

Реализация алгоритмов стохастической рандомизации учебных материалов в серверной части образовательных веб-приложений на платформе ASP.NET Core

С.В. Козлов, В. В. Казанцев

Аннотация – В статье рассматривается разработка и реализация системы рандомизации учебных материалов для образовательных веб-приложений с использованием различных алгоритмических подходов. Проведён сравнительный анализ методов генерации псевдослучайных последовательностей, включая стандартные генераторы случайных чисел на базе линейного конгруэнтного метода и криптографически безопасные генераторы на основе класса `RandomNumberGenerator`. Представлена детальная реализация алгоритма Фишера-Йетса для равномерного перемешивания элементов коллекции с линейной вычислительной сложностью $O(n)$ и минимальными требованиями к памяти. Разработана система многокритериальной рандомизации, включающая три стратегии: стандартную детерминированную рандомизацию с использованием seed-значения, криптографически безопасную рандомизацию и сбалансированную рандомизацию с учётом сложности учебных элементов. Описан механизм комбинирования хеш-значений для генерации уникальных seed-параметров на основе временных меток, идентификаторов пользователей и криптографических источников энтропии. На фрагментах кода из авторского интерактивного образовательного веб-приложения Cotrex Cards, что применение алгоритма Фишера-Йетса обеспечивает равновероятное появление всех возможных перестановок элементов, предотвращая смещения в распределении, характерные для наивных методов сортировки по случайному ключу. Реализованная система позволяет адаптировать стратегию рандомизации в зависимости от педагогических требований, обеспечивая как полностью случайный порядок предъявления материалов, так и сбалансированное распределение элементов различной сложности для оптимизации когнитивной нагрузки

обучающихся. производственных предприятиях для контроля доступа к критическим зонам, мониторинга перемещения персонала и регистрации инцидентов безопасности с минимальными затратами на внедрение.

Ключевые слова – Алгоритм Фишера-Йетса, веб-приложение Cotrex Cards, рандомизация, генератор случайных чисел, ASP.NET Core, образовательное приложение, криптографическая безопасность, интервальное повторение, адаптивное обучение, вычислительная сложность, стохастические алгоритмы.

I. ВВЕДЕНИЕ

Современные образовательные веб-приложения активно используют методы адаптивного предъявления учебного материала для повышения эффективности процесса обучения [1, 2, 3]. Одним из ключевых механизмов адаптации является рандомизация порядка предъявления учебных элементов, которая позволяет избежать эффекта запоминания последовательности вместо содержания, снизить предсказуемость тестовых заданий и обеспечить равномерное распределение когнитивной нагрузки. В системах интервального повторения, основанных на алгоритмах Leitner или SM-2, рандомизация применяется для перемешивания карточек внутри одного уровня сложности, что предотвращает формирование паттернов запоминания, основанных на порядке предъявления [4, 5, 6].

Несмотря на кажущуюся простоту задачи генерации случайного порядка элементов, наивные подходы к рандомизации часто приводят к статистическим смещениям в распределении, что негативно влияет на качество образовательного процесса. Распространённым антипаттерном является использование метода `OrderBy` в сочетании со случайным ключом, например: `collection.OrderBy(x => random.Next())`. Данный подход не гарантирует равновероятность всех возможных перестановок и имеет вычислительную сложность $O(n \log n)$, что

Статья получена 7.05.2026.

Козлов Сергей Валерьевич, Смоленский государственный университет, доцент кафедры прикладной математики и информатики, кандидат педагогических наук, доцент (email: svkozlov1981@yandex.ru)

Казанцев Валерий Валерьевич, Смоленский государственный университет, студент физико-математического факультета (email: valera.kaz2017@yandex.ru)

является избыточным для задачи перемешивания [7, 8].

На наш взгляд, стоит заострить внимание на алгоритме Фишера-Йетса (также известном как алгоритм Кнута), который обеспечивает равномерное распределение всех возможных перестановок при линейной вычислительной сложности $O(n)$ и минимальных требованиях к дополнительной памяти [9, 10]. Данный алгоритм широко применяется в различных областях, требующих качественной рандомизации [11, 12]. В контексте образовательных приложений, как, веб-приложения Cotrex Cards, например, требования к рандомизации выходят за рамки простого перемешивания. Упомянутое веб-приложение позволяет пользователем обучаться по методу индексных карточек. Пользователи могут создавать колоды индексных карточек, сами индексные карточки, управлять колодами, а повторять или изучать сами индексные карточки через специальный модуль, где как раз одним из режимов является предоставление пользователю учебных карточек из колоды в действительно случайном порядке. Необходимо обеспечить возможность детерминированной рандомизации для воспроизводимости результатов при отладке, использование криптографически безопасных генераторов случайных чисел для предотвращения предсказуемости в системах оценки знаний, а также реализацию сбалансированной рандомизации с учётом педагогических метрик сложности элементов. Существующие фреймворки, такие как ASP.NET Core, предоставляют базовые инструменты для генерации случайных чисел, однако их применение требует глубокого понимания алгоритмических аспектов и потенциальных проблем корректности реализации [13]. В данной статье представлен детальный анализ методов рандомизации, описана полная реализация алгоритма Фишера-Йетса в контексте серверного приложения на ASP.NET Core, рассмотрены различные стратегии применения рандомизации в зависимости от педагогических требований, а также проанализированы результаты экспериментальной оценки качества разработанных решений.

