇒ *real* **where** *ε-of* = (λ(*n*, ε, δ). ε)

private abbreviation *F* :: (*real* × *real* × *real*) *filter*
  where $F \equiv (at\text{-}top \times_F at\text{-}right\ 0 \times_F at\text{-}right\ 0)$

private lemma *var-simps*:
  n-of = *fst*
  ε-of = (λ*x*. *fst* (*snd x*))
  δ-of = (λ*x*. *snd* (*snd x*))
  ⟨*proof*⟩ **lemma** *evt-n*: *eventually* (λ*x*. *n-of x* ≥ *n*) *F*
  ⟨*proof*⟩ **lemma** *evt-n-1*: $\forall_F$ *x in F*. *0* ≤ *ln* (*n-of x*)
  ⟨*proof*⟩ **lemma** *evt-n-2*: $\forall_F$ *x in F*. *0* ≤ *ln* (*ln* (*n-of x*))
  ⟨*proof*⟩ **lemma** *evt-ε*: *eventually* (λ*x*. *1*/*ε-of x* ≥ ε ∧ *ε-of x* > *0*) *F*
  ⟨*proof*⟩ **lemma** *evt-δ*: *eventually* (λ*x*. *1*/*δ-of x* ≥ δ ∧ *δ-of x* > *0*) *F*
  ⟨*proof*⟩ **lemma** *evt-δ-1*: $\forall_F$ *x in F*. *0* ≤ *ln* (*1* / *δ-of x*)
  ⟨*proof*⟩

theorem *asymptotic-state-space-complexity*:
  state-space-usage $\in O[F](\lambda(n, \varepsilon, \delta).\ ln\ (1/\delta)/\varepsilon\hat{}2 + ln\ (ln\ n))$
  (**is** - $\in O[?F](?rhs)$)
$\langle proof \rangle$

theorem *asymptotic-seed-space-complexity*:
  seed-space-usage $\in O[F](\lambda(n, \varepsilon, \delta).\ ln\ (1/\delta)+ln\ (1/\varepsilon)\hat{}2 + ln\ n)$
  (**is** - $\in O[?F](?rhs)$)
$\langle proof \rangle$

definition *space-usage x* = *state-space-usage x* + *seed-space-usage x*

theorem *asymptotic-space-complexity*:
  space-usage $\in O[at\text{-}top \times_F at\text{-}right\ 0 \times_F at\text{-}right\ 0](\lambda(n, \varepsilon, \delta).\ ln\ (1/\delta)/\varepsilon\hat{}2 + ln\ n)$
$\langle proof \rangle$

end

unbundle *no-intro-cong-syntax*

end

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2002.

[3] J. Błasiok. Optimal streaming and tracking distinct elements with high probability. *ACM Trans. Algorithms*, 16(1):3:1–3:28, 2020.

[4] P. Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[5] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, pages 281–291, 2001.

[6] O. Goldreich. A sample of samplers: A computational perspective on sampling. In O. Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, volume 6650 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2011.

[7] V. Guruswami, C. Umans, and S. Vadhan. Unbalanced expanders and randomness extractors from parvaresh–vardy codes. *J. ACM*, 56(4), jul 2009.

[8] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, 2010.

[9] E. Karayel. Finite fields. *Archive of Formal Proofs*, June 2022. https://isa-afp.org/entries/Finite_Fields.html, Formal proof development.

[10] E. Karayel. Formalization of randomized approximation algorithms for frequency moments. *Archive of Formal Proofs*, April 2022. https://isa-afp.org/entries/Frequency_Moments.html, Formal proof development.

[11] E. Karayel. Expander graphs. *Archive of Formal Proofs*, March 2023. https://isa-afp.org/entries/Expander_Graphs.html, Formal proof development.

[12] D. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 167–175, USA, 2004. Society for Industrial and Applied Mathematics.