ACT Electoral Commission

Election Management System Modernisation

Ballot Paper Assessment - Randomisation Algorithm

V2.0

by Digital Elections Pty Ltd
4/935 Station Street, Box Hill North,
Victoria, Australia

and Blitzm Systems
Suite 7, 458 Swanston Street
Carlton, VIC 3053

info@blitzm.com
1300 211 248
www.blitzm.com.au

Revision History

Date	Description	Version	Author
05th July 2024	Drafted	Draft v1.0	[redacted]
23 July 2024	Addressing Ro's feedback and new `Ballot paper selection` section	V2.0	[redacted]

Table of Contents

Introduction 4
About this document 4
Overview 4
Background 4
System Description 5
Ballot Paper Selection 6
Algorithm Description 7
Implementation 9

Introduction

About this document

This document provides information about randomization algorithm integrated into the TIGER Ballot Paper Assessment module. It further summarises the code snippet used to generate randomization and other key elements.

Overview

This document describes the methods used to select scanned and digitized ballot papers for an audit. A subset of these ballot papers is chosen through a random but repeatable process.

The randomness ensures that no one can tamper with the selection, and the repeatability allows anyone to verify that the ballot papers were chosen randomly for the audit.

Background

As previously stated, the randomization process ensures that bad actors cannot tamper with the ballots selected for an audit. Additionally, the process is repeatable and verifiable when the correct data is provided.

Therefore, the primary requirements for the algorithm are as follows:

1. Repeatable
2. Randomised - This ensures that it is impossible to predict which ballots will be selected for an audit before they are committed to the system.

Here, a ballot paper is considered committed to the system once it’s scanned preferences and image data have been uploaded to Tiger, where they are arbitrarily ordered based on their sequence in the upload.

Ballot papers are maintained in batches of a maximum of 100 papers per batch and are entered into these batches in no order and can be considered randomly shuffled. These batches are maintained based on the enclosed ballot’s electorate and the polling place it originates from.

System Description

The process to randomly select ballot papers can be made repeatable by using a pseudo-random number generator that uses a particular seed. This seed will be a random number that is generated outside of the system. EACT accomplishes this using a raffle machine that will randomly choose a number between 1-99,999.

The system will then use this seed number to generate a sequence of pseudo-random numbers which are used to select the batches for the audit activity.
To ensure that the batches to be selected for an audit cannot be predicted before they are committed to the system, it is important to enforce a process wherein the random seed used to select batches cannot be entered or known till the ballots to be audited have been committed. This is accomplished by creating a process within the system that blocks seed entry till ballot papers have been uploaded.

There is an *Election Audit* for each election, during which the ballots for that election are audited. EACT uploads the scanned and digitised ballot papers to TIGER, where each ballot paper is tagged with its election, electorate, polling place, the sequence in which it was uploaded, it's position in the batch (the paper number), and the date and time of the upload. Ballot paper preferences and their ordering cannot be changed once they have been committed to the system.

Once ballot paper batches have been uploaded to the TIGER system, EACT can enter the Audit Seed. This seed is a number between 1-99,999 that is randomly chosen from a raffle machine. The Audit Seed must be linked to the date of a ballot paper upload activity.
The date and time of the selected upload is used to ensures that only batches uploaded before seed entry are considered for auditing. A seed can only be entered once for each calendar day, so it is important to ensure all batches meant to be audited have been committed to the system prior to seed entry.

There are electorate-specific Audits in each Election Audit, which can either be Daily or Ad-Hoc. Each audit must be linked to an Audit Seed, which is then used for selecting batches for an audit. For a daily audit, we only consider ballot papers that were committed on the same day as (but before) seed entry. For ad-hoc audits, the entire pool of un-audited ballot paper batches uploaded prior to seed entry is considered during audit batch generation.

Hence, the system is built to enforce the following order of operations –

1. Ballot paper preferences and image uploads
2. Seed number entry
3. Batch generation

Thus, it is not possible to enter a seed without first committing the batches to be audited to the system, ensuring that the ballots that will be chosen in the audit cannot be predicted prior to ballot paper upload.

Ballot Paper Selection

The uploaded ballot paper batches are randomly selected for auditing using the geometric skipping method recommended in '*Assessing the accuracy of the Australian Senate count'* (https://arxiv.org/pdf/2205.14634).
The geometric skipping method used is described in *'Bernoulli Ballot Polling: A Manifest Improvement for Risk-Limiting Audits'* (https://arxiv.org/pdf/1812.06361), the code for which is available on GitHub (https://github.com/pbstark/BernoulliBallotPolling)

While the original research paper applies the sampling method over all ballot papers, this method was adapted to select batches instead to simplify physical ballot paper retrieval during the audit process.

For auditing, a series of 5-10 consecutive ballot papers is selected from each selected batch. The number of ballot papers to be selected is set by EACT. The starting ballot paper for this series is randomly selected. This is accomplished by adding the selected batch number to the initially entered seed value (the one entered by EACT when setting up the Audit Seed). This new number is used as a seed to generate a random number, which corresponds to a ballot paper number within the batch. Once used, this new number is not used again for any other batches.