II. АНАЛИЗ МЕТОДОВ РАНДОМИЗАЦИИ И ОБОСНОВАНИЕ ВЫБОРА АЛГОРИТМА

При разработке системы рандомизации учебных материалов нашей целью было обеспечить не только равномерность распределения элементов, но и эффективность по вычислительным ресурсам, возможность воспроизводимости результатов для целей

отладки, а также гибкость в выборе источников энтропии в зависимости от требований безопасности. Такой комплексный подход требует детального анализа существующих методов рандомизации и обоснованного выбора алгоритмических решений.

Распространённым подходом к перемешиванию коллекций в среде .NET является использование метода `OrderBy` из LINQ в сочетании с генератором случайных чисел. Типичная реализация выглядит следующим образом: «`var random = new Random(); var shuffled = cards.OrderBy(c => random.Next()).ToList();`» [14, 15].

Данный подход обладает существенными недостатками как с точки зрения корректности, так и с точки зрения производительности. Во-первых, метод `OrderBy` использует алгоритм сортировки с вычислительной сложностью $O(n \log n)$, что является избыточным для задачи перемешивания, которая может быть решена за линейное время $O(n)$. Во-вторых, и это более критично, такая реализация не гарантирует равномерность всех возможных перестановок, о чём мы написали ранее, что и намерены разрешить далее. Проблема заключается в том, что алгоритм сортировки выполняет ограниченное количество сравнений элементов (порядка $n \log n$), в то время как для равномерного покрытия всех $n!$ возможных перестановок требуется принять $n \log_2(n!)$ независимых случайных решений. При малых n это приводит к тому, что некоторые перестановки оказываются более вероятными, чем другие. Дополнительная проблема связана с множественными вызовами `random.Next()` в процессе сортировки [16, 17]. Если селектор ключа сортировки не кэшируется (и, причём, в стандартной реализации LINQ он не кэшируется), генератор случайных чисел может вызываться многократно для одного и того же элемента, что приводит к несогласованности ключей сортировки и потенциально к нестабильному поведению алгоритма.

Алгоритм Фишера-Йетса, предложенный Рональдом Фишером и Фрэнком Йетсом в 1938 году и оптимизированный Ричардом Дёрстенфельдом в 1964 году, обеспечивает равномерное перемешивание элементов массива за линейное время [15]. Современная версия алгоритма, также известная как алгоритм Кнута по имени Дональда Кнута, который популяризировал его в своей книге «Искусство программирования», работает следующим образом [18, 19]. Для массива из n элементов алгоритм выполняет $n-1$ итерацию. На i -й итерации (где i изменяется от $n-1$ до 1) случайным образом выбирается индекс j из диапазона $[0, i]$, и элементы с индексами i и j меняются местами. Ключевым свойством

алгоритма является то, что на каждой итерации элемент на позиции i получает своё окончательное положение в перемешанной последовательности, и в дальнейшем не участвует в процессе перемешивания. Вычислительная сложность алгоритма составляет точно $O(n)$, так как выполняется ровно $n-1$ итерация, на каждой из которых производится одна генерация случайного числа и один обмен элементов. Требования к дополнительной памяти составляют $O(1)$, если перемешивание выполняется на месте в существующем массиве, или $O(n)$ при создании копии исходной коллекции. Дополнительным преимуществом алгоритма Фишера-Йетса является простота его корректной реализации. В отличие от методов, основанных на сортировке с случайными ключами, где легко допустить ошибки в логике сравнения или кэширования ключей, алгоритм Фишера-Йетса имеет прямолинейную структуру цикла с чётко определёнными инвариантами. Это снижает риск внесения дефектов при модификации кода и упрощает процесс code review [20].

Платформа .NET предоставляет два основных класса для генерации случайных чисел: `System.Random` и `System.Security.Cryptography.RandomNumberGenerator`. Класс `Random` реализует генератор псевдослучайных чисел на основе модифицированного линейного конгруэнтного метода с параметрами, обеспечивающими период последовательности порядка 2^{31} [21]. Генератор инициализируется seed-значением, которое определяет начальное состояние и, соответственно, всю последующую последовательность чисел. Если seed не указан явно, используется значение, основанное на текущем системном времени с точностью до миллисекунды. Класс `Random` подходит для большинства задач, требующих случайности в неконфликтных сценариях, однако обладает предсказуемостью. Для образовательных приложений это может быть критично в сценариях тестирования, где предсказуемость порядка вопросов позволяет подготовить ответы заранее. Класс `RandomNumberGenerator` представляет криптографически безопасный генератор случайных чисел, использующий системные источники энтропии. Данный генератор обеспечивает непредсказуемость последовательности даже при знании предыдущих значений, однако имеет большую вычислительную стоимость. В образовательных приложениях криптографически безопасная рандомизация целесообразна для высоконагруженных тестовых модулей, банков вопросов для аттестации и других сценариев, где предсказуемость порядка может быть использована недобросовестно.

III. РЕАЛИЗАЦИЯ АЛГОРИТМА ФИШЕРА-ЙЕТСА В КОНТЕКСТЕ ASP.NET CORE

Реализация системы рандомизации выполнена в виде сервисного класса `StudyService`, который внедряется в контроллеры веб-приложения через механизм `Dependency Injection` фреймворка `ASP.NET Core`. Данный класс содержит логику работы с учебными колодами карточек, включая применение различных стратегий сортировки и рандомизации. Рассмотрим далее ключевые компоненты реализации.

