ПРОГРАММИРОВАНИЕ, 2011, № 3, с. 3–41

ТЕСТИРОВАНИЕ, ВЕРИФИКАЦИЯ И ВАЛИДАЦИЯ ПРОГРАММ

УДК 681.3.06

ОБЗОР МЕТОДОВ ПОСТРОЕНИЯ ПОКРЫВАЮЩИХ НАБОРОВ

© **2011 г.** В.В. Кулямин, А.А. Петухов
Институт системного программирования РАН
109004 г. Москва, ул. Солженицына, 25
E-mail: kuliamin@ispras.ru
Поступила в редакцию 03.10.2010 г.

Работа представляет собой обзор методов построения покрывающих наборов, используемых при генерации тестов для интерфейсов с большим количеством параметров. Анализируются область применения этих методов и использующиеся в них алгоритмы. Указывается ряд их характеристик, включая временную сложность и оценку требуемой памяти. В работе приводятся прямые, рекурсивные, оптимизационные, генетические алгоритмы, а также алгоритмы поиска с возвратом, используемые для построения покрывающих наборов. В работе представлены эвристики, позволяющие сократить наборы без потери полноты, и очерчиваются области применимости этих эвристик.

1. ВВЕДЕНИЕ

Высокая сложность современных программных систем и важность решаемых ими задач делают актуальной проблему проверки правильности работы таких систем, т.е. получения подтверждения того, что во всех возможных ситуациях они работают в соответствии с требованиями к ним. Для такой проверки чаще всего используется тестирование – анализ поведения тестируемой системы в конечном наборе специально создаваемых тестовых ситуаций.

Чтобы обеспечить качество и полноту проводимых проверок, нужно иметь как можно больше тестов, но полное тестирование неосуществимо, поскольку число возможных ситуаций, в которых необходимо проверить поведение реальных систем, практически бесконечно. Поэтому на практике стараются выбрать небольшое количество тестовых ситуаций так, чтобы по поведению системы в этих ситуациях можно было судить о ее поведении в целом. Обычно при этом тестовые ситуации разбиваются на классы эквивалентности, так, чтобы поведение тестируемой системы в рамках одного класса менялось слабо, а при переходе между классами – достаточно сильно. После этого полнота тестирования определяется как степень покрытия тестами выделенных классов ситуаций. В ходе выполнения тестов анализируется поведение системы и проводится проверка на его соответствие требованиям.

Поскольку тестирование должно быть ограничено, количество выделяемых классов всегда конечно. Однако оно может быть слишком большим для эффективного проведения тестирования в рамках заданного проекта. Часто возможные ситуации и сценарии поведения тестируемой системы классифицируются по некоторому набору факторов, характеристик или параметров, каждый из которых может принимать конечное множество значений. Различные классы ситуаций соответствуют при этом всем возможным комбинациям значений выделенных факторов.

Примеры факторов, используемых при классификациях ситуаций:

- успешность или неуспешность выполнения некоторой операции;
- ветвление в коде тестируемого компонента,

значением соответствующего фактора является выполнение ветки if или else в этом ветвлении;

- альтернатива (выбор из нескольких возможных вариантов) правила грамматики при тестировании на основе грамматик [1,2], значениями здесь являются возможные способы разрешения альтернативы;

- категория, при тестировании на основе разбиения на категории [3].

Во многих таких случаях для проведения качественного тестирования требуется проверить взаимодействие факторов между собой, их совместное влияние на поведение системы. Для выполнения такой проверки необходимо комбинировать значения факторов так, чтобы, с одной стороны, получить достаточно аккуратное тестирование, а с другой, – не увеличить чрезмерно тестовый набор, поскольку полный перебор всех комбинаций факторов для реальных систем обычно требует слишком больших затрат.

В данной работе рассматриваются методы создания тестовых наборов на основе комбинаций значений факторов, включающих все возможные сочетания пар, троек или больших множеств значений различных факторов. Такие методы успешно применялись для тестирования систем различных типов (см. работы [2,4–8]). Они позволяют создавать небольшие, в сравнении с полным перебором, но в то же время качественно тестирующие систему тестовые наборы при выполнении следующих условий.

- Есть некоторый вид ситуаций или воздействий на систему, имеющий довольно много параметров или факторов.

- Значения каждого из параметров можно разбить на (небольшое) конечное число классов, таких, что все существенные изменения в поведении системы происходят только из-за изменения класса одного из параметров.

- Известно, что ошибки в поведении системы возникают в основном за счет сочетания небольшого количества факторов, определяемых значениями используемых параметров.

При отсутствии дополнительной информации о зависимости между возможными ошибками и возможными значениями параметров эти методы позволяют достаточно эффективно строить тестовые наборы, исследующие большое разнообразие ситуаций, иначе можно использовать такую информацию для построения более компактных нацеленных тестов.

Рассмотрим в качестве примера тестирование пользовательского интерфейса системы перевода денежных средств WebMoney. Этот интерфейс доступен для любого современного браузера и разных операционных систем. Наиболее распространённой операцией в рамках этой системы является перевод денег с одного кошелька на другой с помощью интерфейса WM Keeper Light. Эта операция выполняется при таких часто используемых действиях пользователя в сети, как оплата мобильной связи или Интернета, покупка товара в интернет-магазине. На работу этой операции может оказать влияние несколько факторов. В качестве наиболее существенных из них выберем объём передаваемой суммы, необходимость конвертировать валюту, тип кошелька, с которого идёт платёж, способ аутентификации пользователя в системе WebMoney, браузер и операционную систему пользователя.

