

## 1. APPLICATION OF THE LAW OF MINIMUM FOR TCP

### Закон минимума и анализ различий для приоритизации регрессионных тестов

**Annotation.** *We present the classical VIKOR multi-criteria decision-making method together with two important extensions: interval VIKOR and fuzzy VIKOR. We discuss their theoretical foundations, applicability, advantages and limitations, and provide comparative insight into decision stability under uncertainty. A computational example illustrates interval and fuzzy modifications of VIKOR evaluation and ranking.*

**Keywords.** *VIKOR, MCDM, interval analysis, fuzzy sets.*

**Аннотация.** *В работе рассматривается классический метод многокритериального принятия решений VIKOR, а также две его ключевые модификации: интервальный и нечеткий VIKOR. Излагаются теоретические основы методов, области применимости, особенности и ограничения, а также проводится сравнение устойчивости решений в условиях неопределённости. На иллюстративном примере демонстрируется выполнение и результаты интервальной и нечеткой модификации VIKOR.*

**Ключевые слова.** *VIKOR, многокритериальное принятие решений, интервальный анализ, нечеткие множества.*

#### 1.1. Введение

Метод VIKOR (*VIseKriterijumska Optimizacija I Kompromisno Resenje*) был предложен Сергием Опричевичем как подход к многокритериальному принятию решений, ориентированный на нахождение *компромиссного решения*, максимально близкого к идеальной точке. В отличие от методов, стремящихся к парето-оптимальным решениям, VIKOR фокусируется на ситуациях, где компромисс между критериями является предпочтительным.

Однако классическая версия метода предполагает, что значения критериев заданы в виде точных чисел. В реальных задачах часто присутствуют различные

формы неопределённости: интервальные оценки показателей, экспертные оценки в форме языковых термов, нечеткие треугольные числа и др. Это приводило к необходимости расширить VIKOR.

Существуют две важные модификации:

- **интервальный VIKOR**, предложенный в работе [4], адаптирующий алгоритм к критериям, представленным через интервалы;
- **fuzzy VIKOR**, описанный, например, в работах [2; 5], позволяющий работать с лингвистическими и нечеткими данными.

Сравнительный анализ различных версий VIKOR выполнен в статье [1], где показано, что интервальная и нечеткая модификации позволяют повысить устойчивость решений при наличии неопределенности.

Цель данной работы — систематизировать основные идеи интервальной и нечеткой модификаций VIKOR, показать отличие от классического алгоритма и продемонстрировать работу интервального и нечеткого VIKOR на примере, написанном на языке программирования python.

## 1.2. Обзор метода VIKOR

Метод VIKOR относится к классу многокритериальных подходов принятия решений и был предложен Опричевичем как способ получения компромиссного решения в ситуациях, где необходимо учитывать несколько конфликтующих критериев. Первоначально метод создавался для инженерных задач, требующих выбора оптимального варианта среди множества альтернатив с разнородными характеристиками. Основная идея метода заключается в нахождении решения, минимизирующего расстояние до идеальной точки и одновременно обеспечивающего баланс между улучшением средних показателей и уменьшением максимального отклонения. Метод основан на компромиссном программировании, но отличается тем, что явно использует процедуру поиска решения, приемлемого для большинства заинтересованных сторон, что делает его особенно полезным в задачах, предполагающих наличие противоречивых критериев.

VIKOR применяется в ситуациях, где требуется учитывать не только максимальную близость альтернативы к идеальному варианту, но и степень её доминирования по самому неблагоприятному критерию. Поэтому метод получил

широкое применение в инженерии, инвестиционном анализе, экологии, управлении проектами и других областях, где невозможно достичь одновременной оптимизации всех целей. Преимуществом метода является способность сочетать стратегии минимизации наихудшего отклонения и максимизации интегральной пользы, что позволяет получать взвешенное компромиссное решение.