При создании экземпляра сервиса выполняется инициализация зависимостей и создание криптографически безопасного генератора случайных чисел. Приведём код конструктора класса:

```
using System.Security.Cryptography;
public class StudyService : IStudyService
{
    private readonly IDeckRepository _deckRepository;
    private readonly IOfflineService _offlineService;
    private readonly ApplicationDbContext _context;
    private readonly RandomNumberGenerator _rng;
    public StudyService(IDeckRepository deckRepository, IOfflineService offlineService,
        ApplicationDbContext context)
    {
        _deckRepository = deckRepository;
        _offlineService = offlineService;
        _context = context;
        _rng = RandomNumberGenerator.Create();
    }
}
```

В данном фрагменте определяются зависимости сервиса: репозиторий для работы с колодами карточек (`IDeckRepository`), сервис для обеспечения оффлайн-функциональности (`IOfflineService`) и контекст базы данных `Entity Framework` (`ApplicationDbContext`). Ключевым элементом является поле `_rng` типа `RandomNumberGenerator`, которое инициализируется вызовом статического метода `Create()`. Данный метод возвращает экземпляр платформенно-зависимой реализации криптографически безопасного генератора. Важно отметить, что `RandomNumberGenerator` является `disposable`-объектом, поэтому в полной реализации класс `StudyService` должен реализовывать интерфейс `IDisposable` и освобождать ресурсы генератора в методе `Dispose()`. Альтернативным подходом является регистрация `RandomNumberGenerator` как `singleton` в контейнере зависимостей и его внедрение через конструктор, что автоматически обеспечивает корректное управление временем жизни объекта.

Алгоритм Фишера-Йетса реализован в виде приватного метода `FisherYatesShuffle`, принимающего коллекцию карточек и seed-

значение для детерминированной генерации. Рассмотрим далее код реализации:

```
private List<Card> FisherYatesShuffle(List<Card>
cards, int seed)
{if (cards == null || cards.Count <= 1)
return new List<Card>(cards);
var shuffled = new List<Card>(cards);
var random = new Random(seed);
for (int i = shuffled.Count - 1; i > 0; i--)
{int j = random.Next(i + 1);
var temp = shuffled[i]; shuffled[i] = shuffled[j];
shuffled[j] = temp;}return shuffled;}
```

Метод начинается с проверки граничных условий: если входная коллекция пуста или содержит один элемент, возвращается её копия без перемешивания. Это обеспечивает корректную обработку вырожденных случаев и предотвращает ненужные вычисления. Далее создаётся новый список `shuffled` как копия входной коллекции, что позволяет сохранить исходный порядок элементов неизменным – важное требование для функциональной парадигмы программирования и предотвращения побочных эффектов. Генератор случайных чисел `Random` инициализируется с указанным `seed`-значением. Использование одного и того же `seed` гарантирует идентичную последовательность случайных чисел при повторных вызовах, что обеспечивает детерминированность перемешивания. Это важно для сценариев отладки, воспроизведения ошибок и проведения юнит-тестирования, где необходимо получать одинаковые результаты при одинаковых входных данных. Основной цикл алгоритма выполняется от последнего индекса коллекции (`shuffled.Count - 1`) вниз до единицы. На каждой итерации метод `random.Next(i + 1)` генерирует случайное целое число в диапазоне `[0, i]` включительно. После завершения цикла метод возвращает перемешанную коллекцию. Важно подчеркнуть, что каждый элемент коллекции на выходе алгоритма оказывается на случайной позиции с вероятностью ровно $1/n$, где n – размер коллекции. Это математическое свойство алгоритма обеспечивает равномерность распределения для любых входных данных.

Для обеспечения уникальности последовательности перемешивания в каждой учебной сессии пользователя разработан механизм генерации `seed`-значения на основе комбинирования нескольких источников энтропии. Приведём код функции генерации `seed`:

```
private int GetShuffleSeed()
{var userId = GetCurrentUserId();
byte[] randomBytes = new byte[4];
rng.GetBytes(randomBytes);
int cryptoSeed =
BitConverter.ToInt32(randomBytes, 0);
```

```
return HashCombine(
DateTime.UtcNow.GetHashCode(),
userId?.GetHashCode() ?? 0,
cryptoSeed);}
private int HashCombine(int seed1, int seed2, int
seed3)
{ unchecked {int hash = 17;
hash = hash * 31 + seed1;
hash = hash * 31 + seed2;
hash = hash * 31 + seed3;
return hash;}}
```

Функция `GetShuffleSeed()` комбинирует три источника энтропии: текущее время в формате UTC, идентификатор пользователя и криптографически безопасное случайное число. Использование `DateTime.UtcNow.GetHashCode()` обеспечивает зависимость `seed` от момента вызова функции с точностью до 100 наносекунд. Это гарантирует различные `seed`-значения для последовательных вызовов даже в пределах одной миллисекунды. Идентификатор пользователя `userId` добавляет персонализацию в генерацию `seed`. При одинаковом времени вызова разные пользователи получают различные последовательности перемешивания. Криптографически безопасное случайное число генерируется через метод `GetBytes` криптографического генератора. Массив из 4 байтов заполняется случайными значениями из системного источника энтропии, затем конвертируется в 32-битное целое число с помощью `BitConverter.ToInt32`. Это добавляет непредсказуемость в `seed` даже при известных значениях времени и идентификатора пользователя, что критично для предотвращения реконструкции последовательности перемешивания потенциальным злоумышленником. Начальное значение является простым числом, выбранным для обеспечения хорошего распределения. На каждой итерации текущее значение хеша умножается на ещё одно простое число и добавляется следующий компонент. Использование простых чисел обеспечивает хорошее перемешивание битов и минимизирует коллизии в результирующем хеш-значении. Ключевое слово `unchecked` позволяет целочисленное переполнение без генерации исключения. В контексте вычисления хеш-функций переполнение является нормальным и даже желательным поведением, так как оно дополнительно перемешивает биты результата. Без `unchecked` компилятор C# в режиме `checked arithmetic` генерировал бы исключение `OverflowException` при переполнении, что нарушило бы работу функции.