В качестве возможных значений этих параметров на основе документации по интерфейсу WM Keeper Light [9] выберем следующие (Таблица 1).

Если теперь попробовать составить все возможные комбинации значений факторов, получится $3 \times 2 \times 4 \times 4 \times 3 \times 5 = 1440$ тестов. Это немного, но, например, выполнение их всех вручную потребует значительных затрат.

Эмпирические исследования [7, 8, 10] утверждают, что в подобных случаях большинство ошибок (до 70%) связано с определенными комбинациями значений всего лишь двух параметров. В других работах [11] показано, что комбинации пар факторов при тестировании дают покрытие кода до 80% при более-менее разумном выборе значений отдельных

Таблица 1. Значения параметров для тестирования перевода денег с одного кошелька на другой с помощью интерфейса WM Keeper Light.

Передаваемая сумма	Нужна ли конвертация валюты	Тип кошелька, с которого идет платеж	Браузер	Способ аутентификации	Операционная система
< 100 руб.	Не нужна	WMR рубли	Internet Explorer	Сертификат X.509	Windows XP
$100 - 10000$ руб.	Нужна	WMZ доллары	Mozilla Firefox	Enum-авторизация	Windows Vista
> 10000 руб.		WME евро	Opera	Логин и пароль	Debian Ubuntu
		WMU гривны	Google Chrome		Linux SUSE
					Linux RedHat

параметров. Иными словами, если тесты будут содержать все возможные комбинации пар значений параметров, большая часть ошибок будет ими выявлена. При этом можно задействовать все пары значений различных факторов в небольшом наборе тестов. Минимальное количество таких тестов для данного примера равно 20, поскольку имеется 20 возможных сочетаний типов кошельков и операционных систем. Набор из такого количества тестов действительно можно построить, используя техники, описанные в данной работе (пример см. в таблице 2).

Можно попробовать составить тестовый набор так, чтобы он по-прежнему оставался небольшим, но содержал уже все различные тройки значений параметров. Для приведённого примера потребуется не менее $80 = 5 \times 4 \times 4$ тестов, что все же существенно меньше, чем 1440. Используя описанные ниже методы, можно получить и такой набор. При таком тестировании может быть обнаружено ещё больше ошибок [7, 10].

Данная работа посвящена обзору различных методов построения тестов, использующих все пары, тройки и большие множества значений параметров. Раздел 2 содержит определение покрывающих наборов и основные сведения о них. Раздел 3 посвящен собственно обзору, за ним следует заключение.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Тестовый набор, покрывающий все пары, тройки или большие множества возможных сочетаний значений параметров системы, соответствует математическому понятию покрывающего набора (covering array).

Пусть есть k факторов, оказывающих влияние на работу системы, причем первый фактор может принимать n_1 различных значений, второй n_2 и т.д. k-ый фактор – n_k значений. ***Покрывающим набором глубины*** **t** называется матрица из k столбцов, таких, что в i-ом столбце стоят значения i-ого фактора, и любая комбинация возможных значений любых t факторов встречается хотя бы в одной из ее строк.

Поскольку сами значения факторов не важны, можно обозначать их числами от 0 до $n_i - 1$. Набор чисел (t; k, $n_1 \ldots n_k$) принято называть ***конфигурацией покрывающего набора***, а множество покрывающих наборов с такой конфигурацией обозначать CA(t; k, $n_1 \ldots n_k$). Соответствующее формальное определение таково: целочисленная матрица $N \times k$, обозначаемая далее A, является покрывающим набором из CA(t; $n_1 \ldots n_k$) тогда и только тогда, когда

$$\forall i \in [1..N],\ j \in [1..k]\ A_{ij} \in [0..n_j - 1]\ \&\ \forall j_1, \ldots,$$
$$j_t \in [1..k]\ \forall v_1 \in [0..n_{j1} - 1], \ldots, v_t \in [0..n_{jt} - 1]$$
$$\exists i \in [1..N] \forall q \in [1..t] A_{ijq} = v_q.$$

В конфигурации (t; k, $n_1 \ldots n_k$) число t называется глубиной набора, k – количеством параметров или факторов, а n_i – количеством значений i-го параметра. Все числа t, k, $n_1, \ldots n_k$ назовем характеристиками конфигурации. Покрывающий набор глубины t иногда называют также t-покрывающим

Таблица 2. Минимальный набор тестов для интерфейса WM Keeper Light.