### 1.2.1. Математическая постановка задачи

Пусть задано множество альтернатив  $A = \{A_1, A_2, \dots, A_m\}$ , каждая из которых оценивается по набору критериев  $C = \{c_1, c_2, \dots, c_n\}$ . Определена матрица оценок  $f_{ij}$ , где  $f_{ij}$  — значение альтернативы  $A_i$  по критерию  $c_j$ . Для каждого критерия задан тип: выгодный (benefit), для которого большие значения предпочтительны, или затратный (cost), для которого предпочтительны меньшие значения. Также каждому критерию назначен весовой коэффициент  $w_j$ , удовлетворяющий условию

$$\sum_{j=1}^n w_j = 1.$$

Для каждого критерия определяются лучшие и худшие значения среди всех альтернатив. Для выгодных критериев применяются выражения

$$f_j^* = \max_i f_{ij}, \quad f_j^- = \min_i f_{ij},$$

а для затратных критериев

$$f_j^* = \min_i f_{ij}, \quad f_j^- = \max_i f_{ij}.$$

Затем вычисляются две меры отклонения альтернативы от идеального решения. Первая мера представляет собой взвешенную сумму отклонений:

$$S_i = \sum_{j=1}^n w_j \frac{f_j^* - f_{ij}}{f_j^* - f_j^-}.$$

Вторая мера характеризует максимальное взвешенное отклонение по одному критерию:

$$R_i = \max_{1 \leq j \leq n} \left\{ w_j \frac{f_j^* - f_{ij}}{f_j^* - f_j^-} \right\}.$$

Для получения компромиссного решения определяется интегральный индекс  $Q_i$ :

$$Q_i = v \cdot \frac{S_i - S^*}{S^- - S^*} + (1 - v) \cdot \frac{R_i - R^*}{R^- - R^*},$$

где

$$S^* = \min_i S_i, \quad S^- = \max_i S_i, \quad R^* = \min_i R_i, \quad R^- = \max_i R_i,$$

а параметр  $v \in [0,1]$  отражает предпочтения относительно стратегий компромисса. Значение  $v = 0.5$  соответствует равному учёту обеих мер, приближение  $v \rightarrow 1$  подчеркивает ориентацию на минимизацию суммарных отклонений, а  $v \rightarrow 0$  — на минимизацию наибольшего отклонения.

Итоговое решение определяется альтернативой с минимальным значением  $Q_i$ , однако процедура выбора включает дополнительные условия устойчивости. Если различие между лучшими альтернативами недостаточно велико или ранжирование по метрикам  $S$  и  $R$  противоречит индексу  $Q$ , формируется не одна альтернатива, а набор компромиссных решений.

### 1.3. Интервальная модификация метода VIKOR

Как уже упоминалось в 1.2, классический метод VIKOR предполагает, что оценки всех альтернатив по каждому критерию представлены точечными значениями. Однако в реальных задачах многокритериального анализа нередко встречаются ситуации, когда значения критериев заданы неточно, являются приближёнными или варьируются в некотором допустимом диапазоне. Такие ситуации типичны для инженерных, экономических и экологических задач, в которых невозможно получить точные числа из-за экспертной неопределённости, вариативности данных и измерительных ошибок.

Для учёта неопределённости авторы [4] предложили интервальную модификацию метода VIKOR, в которой вместо точечных значений используются интервальные оценки

$$f_{ij} = [ \underline{f}_{ij}, \overline{f}_{ij} ],$$

где  $\underline{f}_{ij}$  и  $\overline{f}_{ij}$  обозначают нижнюю и верхнюю границы возможного значения критерия. Такой подход позволяет работать с более гибкой моделью данных и принимать решения, устойчивые к колебаниям входной информации.

### 1.3.1. Математическая постановка интервального VIKOR

Пусть задано множество альтернатив  $A = \{A_1, \dots, A_m\}$ , множество критериев  $C = \{c_1, \dots, c_n\}$  и интервальная матрица оценок

$$f_{ij} = [ \underline{f}_{ij}, \bar{f}_{ij} ].$$

Каждый критерий относится к типу *выгодный* или *затратный*, а веса  $w_j$  удовлетворяют условию  $\sum_{j=1}^n w_j = 1$ .