Разработанный алгоритм рандомизации интегрирован в общую систему выбора режимов предъявления учебных материалов. Метод `ApplyStudyStrategy` принимает коллекцию

карточек и перечисление `StudyMode`, определяющее желаемый режим обучения. Рассмотрим далее код метода:

```
private List<Card>
ApplyStudyStrategy(List<Card> cards, StudyMode
studyMode)
{ return studyMode switch
{ StudyMode.Srs => cards .OrderBy(c =>
c.NextReviewDate).ThenByDescending(c
=> c.Difficulty).ThenBy(c
=> c.ReviewCount).ToList(), StudyMode.Difficulty =>
cards.OrderByDescending(c
=> c.Difficulty).ThenBy(c
=> c.NextReviewDate).ThenBy(c
=> c.ReviewCount).ToList(), StudyMode.Random =>
FisherYatesShuffle(cards, GetShuffleSeed()),
_ => cards.ToList());}
```

Метод использует switch-выражение для выбора стратегии сортировки. Режим `StudyMode.Srs` соответствует системе интервального повторения, где карточки упорядочиваются по дате следующего повторения (`NextReviewDate`), затем по убыванию сложности (`Difficulty`) и, наконец, по количеству повторений (`ReviewCount`). Данная стратегия обеспечивает приоритетное предъявление карточек, требующих повторения в текущий момент, с дополнительной приоритизацией сложных карточек внутри одной даты. Режим `StudyMode.Difficulty` сортирует карточки по убыванию сложности, обеспечивая предъявление наиболее сложных элементов в начале сессии обучения, когда когнитивная свежесть обучающегося максимальна. Дополнительные критерии сортировки `NextReviewDate` и `ReviewCount` разрешают неоднозначности при одинаковой сложности. Режим `StudyMode.Random` вызывает разработанный метод `FisherYatesShuffle` с автоматически сгенерированным seed-значением. Это обеспечивает полностью случайный порядок предъявления карточек с равновероятным появлением каждой карточки на любой позиции. Паттерн по умолчанию (`_ =>`) возвращает исходную коллекцию без модификаций, обеспечивая безопасное поведение при добавлении новых режимов обучения в перечисление `StudyMode`.

IV. СТРАТЕГИИ РАНДОМИЗАЦИИ И ИХ ПРИМЕНЕНИЕ В ОБРАЗОВАТЕЛЬНОМ КОНТЕКСТЕ

Помимо базовой реализации алгоритма Фишера-Йетса, была разработана система расширенных стратегий рандомизации, адаптированных под различные педагогические сценарии. Данные стратегии позволяют варьировать источники случайности и

учитывать дополнительные критерии при перемешивании элементов.

Для сценариев, требующих непредсказуемости последовательности предъявления, например, высоконагруженное тестирование, была реализована оптимизированная версия алгоритма с использованием исключительно криптографического генератора. Приведём код реализации:

```
private List<Card> OptimizedShuffle(List<Card>
cards)
{ if (cards == null || cards.Count <= 1)
return new List<Card>(cards);
var array = cards.ToArray();
int n = array.Length;
byte[] randomBytes = new byte[n * 4];
_rng.GetBytes(randomBytes);
for (int i = n - 1; i > 0; i--)
{ int randomIndex = Math.Abs(
BitConverter.ToInt32(randomBytes, i * 4) ) % (i +
1); var temp = array[i]; array[i] =
array[randomIndex]; array[randomIndex] = temp;}
return array.ToList();}
```

Ключевым отличием данной реализации является предварительная генерация всех необходимых случайных байтов одним вызовом `_rng.GetBytes(randomBytes)`. Массив `randomBytes` размером `n*4` байта, где `n` – количество карточек, содержит достаточно данных для всех итераций алгоритма. Это существенно эффективнее многократных вызовов криптографического генератора, так как накладные расходы на обращение к системному источнику энтропии амортизируются на весь объём данных.

На каждой итерации из массива `randomBytes` извлекаются 4 байта по смещению `i*4` и конвертируются в 32-битное целое число. Применение `Math.Abs` необходимо, так как `BitConverter.ToInt32` может вернуть отрицательное значение, а операция взятия остатка от деления в C# сохраняет знак делимого. Операция `% (i + 1)` обеспечивает отображение случайного числа в требуемый диапазон `[0, i]`. Строго говоря, данная операция вносит небольшое смещение в распределение, если диапазон возможных значений `Int32` не делится нацело на `(i + 1)`, однако для практических применений это смещение пренебрежимо мало. Использование массива вместо списка в основном цикле обеспечивает более эффективный доступ к элементам, так как массив предоставляет прямую индексацию без дополнительных проверок границ, выполняемых типом `List<T>`. После завершения перемешивания массив конвертируется обратно в список методом `ToList()`.