< 100 руб.	Не нужна	WMR рубли	Internet Explorer	Сертификат X.509	Windows XP
100-10000 руб.	Нужна	WMZ доллары	Mozilla Firefox	Enum-авторизация	Windows XP
> 10000 руб.	Не нужна	WME евро	Opera	Логин и пароль	Windows XP
100-10000 руб.	Нужна	WMU гривны	Google Chrome	Сертификат X.509	Windows XP
100-10000 руб.	Не нужна	WMR рубли	Mozilla Firefox	Логин и пароль	Windows Vista
> 10000 руб.	Не нужна	WMZ доллары	Opera	Сертификат X.509	Windows Vista
< 100 руб.	Нужна	WME евро	Google Chrome	Enum-авторизация	Windows Vista
> 10000 руб.	Не нужна	WMU гривны	Internet Explorer	Логин и пароль	Windows Vista
100-10000 руб.	Нужна	WMR рубли	Opera	Enum-авторизация	Debian Ubuntu
> 10000 руб.	Не нужна	WMZ доллары	Google Chrome	Логин и пароль	Debian Ubuntu
100-10000 руб.	Не нужна	WME евро	Internet Explorer	Enum-авторизация	Debian Ubuntu
< 100 руб.	Нужна	WMU гривны	Mozilla Firefox	Сертификат X.509	Debian Ubuntu
> 10000 руб.	Не нужна	WMR рубли	Google Chrome	Enum-авторизация	Linux SUSE
< 100 руб.	Нужна	WMZ доллары	Internet Explorer	Логин и пароль	Linux SUSE SUSE
> 10000 руб.	Нужна	WME евро	Mozilla Firefox	Сертификат X.509	Linux SUSE
< 100 руб.	Не нужна	WMU гривны	Opera	Enum-авторизация	Linux SUSE
100-10000 руб.	Нужна	WMR рубли	Google Chrome	Сертификат X.509	Linux RedHat
> 10000 руб.	Не нужна	WMZ доллары	Internet Explorer	Логин и пароль	Linux RedHat
> 10000 руб.	Нужна	WME евро	Mozilla Firefox	Сертификат X.509	Linux RedHat
< 100 руб.	Не нужна	WMU гривны	Opera	Enum-авторизация	Linux RedHat

набором (t-covering array). Количество строк в покрывающем наборе называется его размером.

Покрывающий набор ***минимален***, если не существует покрывающего набора для такой же конфигурации, содержащего меньшее количество строк. Размер минимального покрывающего набора конфигурации (t; k, n_1 ... n_k) обозначается CAN(t; k, n_1 ... n_k).

Если количества значений всех факторов совпадают, т.е. $n_1 = n_2 = \ldots = n_k = n$, соответствующий покрывающий набор называется

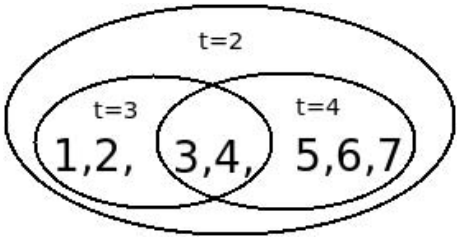

Рис. 1. Графическое представление конфигурации переменной глубины.

однородным. Его конфигурация обозначается (t; k, n), множество таких наборов – CA(t; k, n), минимальный размер такого набора – CAN(t; k, n).

Если количества значений нескольких факторов совпадают, т.е. $n_1 = n_2 = \ldots = n_{p1}$, $n_{p1+1} = n_{p1+2} = \ldots = n_{p2}$ и т.д, то для записи конфигурации покрывающего набора используется экспоненциальная нотация: CA(t; k, n_{p1}^{p1}, n_{p2}^{p2} ...), где $p_1 + p_2 + \cdots = k$.

Существуют более сложные варианты покрывающих наборов – ***покрывающие наборы переменной глубины*** (variable strength covering arrays). В таких наборах встречаются все комбинации из пар значений факторов $i_1, \ldots, i_{p2}$ ($p_2 \geq 2$, $p_2 \leq k$), все комбинации из троек значений факторов $j_1, \ldots, j_{p3}$ ($p_3 \geq 3$, $p_3 \leq k$), ... , все комбинации из t значений факторов $l_1, \ldots, l_{pt}$ ($p_t \geq t$, $p_t \leq k$, $t \leq k$), где $i_p, j_p, \ldots l_p$ – это номера факторов от 1 до k, причем номера, обозначаемые разными буквами, могут совпадать. Например, при наличии 7 факторов, нас могут интересовать комбинации всех пар значений 7 факторов, троек значений 1-го, 2-го, 3-го и 4-го факторов, и четверок значений 3-го, 4-го, 5-го, 6-го и 7-го факторов (см. рис. 1).

Формальное определение набора переменной глубины опустим из-за его громоздкости, а конфигурации таких наборов будем обозначать (t_1, n_1, ... , n_{p1}; t_2, j_1, ... , j_{p2}; ...; t_m, l_1,..., l_{pm}) или (t_1; k, n; t_2, j_1 ... j_{p2}; ... ; t_m, l_1, ... , l_{pm}) в однородном случае. Рис. 1 дает наглядное изображение части конфигурации (2; 7, 10; 3, 1, 2, 3, 4; 4, 3, 4, 5, 6, 7).

При описании тестов с помощью покрывающего набора, каждая строка покрывающего набора соответствует одному тесту, а присутствующее в ее j-ом столбце число соответствует номеру класса значений j-го параметра. Это соответствие позволяет использовать математическую теорию покрывающих наборов для построения тестовых наборов. Поскольку при построении тестов часто важно, чтобы их число было не слишком большим, удобно использовать минимальные или близкие к минимальным покрывающие наборы.

В общем случае для задачи построения минимального покрывающего набора не существует эффективных алгоритмов. В работе [12] с помощью сведения к задаче раскраски графа в три цвета доказано, что нахождение минимального набора из CA(t; k, n) – NP-полная задача. Lei и Tai [13] показали, что частная задача нахождения минимального набора из CA(2; k, n) тоже NP-полна, используя сведение к задаче покрытия вершин графа.

Для размера минимальных покрывающих наборов известна следующая оценка [14]: $CAN(t; k, n) \leq (t-1)\log(k)/\log(n^t/(n^t-1))(1+o(1))$ при $k \to \infty$, что дает $CAN(t; k, n) \leq (t-1)\, n^t \log(k)(1+o(1))$ при $n^t \to \infty$. При этом имеется очевидная нижняя граница $CAN(t; k, n) \geq n^t$, основанная на том, что все возможные комбинации из t значений, каждое из которых можно выбрать n способами, в таком наборе должны присутствовать. Как видно, с ростом количества задействованных параметров k,mlk размер растет лишь логарифмически, что и обуславливает практическую полезность покрывающих наборов для построения небольших, но качественных наборов тестов.