Для интервальных данных определяются *интервальные идеальные* и *анти-идеальные точки*. Для выгодного критерия:

$$f_j^* = \left[ \max_i \underline{f}_{ij}, \max_i \bar{f}_{ij} \right], \quad f_j^- = \left[ \min_i \underline{f}_{ij}, \min_i \bar{f}_{ij} \right].$$

Для затратного критерия:

$$f_j^* = \left[ \min_i \underline{f}_{ij}, \min_i \bar{f}_{ij} \right], \quad f_j^- = \left[ \max_i \underline{f}_{ij}, \max_i \bar{f}_{ij} \right].$$

Далее необходимо определить интервальные меры отклонения. Интервальный нормализованный разрыв альтернативы  $A_i$  по критерию  $c_j$  определяется как

$$D_{ij} = \left[ \frac{f_j^{*(low)} - \bar{f}_{ij}}{f_j^{*(low)} - f_j^{-(high)}}, \frac{f_j^{*(high)} - \underline{f}_{ij}}{f_j^{*(high)} - f_j^{-(low)}} \right],$$

где использование противоположных концов интервалов обеспечивает *наиболее пессимистичную* и *наиболее оптимистичную* оценки, согласуясь с подходом Sayadi et al.

Интервальные показатели  $S_i$  и  $R_i$  вычисляются как:

$$S_i = \left[ \sum_{j=1}^n w_j D_{ij}^{(low)}, \sum_{j=1}^n w_j D_{ij}^{(high)} \right],$$
$$R_i = \left[ \max_j w_j D_{ij}^{(low)}, \max_j w_j D_{ij}^{(high)} \right].$$

### 1.3.2. Особенности метода

Интервальный VIKOR сохраняет структуру исходного метода, но адаптирует все вычисления к интервальной арифметике. Основные особенности:

- вместо точек используются интервалы значений, что делает метод чувствительным к неопределённости;
- нормализация использует “наиболее широкое” сочетание концов интервалов, что гарантирует корректную оценку в условиях неполных данных;
- результаты ранжирования также представляются интервалами и требуют процедуры сравнения интервальных чисел;
- метод может порождать частичные или нестрогие порядки, когда интервалы  $Q_i$  альтернатив пересекаются.

### ***1.3.3. Недостатки и ограничения***

Несмотря на преимущества, интервальная модификация имеет ряд недостатков:

- усложнение вычислений из-за необходимости поддерживать интервальную арифметику;
- увеличение числа случаев, когда альтернативы оказываются несравнимыми из-за пересечения интервалов;
- зависимость качества результата от корректного задания интервалов экспертами.

Тем не менее модификация доказала свою эффективность в задачах с высокой неопределённостью и стала основой для дальнейших расширений метода, включая fuzzy модификацию [2; 5], о которой пойдет речь в 1.4.

### ***1.3.4. Реализация интервальной модификации на Python***

Для иллюстрации работы интервальной модификации метода VIKOR была реализована функция `interval_vikor`, учитывающая интервальные оценки альтернатив по каждому критерию, а также веса критериев и их тип (выгода/затраты).

Ниже приведены результаты работы функции для приведённого примера.

**Метрики для альтернатив:**

Ранжирование альтернатив по возрастанию  $Q$ :

$$A4 \rightarrow A2 \rightarrow A1 \rightarrow A3$$

Лучшая альтернатива: **A4** Условия приемлемости: **выполнены**

Таблица 1.1

Интервальные идеальные и надирные точки		
Критерий	$f^*$ (идеальная точка)	$f^-$ (надирная точка)
1	[0.800, 1.000]	[0.500, 0.700]
2	[0.800, 1.000]	[0.500, 0.700]
3	[0.500, 0.700]	[0.800, 1.000]