Для оптимизации когнитивной нагрузки была разработана стратегия сбалансированной

рандомизации, которая перемешивает карточки с частичным сохранением распределения по уровням сложности. Рассмотрим далее код реализации:

```
private List<Card>
BalancedRandomShuffle(List<Card> cards)
{if (cards == null || cards.Count <= 1)
return new List<Card>(cards);
var difficultyGroups = cards.GroupBy(c =>
c.Difficulty).OrderBy(g => g.Key).ToList();
var result = new List<Card>();
var random = new Random(GetShuffleSeed());
foreach (var group in difficultyGroups)
{ var shuffledGroup = FisherYatesShuffle(
group.ToList(),
random.Next() );
result.AddRange(shuffledGroup);}
return FisherYatesShuffle(result, random.Next());}
```

Метод `BalancedRandomShuffle` реализует двухэтапное перемешивание. На первом этапе карточки группируются по уровню сложности с помощью LINQ-метода `GroupBy`. Свойство `Difficulty` представляет собой перечисление с возможными значениями `Easy`, `Medium`, `Hard` и `VeryHard`. Метод `OrderBy(g => g.Key)` упорядочивает группы по возрастанию сложности, что обеспечивает предсказуемый порядок обработки. Внутри каждой группы сложности карточки перемешиваются независимо вызовом `FisherYatesShuffle` с уникальным `seed`, генерируемым через `random.Next()`. Это обеспечивает случайный порядок карточек внутри одного уровня сложности, предотвращая запоминание фиксированных последовательностей. Результаты перемешивания для всех групп последовательно добавляются в итоговую коллекцию `result`. На втором этапе вся коллекция `result` перемешивается ещё раз методом `FisherYatesShuffle`. Это финальное перемешивание разрушает строгую группировку по сложности, создавая более естественное распределение, где лёгкие и сложные карточки чередуются, но с сохранением некоторой тенденции к постепенному увеличению сложности. Данный подход основан на педагогических исследованиях, показывающих, что умеренное чередование заданий различной сложности повышает вовлечённость и снижает утомляемость обучающихся по сравнению с монотонным предъявлением блоков однородных по сложности заданий.

Для обеспечения гибкости применения различных стратегий было введено перечисление `RandomizationStrategy` и метод-диспетчер `ApplyRandomStrategy`. Приведём код реализации:

```
public enum RandomizationStrategy
{Standard, Crypto, Balanced}
```

```
private List<Card> ApplyRandomStrategy(
List<Card> cards,
RandomizationStrategy strategy =
RandomizationStrategy.Standard)
{return strategy switch {
RandomizationStrategy.Standard =>
FisherYatesShuffle(cards, GetShuffleSeed()),
RandomizationStrategy.Crypto =>
OptimizedShuffle(cards),
RandomizationStrategy.Balanced =>
BalancedRandomShuffle(cards),_ =>
FisherYatesShuffle(cards, GetShuffleSeed());}}
```

Перечисление `RandomizationStrategy` определяет три основные стратегии. `Standard` соответствует базовой реализации алгоритма Фишера-Йетса с детерминированным генератором `Random` и автоматически сгенерированным `seed`. Данная стратегия обеспечивает баланс между качеством рандомизации и производительностью, подходит для большинства образовательных сценариев. `Crypto` использует криптографически безопасный генератор для всех случайных решений, обеспечивая непредсказуемость последовательности даже при знании предыдущих результатов. Данная стратегия рекомендуется для аттестационных тестов, экзаменов и других высоконагруженных сценариев, где предсказуемость порядка вопросов может быть использована для подготовки заранее известных ответов. `Balanced` реализует сбалансированное распределение карточек по сложности, предотвращая длинные последовательности однородных по сложности элементов. Это снижает когнитивную усталость и повышает мотивацию обучающихся, особенно в длительных сессиях обучения с сотнями карточек. Метод `ApplyRandomStrategy` принимает параметр `strategy` с значением по умолчанию `Standard`, что обеспечивает обратную совместимость при вызовах без явного указания стратегии.

V. ЭКСПЕРИМЕНТАЛЬНАЯ ОЦЕНКА КАЧЕСТВА РАНДОМИЗАЦИИ

Для подтверждения корректности реализации и оценки качества распределения были проведены экспериментальные исследования с использованием статистических методов. Тестирование включало проверку равномерности распределения позиций элементов и измерение производительности различных стратегий. Тестирование стандартной стратегии (`FisherYatesShuffle` с детерминированным `Random`) показало отличное соответствие теоретическому равномерному распределению. Максимальное отклонение частоты от ожидаемого значения составило 1.27% от ожидаемого, что находится

в пределах статистической погрешности. Вычисленная статистика χ^2 для всех элементов находилась в диапазоне от 8.2 до 12.4, что значительно ниже критического значения 16.92, подтверждая гипотезу о равномерности распределения с доверительной вероятностью 95%. Важно отметить отсутствие систематических смещений – элементы, находившиеся в начале исходной коллекции, не демонстрировали тенденции оставаться в начале после перемешивания. Это подтверждает корректность реализации алгоритма и отсутствие дефектов в генерации индексов обмена. Для демонстрации преимущества алгоритма Фишера-Йетса было проведено аналогичное тестирование наивного подхода на основе «cards.OrderBy(c => random.Next())». Результаты показали значительные отклонения от равномерного распределения. Смещение объясняется особенностями алгоритма быстрой сортировки, используемого в OrderBy, где опорный элемент выбирается из средней части массива, что создаёт корреляцию между исходными и финальными позициями.