В работах [15–20] можно найти оценки размеров покрывающих наборов для частных случаев, а также обширные таблицы с явным указанием размеров наборов для определенных конфигураций. В данной работе мы опускаем эти оценки, чтобы не перегружать текст.

Исследования покрывающих наборов во многом сосредоточены на поиске минимальных наборов различных конфигураций и разработке эффективных (полиномиальных) алгоритмов построения покрывающих наборов близких к минимальным. Таких алгоритмов известно достаточно много, но они либо работают только для специфических конфигураций, либо дают для большинства

конфигураций наборы, намного превосходящие минимальные. Цель данной работы – проанализировать существующие методы построения покрывающих наборов и выявить их области возможного применения, где они дают минимальные или близкие к минимальным наборы.

3. ОБЗОР АЛГОРИТМОВ ПОСТРОЕНИЯ ПОКРЫВАЮЩИХ НАБОРОВ

В данном обзоре мы постарались охватить все алгоритмы построения покрывающих наборов, представленные в работах [15–19], а также ряд дополнительных:

- алгоритм комбинирования блоков из однородных покрывающих наборов и вспомогательных наборов [5];
- технику „двойной проекции“[21];
- обобщение алгоритма IPO до случая глубины $t > 2$ [22];
- эвристики для оптимизационных алгоритмов [8, 23–26];
- алгоритмы поиска с возвратом [27];
- алгоритм оптимизации заданного покрывающего набора [28].

Кроме того, мы приводим отсутствующие в перечисленных обзорах оценки объемов памяти, необходимой для работы этих алгоритмов.

Существующие алгоритмы построения покрывающих наборов можно классифицировать следующим образом.

- Комбинаторные (прямые) алгоритмы используют соответствие между покрывающими наборами и другими комбинаторными схемами (наборами латинских квадратов, орбитами действий групп и пр.), чтобы построить покрывающий набор в тех случаях, когда соответствующая ему схема устроена достаточно просто.
- Рекурсивные алгоритмы строят покрывающие наборы из покрывающих наборов для конфигураций с меньшими значениями характеристик (например, с меньшим количеством факторов или меньшей глубиной). При этом могут привлекаться и другие комбинаторные схемы (ортогональные наборы, разностные матрицы и пр.).
- Редуцирующие алгоритмы строят покрывающие наборы за счет модификации и сокращения покрывающих наборов для конфигураций с большими значениями характеристик.
- Оптимизационные алгоритмы рассматривают построение покрывающего набора как задачу минимизации числа строк в соответствующей матрице и используют те или иные техники поиска экстремумов такой функции (жадное построение матрицы, симуляция отжига, генетические алгоритмы и пр.).
- Алгоритмы поиска с возвратом последовательно перебирают всевозможные значения, стоящие в ячейках набора, для заданной конфигурации и заданного размера набора. В случае, если очередной полученный набор не является покрывающим, алгоритм откатывается на один или несколько шагов назад и повторяет попытку с другими значениями. Такие алгоритмы используют определённые правила для уменьшения вариантов перебора.

Далее при рассмотрении отдельных алгоритмов мы будем анализировать следующие их характеристики.

- Класс конфигураций наборов, которые можно получить с помощью рассматриваемого алгоритма. Особо отмечаются ситуации, в которых итоговый набор оказывается минимальным.
- Класс алгоритма. Большинство описываемых алгоритмов попадают только в один из перечисленных выше классов, однако некоторые можно использовать как для прямого построения нужного набора, так и для рекурсивной достройки меньшего.

- Временная сложность. Для рекурсивных и редуцирующих алгоритмов будет оцениваться только сложность их самих, вне зависимости от способа построения используемых ими исходных наборов.
- Объем требуемой памяти.
- Возможность построения некоторых частей набора независимо от других. Такая возможность полезна для более эффективного использования памяти, так как при этом можно не хранить весь получаемый набор. Приводятся оценки объема памяти, достаточного при использовании такого экономного режима.
- Возможность учета ограничений на используемые в наборе комбинации значений параметров. Она нужна, если не все возможные комбинации значений параметров тестируемого интерфейса имеют смысл или допустимы при обращениях к нему.

В данном обзоре не будут повторяться доказательства теорем о том, что построенные тем или иным методом наборы являются покрывающими и минимальными, поскольку эта информация может быть найдена в работах [15–20]. Кроме того, в данной работе мы не приводим алгоритмов построения покрывающих наборов для конечных классов конфигураций, которые также можно найти в перечисленных выше работах.

В тех случаях, когда алгоритм не использует случайного выбора каких-либо элементов или чисел, итоговый набор определяется однозначно. Однако любые перестановки столбцов и строк или перестановки значений любого параметра позволяют получить из одного покрывающего набора другие, эквивалентные данному.

Далее сначала рассматриваются алгоритмы построения однородных наборов, а затем – неоднородных или наборов переменной глубины.

3.1. Алгоритмы построения однородных покрывающих наборов

Наиболее хорошо развиты прямые и рекурсивные методы построения однородных покрывающих наборов, имеющих одинаковое количество значений всех параметров [16, 18].

3.1.1. „Булевский“алгоритм построения покрывающих наборов [19, 29, 30]

Данный алгоритм комбинаторного типа строит покрывающий набор из CA(2; k, 2) для любых значений $k \geq 1$.