Таблица 1.2

Интервальные показатели S, R и Q для альтернатив				
Альтернатива	S-интервал	R-интервал	Q-интервал	Q_center
A1	[0.6667, 0.6667]	[0.3000, 0.3000]	[0.7500, 0.7500]	0.7500
A2	[0.4000, 0.4000]	[0.3000, 0.3000]	[0.3500, 0.3500]	0.3500
A3	[0.6000, 0.6000]	[0.4000, 0.4000]	[0.9000, 0.9000]	0.9000
A4	[0.3333, 0.3333]	[0.2000, 0.2000]	[0.0000, 0.0000]	0.0000

### 1.3.5. Анализ чувствительности к коэффициенту $\nu$

Ниже приведён анализ влияния коэффициента стратегии  $\nu$  на выбор альтернатив:

Таблица 1.3

Анализ чувствительности альтернатив к коэффициенту $\nu$		
$\nu$	Лучшая альтернатива	Q_центры (A1, A2, A3, A4)
0.3	A4	[0.650, 0.410, 0.940, 0.000]
0.5	A4	[0.750, 0.350, 0.900, 0.000]
0.7	A4+A2	[0.850, 0.290, 0.860, 0.000]
1.0	A4+A2	[1.000, 0.200, 0.800, 0.000]

Данные показывают, что при изменении  $\nu$  меняется компромисс между групповой стратегией (S) и наибольшим отклонением (R), что отражается на выборе лучшей альтернативы.

## 1.4. Нечёткая (fuzzy) модификация метода VIKOR

В ряде практических задач исходные данные содержат неопределённость, которую невозможно корректно описать только интервалами. В таких случаях

применяется нечеткая (fuzzy) модификация метода VIKOR, предложенная в [3]. В этой версии критерии и веса допускают неопределённость, выражаемую *треугольными нечеткими числами* (TFN). Нечеткие оценки позволяют моделировать неточность экспертных суждений, а также различать нижнюю, наиболее вероятную и верхнюю оценку каждого параметра.

Треугольное нечеткое число определяется как

$$\tilde{x} = (x^L, x^M, x^U),$$

где  $x^L$ ,  $x^M$  и  $x^U$  соответствуют нижней, модальной и верхней оценкам соответственно. Все операции в алгоритме выполняются покомпонентно, что обеспечивает корректность результата в нечеткой среде.

#### 1.4.0.1. Определение нечетких идеального и антиидеального решений

Аналогично классическому VIKOR для каждого критерия определяются лучшие и худшие значения. Отличие заключается в том, что каждая оценка является TFN, а операции max и min применяются к соответствующим компонентам.

Для выгодных критериев:

$$\tilde{f}_j^* = \left( \max_i f_{ij}^L, \max_i f_{ij}^M, \max_i f_{ij}^U \right), \quad \tilde{f}_j^- = \left( \min_i f_{ij}^L, \min_i f_{ij}^M, \min_i f_{ij}^U \right).$$

Для затратных критериев:

$$\tilde{f}_j^* = \left( \min_i f_{ij}^L, \min_i f_{ij}^M, \min_i f_{ij}^U \right), \quad \tilde{f}_j^- = \left( \max_i f_{ij}^L, \max_i f_{ij}^M, \max_i f_{ij}^U \right).$$

#### 1.4.0.2. Нормирование нечетких расстояний

Нечеткое расстояние альтернативы от идеального решения также является треугольным числом. В работе [3] предложена нормировка вида

$$\tilde{d}_{ij} = \left( \frac{f_j^L - f_{ij}^U}{f_j^U - f_j^{-L}}, \frac{f_j^M - f_{ij}^M}{f_j^M - f_j^{-M}}, \frac{f_j^U - f_{ij}^L}{f_j^L - f_j^{-U}} \right),$$

что гарантирует сохранение структуры TFN.