The randomly selected ballot is the start of the series, and the next 4-9 ballot papers in the batch are selected for the audit as well. If the starting ballot paper number is too high, the selection will be 'wrapped' around the batch. I.e., in a batch of 100, if our starting ballot number is 98, the ballots selected will be paper numbers [98, 99, 100, 1, 2] of the batch.

Algorithm Description

The process is divided into three main parts:

1. **Retrieve batches available for audit:** We retrieve the ballot paper batches intended for a specific audit that were uploaded before the audit's seed date and time.

```
func GetBallotsForAudit(audit):

        var query = database.BallotPapers
                .Where(election is audit.Election)
                .Where(electorate is audit.Electorate)
                .Where(fileUploadDateTime <= audit.Seed.DateTime);

        // if it is a daily audit, we restrict the batches to be from the same calendar day
        if(audit.Type  is “Daily”):
                query = query.Where(fileUploadCalendarDay is audit.Seed.CalendarDay);

        var batches = query.GroupBy(a => a.BatchNumber)
        return batches.ToList();
```

2. **Pick random ballot paper batches:** We use the process of geometric skipping, as mentioned in the previous section, to get the index of the ballot papers to sample, in a list.

```
func GeometricSkipping(int numberOfBatchesToSelect, int totalAvailableBatches, Random random):
        var sample = Set(size= numberOfBatchesToSelect)
        var samplingRate = numberOfBatchesToSelect / totalAvailableBatches;
         // if all ballots need to be audited
        if (samplingRate == 1):
                return [list of numbers from 0 to totalAvailableBatches]

         // generate random uniform variable using seed
var runifs = [list of random floating numbers, of length numberOfBatchesToSelect]
var randomVariables = [for each runif x, return ceiling(log(x)/log(1- samplingRate))];

var values = cumulativeSum(randomVariables);

for (i in values)
if (i < totalAvailableBatches)
sample.Add(i);

return sample
```

3. **Create audit batches:** This is the code snippet where we utilize the previously defined functions to generate an audit batch, comprising a set of papers designated for auditing.

```
func GenerateAuditBatches(audit):

var batches = GetBallotsForAudit(audit);
var createdBatches = 0;
var random = new Random(audit.Seed)

// keep removing selected batches and selecting more batches till we have the required number
of batches

while (createdBatches < audit.NumBatches):
// generate random numbers, create audit batches for each index



var selectedIndices = GeometricSkipping(audit.numberOfBatchesToSelect – createdBatches,
batches.Count, random);
for (index in selectedIndices):
        var batch = batches.AtIndex(index);
        var batchRandom = new Random(audit.Seed + batch.BatchNumber);
        var startingBallot = batchRandom.GetRandomNumber();

        database.Add(new AuditBatch(batch.BatchNumber, startingBallot));
        createdBatches++
```

Implementation

The current implementation is in .Net 8.0 and does not use any external libraries for sampling. We utilise the inbuilt System. Random library for random number generation.