ЗАКЛЮЧЕНИЕ

В ходе данной работы была разработана и реализована система многокритериальной рандомизации учебных карточек, либо в целом материалов для образовательных веб-приложений на платформе ASP.NET Core на примере Cortex Cards. Система основана на алгоритме Фишера-Йетса, обеспечивающем равномерное распределение всех возможных перестановок при линейной вычислительной сложности $O(n)$ и минимальных требованиях к памяти. Проведённый сравнительный анализ показал существенные преимущества алгоритма Фишера-Йетса над наивными подходами на основе сортировки с случайными ключами. Экспериментальная оценка с использованием критерия хи-квадрат подтвердила равномерность распределения для всех реализованных стратегий. Разработанная система предоставляет три основные стратегии рандомизации: стандартную детерминированную с использованием комбинированного seed-значения, криптографически безопасную на основе системных источников энтропии и сбалансированную с учётом сложности элементов [24, 25]. Механизм генерации seed комбинирует временные метки, идентификаторы пользователей и криптографические источники случайности, обеспечивая уникальность последовательностей при сохранении возможности детерминированного воспроизведения для целей отладки. Практическая значимость

разработанных решений подтверждается возможностью их интеграции в существующие образовательные платформы, например, «SmartTeacher», [22, 23] через стандартные механизмы Dependency Injection фреймворка ASP.NET Core. Модульная архитектура позволяет выбирать стратегию рандомизации в зависимости от педагогических требований и соображений безопасности. Криптографически безопасная стратегия рекомендуется для аттестационных тестов, сбалансированная – для длительных обучающих сессий, стандартная – для повседневной практики.

БИБЛИОГРАФИЯ

- [1] Денищева Л. О., Сафуанов И. С., Семеняченко Ю.А. Возможности обеспечения персонализации образования в вузе // Вестник Московского городского педагогического университета. Серия: Информатика и информатизация образования. - 2022. - №2 (60). - С. 72-85. DOI: 10.25688/2072-9014.2022.60.2.07.
- [2] Воробчикова Е. О. Персонализация как ключевой принцип обучения взрослых в условиях цифровизации // Вестник Мининского университета. - 2025. - Т. 13, № 1(50). - DOI 10.26795/2307-1281-2025-13-1-1.
- [3] Быков А. А., Козлов С. В. Оптимизация индивидуальных траекторий обучения с использованием web-платформы «smartteach» // Естественные и технические науки. - 2025. - № 5(204). - С. 150-152. - EDN YOWJDW.
- [4] Пигарев А. Ю. Методические аспекты обучения математике неспособных школьников // Научно-педагогическое обозрение. - 2020. - № 1(29). - С. 37-45. - DOI 10.23951/2307-6127-2020-1-37-45.
- [5] Милушенко Т. В. Организация микрообучения в рамках модульного подхода обучения иностранным языкам в языковых вузах // Вестник Сибирского института бизнеса и информационных технологий. - 2022. - Т. 11, № 3. - С. 42-47. - DOI 10.24412/2225-8264-2022-3-42-47.
- [6] Михайлов И. Э., П. В. Балахнин Метод модификации последовательности интервального повторения, основанный на активности пользователя в сети интернет / И. Э. Михайлов, // Научно-технический вестник информационных технологий, механики и оптики. - 2019. - Т. 19, № 4. - С. 689-696. - DOI 10.17586/2226-1494-2019-19-4-689-696.
- [7] Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. - 4-е изд. - СПб.: Питер, 2026. - 896 с.
- [8] Брюс П., Брюс Э., Гедек П. Практическая статистика для специалистов Data Science. - 2-е изд. - СПб.: БХВ, 2021. - 352 с.
- [9] Khoshyaran M. M. Regularities in sequences of observations // Open Journal of Statistics. - 2012. - №2(4). - Pp. 408-414. - DOI: 10.4236/ojs.2012.24049.
- [10] Кнут Д. Э. Искусство программирования. Т. 2. Получисленные алгоритмы. - 3 изд. - М.: Вильямс, 2020. - 832 с.
- [11] Разработка комплексного инструмента для исследования информационной структуры текстовых и бинарных данных / В. А. Мурлина, Н. С. Щербинин, А. Б. Ярш [и др.] // Политематический сетевой электронный научный журнал Кубанского государственного аграрного университета. - 2025. - № 210. - С. 457-467. - DOI 10.21515/1990-4665-210-045.
- [12] Хотина Г. К., Амосов А. Г. Случайность распределения замедленного хрупкого растрескивания трубопроводов // Инновации и инвестиции. - 2021. - № 5. - С. 190-192.
- [13] Гриффитс И. Програмуємо на C# 8.0. Разработка приложений. - 1-е изд. - СПб.: Питер, 2021. - 944 с.
- [14] Драпеза И. С. Методическая разработка по изучению алгоритмов обработки массивов с помощью современных средств языка программирования // Современные