#### 1.4.0.3. Нечеткие показатели $S_i$ и $R_i$

Аналоги интегральной и максимальной меры отклонения определяются как



$$\tilde{S}_i = \sum_{j=1}^n (\tilde{w}_j \otimes \tilde{d}_{ij}),$$

$$\tilde{R}_i = \max_j (\tilde{w}_j \otimes \tilde{d}_{ij}),$$

где  $\otimes$  обозначает покомпонентное умножение TFN, а максимум по критериям также определяется покомпонентно:

$$\max((a^L, a^M, a^U), (b^L, b^M, b^U)) = (\max(a^L, b^L), \max(a^M, b^M), \max(a^U, b^U)).$$

#### 1.4.0.4. Дефаззификация и вычисление итогового показателя

Поскольку результатом являются нечеткие числа, необходимо дефаззифицировать  $\tilde{S}_i$  и  $\tilde{R}_i$ . В [3] применяется классическая дефаззификация треугольного числа:

$$\text{Defuzz}(\tilde{x}) = \frac{x^L + x^M + x^U}{3}.$$

Таким образом,

$$S_i = \text{Defuzz}(\tilde{S}_i), \quad R_i = \text{Defuzz}(\tilde{R}_i).$$

После этого итоговый компромиссный показатель вычисляется по стандартной формуле VIKOR:

$$Q_i = v \frac{S_i - S^*}{S^- - S^*} + (1 - v) \frac{R_i - R^*}{R^- - R^*},$$

где

$$S^* = \min_i S_i, \quad S^- = \max_i S_i, \quad R^* = \min_i R_i, \quad R^- = \max_i R_i,$$

а  $v \in [0,1]$  — параметр компромисса.

#### 1.4.0.5. Итоговое ранжирование

Окончательные решения выбираются по тем же критериям приемлемости преимущества и стабильности, что и в оригинальном методе VIKOR. Отличие заключается только в том, что все сравнения проводятся над дефаззифицированными значениями.

## 1.5. Обзор существующих подходов к ТСП

Задача приоритизации тестовых случаев активно исследуется на протяжении последних десятилетий. Наиболее распространёнными являются следующие классы методов.

### 1.5.1. *Coverage-based методы*

Coverage-based ТСП методы используют информацию о покрытии кода тестами. Предполагается, что тесты, покрывающие больше кода (особенно изменённого), с большей вероятностью обнаружат дефекты.

Классический **Total** метод сортирует тесты по убыванию общего покрытия — в первую очередь запускаются тесты, покрывающие наибольшее число элементов (строк, ветвей, методов и т. п.). Аналогично, **Total-Diff** фокусируется на покрытии именно изменённых (impacted) элементов.

Метод **Additional** выбирает тесты итеративно: на каждом шаге выбирается тест, добавляющий максимальное количество *ещё не покрытых* элементов. Его дифф-ориентированный вариант **Additional-Diff** работает только с множеством изменённых элементов программы.

Преимуществом таких методов является простота и широкая поддержка инструментами покрытия. Однако они не учитывают структуру зависимостей между компонентами и предполагают, что все покрытые элементы одинаково важны.

### 1.5.2. *Diff-based методы*

Diff-based подходы используют информацию о различиях между версиями программы: на основе diff (например, по байт-коду) выделяются изменённые файлы, классы или методы. Далее тесты сортируются по тому, какое количество изменённых элементов они покрывают.

Методы Total-Diff и Additional-Diff относятся к этому классу, но существуют и более сложные варианты, интегрирующие дифф-информацию с историей предыдущих запусков тестов, покрытием и другими факторами.

### ***1.5.3. History-based методы***

Исторические методы приоритизации используют информацию о прошлых запусках тестов: например, тесты, чаще всего обнаруживавшие дефекты в прошлых версиях, могут запускаться раньше. Такие подходы хорошо работают при наличии длинной истории изменений и запусков, но малоприменимы для новых проектов.