Краткое описание алгоритма.

1. Выберем наименьшее N такое, что выполнено $k \leq C_{N-1}^{\lceil N/2 \rceil}$. Здесь $\lceil x \rceil$ – наименьшее целое число, большее или равное x, C_q^r – биномиальный коэффициент. Это число N равно размеру итогового набора.
2. Первую строку набора сделаем состоящей целиком из 0.
3. Оставшиеся $N-1$ строк строятся по столбцам, в качестве этих столбцов берутся все возможные последовательности из $\lceil N/2 \rceil$ единиц и $\lceil N/2 \rceil - 1$ нулей.

Легко убедиться, что получаемый так набор действительно покрывающий, доказательство того, что он минимальный см. в [19].

Алгоритм имеет временную оценку $O(k)$ и требует $O(k \cdot \log_2 k)$ памяти.

3.1.2 „Аффинный“алгоритм построения покрывающих наборов [5, 19]

Данный алгоритм комбинаторного типа строит покрывающий набор из CA(t; n+1, n) для n, равному степени простого числа, $n = p^k, k \geq 1$, и любых значений $t \leq n+1$. При $t = 3$ и $n = 2^k$ получается набор из CA(t; n+2, n).

Краткое описание алгоритма.

1. Для каждой степени простого числа $n = p^k$ существует конечное поле с таким количеством элементов, называемое полем Галуа $GF(p^k)$ [31]. Построим таблицу из элементов поля $GF(p^k)$ следующим образом.
2. Каждую строку таблицы кодируем при помощи последовательности $a_0\, a_1\, \ldots\, a_{t-1}$, где a_i принимает все возможные значения из поля $GF(p^k)$, всего получается n^t строк.

3. В первом столбце в строке с кодом $a_0\, a_1 \ldots a_{t-1}$, стоит значение равное a_0. Первому столбцу присваивается номер ∞.

4. Во втором столбце в строке с кодом $a_0\, a_1 \ldots a_{t-1}$, стоит значение равное a_{t-1}. Второму столбцу присвоим номер 0.

5. Все остальные столбцы, с третьего по (n+1)-й, с номерами $m = 1 \ldots (n-1)$ построим так, чтобы в строке с кодом $a_0\, a_1 \ldots a_{t-1}$, и столбце с номером m стояло значение вычисленное по формуле $\sum_{i=0}^{t-1} a_i m^i$ в арифметике $GF(p^k)$.

6. Для глубины $t = 3$ и $p = 2$ нужно добавить в эту таблицу еще один столбец, состоящий из значений a_1.

Построенная так таблица будет минимальным покрывающим набором из $CA(t;\ p^k + 1,\ p^k)$ или из $CA(3;\ 2^k + 2,\ 2^k)$, $CAN(t;\ p^k + 1,\ p^k) = p^{tk}$, $CAN(3; 2^k + 2,\ 2^k) = 2^{3k}$, доказательство см. в [19].

Временная сложность алгоритма оценивается как $O(n^t)$, а необходимая память – как $O(n)$, поскольку можно строить набор построчно. Однако для реализации необходимо смоделировать арифметику многочленов в поле Галуа, т.е. создать таблицы умножения и сложения. Создание таблиц требует $O(\log_p^4 n)$ операций, если известны неприводимые многочлены соответствующей степени, хранение таблиц займет $O(\log_p^3 n)$ памяти.

3.1.3 Алгоритм, основанный на действиях групп [17, 32–35]

Данный алгоритм комбинаторного типа строит покрывающие наборы для нескольких видов конфигураций, описанных ниже.

Краткое описание алгоритма.

1. Выбирается исходный набор, число столбцов в нем равно числу параметров в конфигурации. Пусть число строк в нем равно N_{start}.

2. Выбирается группа перестановок множества возможных значений параметров, которая будет действовать и на исходный набор. Пусть порядок группы равен n.

3. В результате действия каждого из n элементов группы на исходный набор получается n наборов, по размеру равных исходному. Один из этих наборов – это и есть исходный набор, результат действия единицы группы.

4. Полученные наборы присоединяются друг к другу снизу.

5. В ряде случаев в конце добавляется дополнительный набор, число столбцов в нем равно числу параметров в конфигурации. Пусть число строк в нем равно N_{end}.

6. В результате получаем итоговый набор числом строк равным $(nN_{start} + N_{end})$.

Выбор исходного набора, группы и дополнительного набора.

Для конфигураций (3; 2k, q+1) и частного случая – (3; 2k, k) [34], где $k > 2$, $q \geq k - 1$, q – степень простого числа. Размер итогового набора $(2k-1)(q^3 - q) + q + 1$. Алгоритм эффективен, если 2k – небольшое число, так как в этом случае размер набора стремится к $O(q^3)$, а минимальный возможный набор имеет размер $(q+1)^3$. Однако, если 2k близко к q или больше, тогда размер набора становится $O(q^4)$, что на порядок больше, чем минимальный.

1. Для полного графа с 2k вершинами существует 1-факторизация на 2k – 1 графов [36].

2. В каждой факторизации 2k вершин нумеруются от 0 до 2k – 1, а k ребер нумеруются от 0 до k – 1.

3. Вершины полного графа ставятся в соответствие столбцам исходного набора, а графы факторизации – строкам. Таким образом, в исходном наборе будет 2k столбцов и 2k – 1 строк.

4. Идем последовательно по графам факторизации и формируем строку исходного набора следующим образом: в столбец с номером вершины i проставляется номер ребра, инцидентного этой вершине в данном графе.