- информационные технологии и ИТ-образование. – 2018. – Т. 14, № 4. – С. 994-999. – DOI 10.25559/SITITO.14.201804.994-999.
- [15] Метод тестирования производительности и стресс-тестирования центральных сервисов идентификации облачных систем на примере Openstack Keystone / И. В. Богомолов, А. В. Алексиянц, А. В. Шер [и др.] // Труды Института системного программирования РАН. – 2015. – Т. 27, № 5. – С. 49-58. – DOI 10.15514/ISPRAS-2015-27(5)-4.
- [16] Беккер И. А., Якимов Е. А., Скрылев Н. П. Вычисление трудоемкости алгоритма, реализованного на языке программирования C# // Системный анализ и прикладная информатика. – 2023. – № 4. – С. 14-19. – DOI 10.21122/2309-4923-2023-4-14-19.
- [17] Овчинников В. А., Иванова Г. С. Оптимизирующие преобразования алгоритмов, использующие свойства множеств, предикатов и операций над ними // Вестник Московского государственного технического университета им. Н.Э. Баумана. Серия Приборостроение. – 2013. – № 4(93). – С. 53-66. – EDN RQDNAF.
- [18] Агибалов Г. П., Липский В. Б., Панкратова И. А. О криптографическом расширении и его реализации для Русского языка программирования // Прикладная дискретная математика. – 2013. – № 3(21). – С. 93-104. – EDN RCSET.
- [19] Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение / Р. Морелос-Сарагоса; пер. с англ. В.Б. Афанасьева. – М.: Техносфера, 2005. – 320 с.
- [20] Демин А. А., Карпунин А. А., Ганев Ю. М. Методы верификации и валидации сложных программных систем // Программные продукты и системы. – 2014. – № 4. – С. 229-233. – EDN TPOWQX.
- [21] Кокоса К. Управление памятью в .NET для профессионалов. – М.: ДМК, 2020. – 800 с.
- [22] Быков А. А., Козлов С. В. Интеллектуальное формирование наборов тестовых заданий с использованием web-платформы «smarteach» // Естественные и технические науки. – 2025. – № 5(204). – С. 147-149. – EDN GQUHIM.
- [23] Быков А. А., Козлов С. В., Исаков М. А. Использование web-платформы "smarteach" как инструмента построения образовательных карт // Естественные и технические науки. – 2024. – № 10(197). – С. 171-173. – EDN BNYICU.
- [24] Кузнецова Н. М., Т. В. Карлова, А. Н. Запольская Применение рандомизации алгоритмов шифрования в автоматизированной системе защиты информационных ресурсов промышленного предприятия // Автоматизация и моделирование в проектировании и управлении. – 2022. – № 3(17). – С. 10-15. – DOI 10.30987/2658-6436-2022-3-10-15.
- [25] Молдовян А. А., Молдовян Д. Н., Костина А. А. Рандомизация в постквантовых алгоритмах ЭЦП с секретной группой // Информатика и автоматизация. – 2025. – Т. 24, № 6. – С. 1810-1835. – DOI 10.15622/ia.24.6.9.

Implementation of stochastic randomization algorithms for educational materials in the backend of educational web applications on the platform ASP.NET Core

S.V. Kozlov, V.V. Kazantsev

Abstract – The article discusses the development and implementation of a system for randomizing educational materials for educational web applications using various algorithmic approaches. A comparative analysis of pseudorandom sequence generation methods is carried out, including standard random number generators based on the linear congruent method and cryptographically secure generators based on the RandomNumberGenerator class. A detailed implementation of the Fisher-Yates algorithm for uniform mixing of collection elements with linear computational complexity $O(n)$ and minimal memory requirements is presented. A multi-criteria randomization system has been developed that includes three strategies: standard deterministic randomization using a seed value, cryptographically secure randomization, and balanced randomization taking into account the complexity of the learning elements. A mechanism for combining hash values to generate unique seed parameters based on timestamps, user IDs, and cryptographic sources of entropy is described. Based on code fragments from the author's interactive educational web application Cotrex Cards, it is shown that the use of the Fisher-Yates algorithm ensures the equal probability of all possible permutations of elements, preventing biases in the distribution characteristic of naive random key sorting methods. The implemented system makes it possible to adapt the randomization strategy depending on pedagogical requirements, providing both a completely random order of presentation of materials and a balanced distribution of elements of varying complexity to optimize the cognitive load of students. to control access to critical areas, monitor personnel movements, and register security incidents with minimal implementation costs.

Keywords – Fisher-Yates algorithm, Cotrex Cards web application, randomization, random number generator, ASP.NET Core, educational application, cryptographic security, interval repetition, adaptive learning, computational complexity, stochastic algorithms.

REFERENCES

[1] Denishheva L. O., Safuanov I. S., Semenjachenko Ju.A. Vozmozhnosti obespechenija personalizacii obrazovanija v vuze // Vestnik Moskovskogo gorodskogo pedagogicheskogo universiteta. Serija: Informatika i informatizacija obrazovanija. - 2022. - #2 (60). - S. 72-85. DOI: 10.25688/2072-9014.2022.60.2.07.