### ***1.5.4. Ограничения традиционных подходов***

Основные ограничения классических методов ТСП можно сформулировать следующим образом:

- отсутствие моделирования *распространения изменений* через граф вызовов: изменение в одном методе может косвенно повлиять на поведение множества других методов;
- равное отношение ко всем изменённым элементам без учёта того, насколько логика метода зависит от его входных параметров;
- игнорирование *похожести* тестов: два теста с почти идентичным покрытием вносят мало дополнительной информации, но могут выполняться подряд.

Для преодоления этих ограничений в рассматриваемой статье предложено объединить анализ потока управления, анализ различий, модель цепи Маркова и закон минимума.

## **1.6. Методология работы**

### ***1.6.1. Общая архитектура анализа***

Предлагаемый в статье подход можно представить как конвейер, включающий несколько этапов:

- А. анализ изменений (diff) на уровне байт-кода и вычисление *ratio of change* для методов;
- В. построение графа вызовов (Call Graph) и графов потока управления (Control Flow Graph, CFG);

- C. выполнение *forward slicing* по CFG и оценка чувствительности методов к параметрам;
- D. построение цепи Маркова на множестве методов и вычисление вектора влияния (*impact vector*);
- E. вычисление метрик LoM-Score и Dis-LoM-Score для тестов;
- F. упорядочивание тестов в соответствии с полученными метриками.

В качестве инструментов используются:

- **Soot** — фреймворк для анализа Java-байткода, позволяющий строить CFG;
- **java-callgraph** — утилита для построения графа вызовов;
- **reJ** — инструмент для сравнения байт-кода и оценки *ratio of change*.

### 1.6.2. *Forward slicing и оценка чувствительности*

Для каждого метода программы строится граф потока управления — CFG. Вершины CFG соответствуют отдельным инструкциям или операторам, а рёбра обозначают возможные переходы управления (следующая инструкция, ветвления, выход из цикла и т. п.).

Далее выполняется *forward slicing* относительно параметров метода. На вход slicing-процедуры подаются:

- начальные узлы (источники) — параметры метода;
- CFG рассматриваемого метода.

Алгоритм находит все инструкции, которые напрямую используют параметры, а затем распространяет зависимость вперёд по CFG, учитывая передачу значений через переменные. В результате получается подграф  $pCFG_{m_i} \subseteq CFG_{m_i}$ , содержащий только те инструкции, которые зависят от параметров метода  $m_i$ .

На основе этого вводится метрика чувствительности метода к параметрам:

$$P(m_i) = \frac{|pCFG_{m_i}|}{|CFG_{m_i}|},$$

где  $|pCFG_{m_i}|$  — количество вершин в подграфе, а  $|CFG_{m_i}|$  — общее количество вершин CFG. Чем выше  $P(m_i)$ , тем большая часть тела метода зависит от его параметров, и тем потенциально сильнее влияние изменений входных данных на поведение метода.

### 1.6.3. Модель цепи Маркова

Следующий элемент методологии — построение модели цепи Маркова на множество методов. Состояния цепи соответствуют методам программы, а переходы описывают возможное распространение влияния изменений по графу вызовов.

Пусть метод  $A$  вызывает метод  $B$ . Тогда в матрице переходов  $T$  вводится элемент  $T_{AB} > 0$ . Вес перехода может зависеть, в частности, от чувствительности метода  $B$  (значения  $P(B)$ ). Далее веса для каждого состояния нормируются таким образом, чтобы сумма по строке была равна единице, то есть матрица  $T$  становится стохастической.

Для инициализации цепи используется вектор  $I$ , основанный на данных diff. Инструмент reJ позволяет оценить *ratio of change* для каждого метода:

$$roc(m_i) = \frac{\text{\#изменённых инструкций}}{\text{\#всех инструкций}}.$$

После нормирования значений  $roc(m_i)$  получается начальный вектор  $I$ , отражающий исходную “силу” изменений.

Вектор влияния (impact vector) определяется как:

$$Impact = I \times T,$$

где произведение понимается в обычном матричном смысле. Компонента  $Impact(m_i)$  интерпретируется как вероятность того, что метод  $m_i$  будет затронут изменениями напрямую или через транзитивные зависимости.