5. Исходный набор A можно также задать следующим образом:
$A_{i,j} = 0,\ 0 \le i \le 2k-2,\ j = 0;$
$A_{i,j} = |i-j+1|,\ |i-j+1| < k,\ j \ne 0;$
$A_{i,j} = 2k-1-|i-j+1|,\ k \le |i-j+1| < 2k-1,\ j \ne 0.$

6. Рассмотрим проективную группу PGL(q) = PGL(2, GF(q)) над конечным полем GF(q). Эта группа действует на элементах поля GF(q) перестановками. Ее порядок равен $q^3 - q$. Вычислить перестановки, соответствующие элементам этой группы, можно за $O(q^3)$ операций.

7. В качестве дополнительного набора выбираем набор C, где $C_{ij} = i$, $0 \le i \le 2k-1,\ 0 \le j \le q$.

Полученная так таблица представляет покрывающий набор, доказательство см. [34]. Однако обычно он не бывает минимальным.

Временная сложность алгоритма вместе с нахождением элементов группы PGL(q) – $O(2k((2k-1)(q^3-q)+q+1)+q^3) = O(k^2q^3)$, и необходимая память – $O(2k(2k-1)+q^4-q^2) = O(k^2+q^4)$, поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходный набор и элементы группы PGL(q).

Для конфигурации (2; 4, q), где $q \equiv 2 \pmod 4$. Размер построенного набора (q+1)q [35].

1. В качестве исходного набора выбирается транспонированный покрывающий набор разностей (difference covering array) – DCA(4; q+1; q), который определяется следующим образом: пусть (G,*) – группа порядка q, тогда DCA(k; n; q) – это матрица A с элементами $a_{ij}, 0 \le i \le k-1,\ 0 \le j \le n-1$ принадлежащими G, и такими, что для любых двух различных строк t и h, $0 \le t < h \le k-1$, каждый элемент G присутствует в векторе разности $\Delta_{th} = \{d_{hj} * d_{tj}^{-1},\ 0 \le j \le n-1\}$ хотя бы один раз. Способы нахождения DCA(4; q + 1; q), где $q \equiv 2 \pmod 4$ описаны в [35].

2. В качестве группы выбирается кольцо вычетов Z_q, если $q \equiv 6 \pmod{12}$ и $Z_2 + GF(q_1) + GF(q_2) + \ldots + GF(q_t)$, если $q \equiv 2$ или $10 \pmod{12}$, где q=2u, $u \ge 7$; $u = q_1q_2 \ldots q_t$ – разложение на степени простых чисел.

3. Дополнительного набора нет.

Полученная так таблица представляет покрывающий набор, доказательство см. [35], но не всегда минимальный.

Временная сложность алгоритма – $O(4(q+1)q) = O(q^2)$, необходимая память – $O(4(q+1)+q^2) = O(q^2)$, поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходный набор и элементы группы.

Для конфигураций (2; l, g) и (2; l+1, g) с размером, равным l(g – 1)+1 и (l+1)(g – 1)+1 соответственно, где конечное число пар (g, l) для каждого вида конфигурации можно найти в работах [17, 34]. Кроме пары (g, l) необходим вектор $(v_0, \ldots, v_{l-1})$, $v_i \in Z_{g-1} \cup \{\infty\}$, $v_0 = \infty$, который является исходным вектором покрытия (cover starter) для первого вида конфигурации и особым исходным вектором покрытия (distinct cover starter) для второго вида.

Пусть есть множество

$$D_s = \{(v_j - v_i)(\mathrm{mod}(g-1)) : j - i = s(\mathrm{mod}\, l),\ v_i \ne \infty,\ v_j \ne \infty\}.$$

Когда $D_s = Z_{g-1}$ для всех $1 \le s < l$, тогда вектор $(v_0, \ldots, v_{l-1})$ называется ***(g, l)-исходным вектором покрытия.*** Когда $Z_{g-1} \setminus \{0\} \subseteq D_s$ для всех $1 \le s < l$, и $\{v_1, \ldots, v_{l-1}\} = Z_{g-1}$, такой вектор $(v_0, \ldots, v_{l-1})$ называется ***особым (g, l)-исходным вектором покрытия*** [17]. Эти вектора находятся полным перебором [33] или эвристическим поиском [34].

1. Составляем матрицу из циклических сдвигов исходного вектора. Вектор циклически сдвигается l раз. В результате получается квадратная диагональная матрица размера l на l. Эту матрицу берем в качестве исходной для конфигурации вида (2; l, g). Для конфигурации вида (2; l+1, g) в качестве исходной берем ее же, пополненную столбцом из элементов 0.

2. В качестве группы выбирается кольцо вычетов Z_{g-1}, при этом группа не действует на символы ∞.

3. Дополнительный набор – это строка, состоящая из символов ∞.

4. Все символы ∞ заменяются на число g.

Построенная так таблица будет покрывающим набором, но не минимальным [17].

Временная сложность алгоритма, если известен исходный вектор покрытия – $O(l(l(g-1)+1)) = O(l^2 g)$, а необходимая память – $O(l+(g-1)g) = O(l+g^2)$, поскольку можно строить набор построчно, но необходимо хранить исходный вектор покрытия и элементы группы.