The code can be found as below –

```
private async Task<List<BallotPaperScannedBatch>> GetBallotsForAudit(AuditEntity audit, Guid
electionId)
        {
            // retrieve batches and the number of ballots in the batch
            var batchesQuery = await this
                .dbContext.BallotPaperEntities.Include(a => a.File)
                .ThenInclude(a => a.Metadata)
                .Include(a => a.AuditActivityThread)
                .Where(a => a.BallotSource == BallotSourceDTO.Scanning && a.BallotType ==
BallotTypeDTO.Paper)
                .Where(a => a.ElectionId == electionId)
                .Where(a => a.ElectorateId == audit.ElectorateId)
                .Where(a => a.FileId != null)
                // ensures we're always getting files that are on or before the date selected
for the audit
                .Where(a => a.File.Metadata.CreatedAt <= audit.AuditSeed.Date)
                .OrderBy(a => a.PaperIndex)
                .ToListAsync();

            // if it's a daily audit, we only consider dates on the same day
            if (audit.AuditType == AuditTypeDTO.Daily)
            {
                batchesQuery = batchesQuery
                    .Where(a =>
                        TimeZoneInfo
                            .ConvertTime(audit.AuditSeed.Date, TimeZones.AustraliaCanberra)
                            .ToString("yyyy-MM-dd")
                        == TimeZoneInfo
                            .ConvertTime(a.File.Metadata.CreatedAt,
TimeZones.AustraliaCanberra)
                            .ToString("yyyy-MM-dd")
                    )
                    .ToList();
            }

            var batches = batchesQuery
                .GroupBy(a => new
                {
                    a.ScannedBatchNumber,
                    a.BatchNumber,
                    a.PollingPlaceId
                })
                .Where(a => a.All(x => x.AuditBatchId == null) && !a.All(x => x.IsInformal))
                .Select(a => new BallotPaperScannedBatch
                {
                    BatchNumber = a.Key.ScannedBatchNumber ?? a.Key.BatchNumber,
                    Count = a.Count(),
                    PollingPlaceId = a.Key.PollingPlaceId,
                    Papers = [.. a.OrderBy(a => a.PaperNumber)],
                })
                .ToList();

            return batches;
        }
private static List<int> GeometricSkipping(int numBatches, int totalBatches, Random random)
        {
            var sample = new HashSet<int>(numBatches);
            var samplingRate = (double)numBatches / totalBatches;

            // if all ballots need to be audited
            if (samplingRate == 1)
            {
                return Enumerable.Range(0, numBatches).ToList();
            }

            // generate random uniform variable using seed
            var runifs = Enumerable.Range(0, numBatches).Select(i =>
random.NextDouble()).ToList();
            var randomVariables = runifs
                .Select(x => (int)Math.Ceiling(Math.Log(x) / Math.Log(1 - samplingRate)))
                .ToList();

            var vals = GetCumulativeSum(randomVariables).Select(x => x - 1).ToList();

            for (int i = 0; i < numBatches; i++)
            {
                if (vals[i] < totalBatches)
                {
                    sample.Add(vals[i]);
                }
            }
            return [.. sample];
        }
public override async Task<Empty> GenerateAuditBatches(
            GenerateAuditBatchesRequest request,
            ServerCallContext context
        )
        {
            var audit =
                await this
                    .dbContext.AuditEntities.Include(a => a.AuditSeed)
                    .ThenInclude(a => a.Metadata)
                    .Include(a => a.Electorate)
                    .Include(a => a.ElectionAudit)
                    .Include(a => a.AuditActivityThread)
                    .SingleOrDefaultAsync(a => a.Id.ToString() == request.Id)
                ?? throw new RpcException(
                    new Status(StatusCode.NotFound, "Audit not found. Invalid audit ID in
request.")
                );

            var batches = await this.GetBallotsForAudit(audit,
audit.ElectionAudit.ElectionId);

            if (batches.Count == 0)
            {
                throw new RpcException(
                    new Status(
                        StatusCode.FailedPrecondition,
                        $"No batches to be audited with seed {audit.AuditSeed.Seed} were
found."
                    )
                );
            }

            if (audit.NumBatches > batches.Count)
            {
                throw new RpcException(
                    new Status(
                        StatusCode.FailedPrecondition,
                        $"Number of batches to be selected for audit exceeds number of
batches available."
                    )
                );
            }

            var createdBatches = 0;
            var random = new Random(audit.AuditSeed.Seed);

            // keep removing selected batches and selecting more batches till we have the
required number of batches
            while (createdBatches < audit.NumBatches)
            {
                // generate random numbers, create audit batches for each index
                var selectedIndices = GeometricSkipping(audit.NumBatches - createdBatches,
batches.Count, random);
                var selectedBatches = batches.Where((a, index) =>
selectedIndices.Contains(index));

                foreach (var batch in selectedBatches)
                {
                    // randomly generated starting ballot paper number
                    var batchRandom = new Random(audit.AuditSeed.Seed + batch.BatchNumber);
                    var startingNumber = batchRandom.Next(1, batch.Count + 1);

                    var ballotPapers = new List<BallotPaperEntity>(audit.PapersPerBatch);

                    // if the batch has fewer papers than are required, select all papers in
the batch
                    if (batch.Count <= audit.PapersPerBatch)
                    {
                        ballotPapers.AddRange(batch.Papers);
                        startingNumber = 1;
                    }
                    else
                    {
                        // ballot paper numbers are indexed from 1, so we handle that here
                        ballotPapers.AddRange(
                            batch.Papers.GetRange(
                                startingNumber - 1,
                                Math.Min(audit.PapersPerBatch, batch.Count - startingNumber +
1)
                            )
                        );

                        if (ballotPapers.Count < audit.PapersPerBatch)
                        {
                            ballotPapers.AddRange(batch.Papers.Take(audit.PapersPerBatch -
ballotPapers.Count));
                        }
                    }

                    for (int i = 0; i < ballotPapers.Count; i++)
                    {
                        ballotPapers[i].AuditBatchIndex = i;
                    }

                    await dbContext.AddAsync(
                        new AuditBatchEntity
                        {
                            AuditId = audit.Id,
                            Status = AuditBatchStatusDTO.Pending,
                            BatchNumber = batch.BatchNumber,
                            PollingPlaceId = batch.PollingPlaceId,
                            StartingBallotNumber = startingNumber,
                            BallotPapers = ballotPapers,
                            Metadata =
MetadataEntity.CreateBy(this.httpContextAccessor.HttpContext.User),
                            AuditActivityThread = AuditActivityThreadEntity.NewThread(),
                        }
                    );

                    createdBatches += 1;
                }
                var selectedBatchNumbers = selectedBatches.Select(x => x.BatchNumber);
                batches.RemoveAll(x => selectedBatchNumbers.Contains(x.BatchNumber));
            }

            audit.AuditActivityThread.AddActivity(
                this.httpContextAccessor.HttpContext.User,
                $"Generated batches for {audit.Electorate.Name} {audit.AuditType} with seed
{audit.AuditSeed.Seed}",
                AuditActionTypeDTO.AuditActionTypeAudit
            );
            await this.dbContext.SaveChangesAsync();
            return new Empty();
        }
```