### 1.6.4. Метрика LoM-Score

Основой LoM-Score является закон минимума, известный из биологии (закон Либиха): рост организма ограничен фактором, находящимся в дефиците. Переноса эту идею на тестирование, можно сказать, что эффективность теста ограничивается его “самым слабым” покрытием.

Для каждого теста  $t$  рассматривается множество покрываемых им методов  $M_t$ . Для всех  $m \in M_t$  берутся значения  $Impact(m)$ . Далее выполняются следующие шаги:

А. значения  $Impact(m)$  сортируются по возрастанию;

В. из отсортированного набора выбирается нижний квартиль (first quartile,  $Q_1$ );

С. LoM-Score определяется как среднее значение элементов нижнего квартиля.

Таким образом, если среди покрываемых тестом методов есть “слабые” с точки зрения вероятности влияния, LoM-Score будет ниже. Тесты с высоким LoM-Score считаются более перспективными для раннего запуска.

### ***1.6.5. Метрика Dis-LoM-Score***

Несмотря на полезность LoM-Score, возможна ситуация, когда несколько тестов с высокими значениями LoM имеют почти одинаковое покрытие методов. Запуск таких тестов подряд приводит к дублированию работы и задержке обнаружения дефектов в других частях системы.

Для учёта непохожести тестов вводится Dis-LoM-Score. Для оценки сходства двух тестов  $t_i$  и  $t_j$  используется коэффициент Жаккара:

$$Sim(t_i, t_j) = \frac{|Cover_i \cap Cover_j|}{|Cover_i \cup Cover_j|},$$

где  $Cover_i$  и  $Cover_j$  — множества методов, покрываемых соответствующими тестами.

При выборе следующего теста в порядке запуска для теста  $t$  рассчитывается:

$$DisLoM(t) = LoM(t) \cdot \left(1 - \max_{t' \in Selected} Sim(t, t')\right),$$

где *Selected* — множество уже выбранных тестов. Тесты, сильно похожие на уже выполненные (имеющие высокое сходство по Жаккару), получают дополнительный штраф и смещаются вниз в порядке исполнения.

## **1.7. Экспериментальная часть**

### ***1.7.1. Данные и настройки экспериментов***

Экспериментальная оценка метода проведена авторами статьи на основе набора Defects4J, включающего реальные Java-проекты с искусственно сохранёнными дефектными версиями. В частности, рассматривались следующие проекты:

- Jsoup, Gson, Csv;
- JacksonDatabind, JacksonXml, JacksonCore;
- Compress, Time, Codec, Lang.

Для генерации искусственных дефектов применялось мутационное тестирование с использованием инструмента Major. Качество различных стратегий приоритизации оценивалось по метрике APFD (Average Percentage of Faults Detected), показывающей, насколько быстро по порядку тестов обнаруживаются дефекты.

В качестве базовых методов сравнения (baseline) использовались:

- Total и Additional;
- Total-Diff и Additional-Diff;
- случайные порядки (Random).

### ***1.7.2. Результаты для LoM и Dis-LoM***

В таблице 1.4 приведены значения APFD для методов LoM и Dis-LoM для части проектов Defects4J (по данным оригинальной статьи).

*Таблица 1.4*

<b>Сравнение LoM и Dis-LoM по APFD</b>			
Проект	LoM	Dis-LoM	$\Delta$
Csv	0,9643	0,9643	0,0000
JacksonDatabind	0,9921	0,9924	+0,0003
JacksonXml	0,8795	0,9399	+0,0604
Time	0,9791	0,9915	+0,0124
Codec	0,7549	0,9540	+0,1991
JacksonCore	0,9715	0,9729	+0,0014
Lang	0,9938	0,9938	0,0000

Можно отметить, что Dis-LoM никогда не показывает результаты хуже, чем LoM. На некоторых проектах (Csv, Lang) значения APFD совпадают, на других наблюдается небольшое улучшение. Наиболее заметный прирост достигается на проектах Codec и JacksonXml, где Dis-LoM значительно смещает полезные тесты вверх по порядку, устраняя дублирование покрытия.