3.1.4 Алгоритм понижения глубины набора [34]

Этот редуцирующий алгоритм строит набор A, принадлежащий CA(t – 1; k – 1, n), из набора B, принадлежащего CA(t; k, n), размера N. Размер набора A равен N/n. Используя результаты предыдущего пункта можно построить набор из CA(2; 2k – 1, q+1) с помощью исходного CA(3; 2k, q+1), где $k > 2$, $q \geq k-1$, q – степень простого числа. Размер итогового набора равен

$$((2k-1)(q^3-q)+q+1)/(q+1) = (2k-1)q(q-1)+1.$$

Краткое описание алгоритма.

1. Выбрать j-ый столбец исходного набора B.

2. Выбрать одно из возможных значений параметров.

3. Оставить в исходном наборе только строки, содержащие выбранное значение в j-ом столбце.

4. Удалить j-ый столбец.

Полученная так таблица представляет покрывающий набор, доказательство см. [34]. Однако построенный так набор может не быть минимальным, даже если исходный набор минимален.

Временная сложность алгоритма O(Nk), и таков же объем необходимой памяти, поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходный набор.

3.1.5 Мультипликативный алгоритм [17, 19]

Данный алгоритм рекурсивного типа строит покрывающий набор из $CA(t; k, n_1 \cdot n_2)$, используя покрывающий набор из $CA(t; k, n_1)$ и покрывающий набор из $CA(t; k, n_2)$. Размер итогового набора будет равен произведению размеров исходных наборов. Для большого числа параметров k этот алгоритм строит наборы, далекие от минимальных.

Краткое описание алгоритма.

1. Пусть есть два покрывающих набора A из $CA(t; k, n_1)$ и B из $CA(t; k, n_2)$.

2. Обозначим элементы двух этих наборов через a_{ij} и b_{lj}, – у этих наборов одинаковое число столбцов, и, возможно, разное число строк.

3. Любое число от 0 до $(n_1 \cdot n_2 - 1)$ можно однозначно представить в виде $n_2 \cdot i + m$, где i лежит от 0 до $(n_1 - 1)$, m – от 0 до $(n_2 - 1)$. Таким образом, можно однозначно установить соответствие между индексами строк таблицы получаемого набора и парами индексов строк двух исходных таблиц – (i, m), где i – индекс строк первого набора, m – индекс строк второго набора.

4. Элементы этой таблицы могут быть построены по формуле $x_{(i,m)j} = n_2 \cdot a_{ij} + b_{mj}$.

Полученная так таблица представляет покрывающий набор с k параметрами, $n_1 \cdot n_2$ значениями для глубины t, доказательство см. [19]. Однако, построенный так набор может быть не минимальным, даже если минимальны исходные наборы. Контрпример (см. [37]): существует покрывающий набор для конфигурации (2; 6, 10) с числом строк равным 102, тогда как описанный метод дает покрывающий набор для конфигурации (2; 6, 10) с числом строк равным 150, используя в качестве исходных минимальные наборы из

CA(2; 6, 2)(CAN(2; 6, 2) = 6) и из CA(2; 6, 5) (CAN(2; 6, 5) = 25).

Временная сложность алгоритма $O(k(1 + N_1 + N_2))$, где N_1 – число строк в первом исходном наборе, N_2 – число строк во втором, и такова же необходимая память, поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходные наборы.

3.1.6 Построение однородных покрывающих наборов глубины 2 с использованием рекурсивных конструкций [17, 19, 33]

Введем несколько вспомогательных определений [33].

Разделённый покрывающий набор (partitioned covering array), PCA(N; 2; (k_1, k_2), n) – это покрывающий набор из $CA(2; k_1 + k_2, n)$ размера N, допускающий следующее разбиение (с учетом того, что строки и столбцы можно менять местами):

Таблица 3. Разделённый покрывающий набор PCA.

A1	A2
P	X

Здесь A_1 – матрица размером $N - n$ на k_1, A_2 – матрица размером $N - n$ на k_2, P – матрица размером n на k_1, в каждом столбце которой встречаются все числа от 0 до $n - 1$, X – матрица размером k_2 на n. Не нарушая общности, можно считать P матрицей D в которой i-ая строка содержит значение i, $0 \le i \le n - 1$. Пример PCA(15; 2; (14, 6), 3):

Любой покрывающий набор из CA(2; k, n) размера N может быть представлен в виде PCA(N; 2; (1, k – 1), n).

SCA(N; 2; (k_1, k_2), n) – это покрывающий набор из $CA(2; k_1 + k_2, n)$ размера N, в котором для всех i: $0 \le i \le n - 1$, в строке с номером $N - n + i$ первые k_1 значений равны i, а последние k_2 значений равны 0. Например, набор из CA(2; k, n) может быть представлен в виде $SCA(n_2; 2; (n, 1), n)$, если n – степень простого числа. SCA является частным случаем PCA, когда все элементы X равны 0.

Данный алгоритм рекурсивного типа строит покрывающие наборы для нескольких видов конфигураций, используя рекурсивные схемы. Конфигурации, для которых работает алгоритм (везде, если не оговорено дополнительно, $r \ge 1$, n – степень простого числа):

$$\text{SCA (или PCA)}(N + r(n^2 - n);\ 2;\ (k_1 n^r,\ r k_1 n^{r-1} + k_2 n^r), n)$$

(то есть, набор из $CA(2;\ k_1 n^r + r k_1 n^{r-1} + k_2 n^r, n)$, с числом строк $N + r(n^2 - n)$), используя исходный SCA (соответственно PCA) (N; 2; (k_1, k_2), n).