- [2] Vorobchikova E. O. Personalizacija kak ključevoj princip obuchenija vzroslyh v uslovijah cifrovizacii // Vestnik Mininskogo universiteta. - 2025. - T. 13, # 1(50). - DOI 10.26795/2307-1281-2025-13-1-1.
- [3] Bykov A. A., Kozlov S. V. Optimizacija individual'nyh traektorij obuchenija s ispol'zovaniem web-platforny «smartteach» // Estestvennye i tehničeskije nauki. - 2025. - # 5(204). - S. 150-152. - EDN YOWJDW.
- [4] Pigarev A. Ju. Metodicheskie aspekty obuchenija matematike nesposobnyh škol'nikov // Nauchno-pedagogičeskoe obozrenie. - 2020. - # 1(29). - S. 37-45. - DOI 10.23951/2307-6127-2020-1-37-45.
- [5] Miljushenko T. V. Organizacija mikroobuchenija v ramkah modul'nogo podhoda obuchenija inostrannym jazykam v jazykovykh vuzah // Vestnik Sibirskogo instituta biznesa i informacionnyh tehnologij. - 2022. - T. 11, # 3. - S. 42-47. - DOI 10.24412/2225-8264-2022-3-42-47.
- [6] Mihajlov I. Je., P. V. Balakshin Metod modifikacii posledovatel'nosti interval'nogo povtorenija, osnovannyj na aktivnosti pol'zovatelja v seti internet / I. Je. Mihajlov, // Nauchno-tehnicheskij vestnik informacionnyh tehnologij, mehaniki i optiki. - 2019. - T. 19, # 4. - S. 689-696. - DOI 10.17586/2226-1494-2019-19-4-689-696.
- [7] Rihter D. CLR via C#. Programirovanie na platforme Microsoft .NET Framework 4.5 na jazyke C#. - 4-e izd. - SPb.: Piter, 2026. - 896 s.
- [8] Brjus P., Brjus Je., Gedek P. Praktičeskaja statistika dlja specialistov Data Science. - 2-e izd. - SPb.: BHV, 2021. - 352 s.
- [9] Khoshyaran M. M. Regularities in sequences of observations // Open Journal of Statistics. - 2012. - #2(4). - Pp. 408-414. - DOI: 10.4236/ojs.2012.24049.
- [10] Knut D. Je. Iskusstvo programirovanija. T. 2. Poluchislennye algoritmy. - 3 izd. - M.: Vil'jams, 2020. - 832 s.
- [11] Razrabotka kompleksnogo instrumenta dlja issledovanija informacionnoj struktury tekstovyh i binarnykh dannyh / V. A. Murlina, N. S. Shherbinin, A. B. Jarysh [i dr.] // Politematicheskij setevoj jelektronnyj nauchnyj zhurnal Kubanskogo gosudarstvennogo agramogo universiteta. - 2025. - # 210. - S. 457-467. - DOI 10.21515/1990-4665-210-045.
- [12] Hotina G. K., Amosov A. G. Sluchajnost' raspredelenija zamedlennogo hrupkogo rastreskivaniya truboprovodov // Innovacii i investicii. - 2021. - # 5. - S. 190-192.
- [13] Griffiths I. Programmiruem na C# 8.0. Razrabotka prilozhenij. - 1-e izd. - SPb.: Piter, 2021. - 944 s.
- [14] Drapeza I. S. Metodicheskaja razrabotka po izucheniju algoritmov obrabotki massivov s pomoshh'ju sovremennyh sredstv jazyka programirovanija // Sovremennye informacionnye tehnologii i IT-obrazovanie. - 2018. - T. 14, # 4. - S. 994-999. - DOI 10.25559/SITITO.14.201804.994-999.
- [15] Metod testirovanija proizvoditel'nosti i stress-testirovanija central'nyh servisov identifikacii oblachnyh sistem na primere Openstack Keystone / I. V. Bogomolov, A. V. Aleksijanc, A. V. Sher [i dr.] // Trudy Instituta sistemnogo programirovanija RAN. - 2015. - T. 27, # 5. - S. 49-58. - DOI 10.15514/ISPRAS-2015-27(5)-4.

- [16] Bekker I. A., Jakimov E. A., Skrylev N. P. Vychislenie trudoemkosti algoritma, realizovannogo na jazyke programirovaniya S# // Sistemnyj analiz i prikladnaja informatika. – 2023. – # 4. – S. 14-19. – DOI 10.21122/2309-4923-2023-4-14-19.
- [17] Ovchinnikov V. A., Ivanova G. S. Optimizirujushhie preobrazovaniya algoritmov, ispol'zujushhie svojstva mnozhestv, predikativ i operacij nad nimi // Vestnik Moskovskogo gosudarstvennogo tehničeskogo universiteta im. N.Je. Baumana. Serija Priborostroenie. – 2013. – # 4(93). – S. 53-66. – EDN RQDNAF.
- [18] Agibalov G. P., Lipskij V. B., Pankratova I. A. O kriptograficheskom rasshirenii i ego realizacii dlja Russkogo jazyka programirovaniya // Prikladnaja diskretnaja matematika. – 2013. – # 3(21). – S. 93-104. – EDN RCESET.
- [19] Morelos-Saragosa R. Iskustvo pomehoustojchivogo kodirovaniya. Metody, algoritmy, primenenie / R. Morelos-Saragosa; per. s angl. V.B. Afanas'eva. – M.: Tehnosfera, 2005. – 320 s.
- [20] Demin A. A., Karpunin A. A., Ganev Ju. M. Metody verifikacii i validacii slozhnyh programnyh sistem // Programmnye produkty i sistemy. – 2014. – # 4. – S. 229-233. – EDN TPOWQX.
- [21] Kokosa K. Upravlenie pamjat'ju v .NET dlja professionalov. – M.: DMK, 2020. – 800 s.
- [22] Bykov A. A., Kozlov S. V. Intellektual'noe formirovanie naborov testovyh zadaniy s ispol'zovaniem web-platfomy «smarteach» // Estestvennye i tehničeskije nauki. – 2025. – # 5(204). – S. 147-149. – EDN GQUHIM.
- [23] Bykov A. A., Kozlov S. V., Isakov M. A. Ispol'zovanie web-platfomy "smarteach" kak instrumenta postroenija obrazovatel'nyh kart // Estestvennye i tehničeskije nauki. – 2024. – # 10(197). – S. 171-173. – EDN BHYICU.
- [24] Kuznecova N. M., T. V. Karlova, A. N. Zapol'skaja. Primenenie randomizacii algoritmov shifrovaniya v avtomatizirovannoj sisteme zashhity informacionnyh resursov promyshlennogo predpriyatija // Avtomatizacija i modelirovanie v proektirovanii i upravlenii. – 2022. – # 3(17). – S. 10-15. – DOI 10.30987/2658-6436-2022-3-10-15.
- [25] Moldovjan A. A., Moldovjan D. N., Kostina A. A. Randomizacija v postkvantovyh algoritmah JeCP s sekretnoj gruppoj // Informatika i avtomatizacija. – 2025. – T. 24, # 6. – S. 1810-1835. – DOI 10.15622/ia.24.6.9.