### 1.7.3. Сравнение с базовыми методами

В среднем по рассматриваемым проектам значения APFD для различных методов приоритизации распределились следующим образом (по данным статьи):

Таблица 1.5

Средние значения APFD для разных методов

Метод	Средний APFD
Additional-Diff	0,9805
Dis-LoM	0,9727
Total-Diff	0,9625
LoM	0,9336
Additional	0,8656
Random (avg)	0,8236
Total	0,8062

Из таблицы 1.5 видно, что методы, учитывающие diff (Additional-Diff, Total-Diff), демонстрируют более высокие значения APFD по сравнению с классическими Total и Additional. Предложенный метод Dis-LoM оказывается близок к лучшему baseline Additional-Diff и превосходит остальные методы, в том числе стандартные coverage-based и случайный порядок.

### 1.7.4. Краткий анализ результатов

Результаты показывают, что:

- интеграция информации о распространении изменений (через цепь Маркова) и закона минимума позволяет более точно оценивать важность тестов по отношению к изменённому коду;
- учёт различий между тестами (Dis-LoM) уменьшает дублирование и ускоряет обнаружение дефектов в разных частях системы;
- Dis-LoM демонстрирует стабильное улучшение по сравнению с LoM и конкурентоспособен с лучшими diff-ориентированными baseline.



## 1.8. Выводы

В работе рассмотрен подход к приоритизации регрессионных тестов, основанный на сочетании анализа различий, закона минимума и модели цепей Маркова. В отличие от классических coverage-based методов, данный подход:

- моделирует распространение влияния изменений по графу вызовов;
- учитывает чувствительность методов к параметрам через forward slicing;
- опирается на закон минимума для выявления “узких мест” в покрытии;
- снижает избыточность тестов за счёт анализа непохожести покрытия.

Экспериментальные результаты, полученные авторами исходной статьи на наборе Defects4J, показывают, что метрика Dis-LoM обеспечивает высокие значения APFD и сопоставима с лучшими diff-ориентированными методами, превосходя при этом классические Total и Additional, а также случайный порядок.

Перспективными направлениями развития подхода являются:

- расширение анализа на другие языки программирования и типы проектов;
- интеграция исторической информации о сбоях тестов;
- оптимизация вычислительной стоимости построения графов и модели цепи Маркова для очень больших систем.

## Библиографический список

1. *Chatterjee P., Chakraborty S.* A comparative analysis of VIKOR method and its variants // *Decision Science Letters*. — 2016. — Т. 5, № 4. — С. 469—486. — DOI 10.5267/j.dsl.2016.5.004.
2. *Liu P., Qin X.* An Extended VIKOR Method for Decision Making Problem with Interval-Valued Linguistic Intuitionistic Fuzzy Numbers Based on Entropy // *Informatica*. — 2017. — Т. 28, № 4. — С. 665—685. — DOI 10.15388/Informatica.2017.151.
3. *Opricovic S.* Fuzzy VIKOR with an application to water resources planning // *Expert Systems with Applications*. — 2011. — Т. 38, № 10. — С. 12983—12990. — DOI 10.1016/j.eswa.2011.04.097. — URL: <https://www.sciencedirect.com/science/article/pii/S0957417411006245>.

4. *Sayadi M. K., Heydari M., Shahanaghi K.* Extension of VIKOR method for decision making problem with interval numbers // *Applied Mathematical Modelling*. — 2009. — T. 33, № 5. — C. 2257—2262. — DOI 10.1016/j.apm.2008.06.002.
5. *Wan S.-P.* The extended VIKOR method for multi-attribute group decision making with triangular intuitionistic fuzzy numbers // *Knowledge-Based Systems*. — 2013. — T. 52. — C. 65—77. — DOI 10.1016/j.knosys.2013.06.019.