Также, используя тот же исходный SCA (или PCA) и проделав дополнительные действия, можно получить набор для большего числа параметров – SCA (или $PCA)(N + r(n^2 - n);\ 2;\ (n(k_1 + k_2)D_{r,n} + k_1 D_{r-1},\ n(k_1 + k_2)D_{r-1} + k_1 D_{r-2,n}),\ n)$, где $r \ge 2$, $D_{r,t} = \sum_{i=1}^{\lfloor (r+1)/2 \rfloor} C_{i-1}^{r-i} t^{r-i}$, C_i^r – число распределений из r по i. Это будет набор из $CA(2;\ n(k_1 + k_2)D_{r+1,n} + k_1 D_{r,n},\ n)$, с числом строк $N + r(n^2 - n)$. Числа N, k_1, k_2, n принимают следующие значения:

(i) $k_1 = 1$, $k_2 = m - 1$, N – число строк в исходном покрывающем наборе из CA(2; m, n), полученном другими методами и представленном в виде PCA(N; 2; (1, m –1), n).

(ii) $k_1 = n$, $k_2 = 1$, $N = n^2$, исходный $SCA(n^2; 2; (n, 1), n)$ – это набор из CA(2; n+1, n) с числом строк равным n^2, полученный с помощью алгоритма описанного в пункте 3.1.2.

(iii) $k_1 = 1$, $k_2 = 1$, $N = (l + 1)(n - 1) + 1$, для чисел n и l существует особый (n, l) – исходный вектор покрытия, см. пункт 3.1.3. Исходный SCA – это набор из CA(2; l+1, n), число строк в котором равно (l+1)(n – 1)+1. Этот набор получен с помощью алгоритма, описанного в пункте 3.1.3 и, согласно [33], набор представим в виде $SCA((l + 1)(n - 1) + 1;\ 2;\ (l,\ 1),\ n)$.

(iv)n=3, ($k_1 = 14$, $k_2 = 1$, $N = 15$), ($k_1 = 60$, $k_2 = 14$, $N = 21$), ($k_1 = 220$, $k_2 = 114$, $N = 27$), ($k_1 = 1092$, $k_2 = 220$, $N = 33$). Исходные SCA получены эвристическими методами, их можно найти в [33]. Для метода подходят любые исходные SCA(N; 2; (k_1, k_2), n) близкие к минимальным, где n – степень простого числа.

Краткое описание алгоритма.

Таблица 4. PCA(15; 2; (14, 6), 3) [35].

0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1
0	0	1	1	0	1	2	2	2	2	2	2	2	2	0	0	0	2	2	2
0	2	2	2	0	1	0	0	0	1	2	0	1	1	2	2	2	0	2	1
1	1	1	0	1	0	1	1	0	0	2	2	2	1	2	2	2	2	1	0
1	2	0	2	1	0	0	2	2	1	0	1	0	2	0	1	2	2	2	1
1	2	2	1	2	0	2	1	1	0	1	0	2	0	1	0	2	1	2	1
2	0	0	1	2	2	2	2	0	2	2	1	0	1	2	2	2	1	0	2
2	1	2	0	2	2	1	2	2	0	1	0	1	2	0	2	1	2	2	0
2	1	1	2	0	2	0	1	1	2	1	2	0	2	2	0	1	1	0	1
2	1	1	2	2	1	2	0	2	1	0	1	2	0	2	1	0	1	0	0
1	2	2	1	1	2	1	0	1	2	0	2	1	0	1	2	0	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	1	0	1	0	2	2	1	2	2	0	1	0	1	1	1	0	1	2
1	0	2	2	0	1	1	1	2	0	0	1	0	1	1	1	1	0	1	2
0	1	0	1	2	2	0	0	0	1	1	2	2	2	1	1	1	0	1	2

1. Пусть есть ***исходный*** SCA (или PCA) (N; 2; (k_1, k_2), n) с разбиением A_1, A_2, D и X, см. таблицу 3.

2. Пусть есть ***вспомогательный*** SCA(M; 2; (l_1, l_2), n) с разбиением B_1, B_2, D', и O'.

3. Назовем соединением матриц A размером N на k и B размером M на l ($A \otimes B$) матрицу C размером N+M на kl, в которой элементы заданы следующим образом: $C_{i,(f-1)k+g} = A_{i,g}$, $1 \le i \le N$, $1 \le f \le l$, $1 \le g \le k$, $C_{N+i,(f-1)k+g} = B_{i,f}$, $1 \le i \le M$, $1 \le f \le l$, $1 \le g \le k$.

4. Итоговый SCA (или PCA) ($N + M - n$; 2; (k_1l_1, $k_1l_2 + k_2l_1$), n) получается из исходного SCA (или PCA) и вспомогательного SCA путем присоединения друг к другу горизонтально матриц $A_1 \otimes B_1$, $A_2 \otimes B_1$, $A_1 \otimes B_2$. Затем присоединяем снизу матрицу, составленную из горизонтального соединения матриц D, k_1X и O' соответствующих размеров, где k_1X – это k_1 раз повторенная матрица X (см. таблицу 5).

5. Дополнительные действия: если B_1 содержит подматрицу $n \times \eta$ ($\eta \times n$), в которой в каждой строке (соответственно, столбце) все числа от 0 до $n - 1$ различны, то в итоговой матрице строчки и столбцы можно переставить так, чтобы получился SCA (или PCA)($N + M - n$; 2; ($\eta(k_1 + k_2)$, $(l_1 - \eta)k_1 + (+k_2) + k_1l_2$), n). Потребуется $n\eta(k_1 + k_2)$ перестановок.

Таблица 5. Построение итогового SCA (или PCA).

$A_1 \otimes B_1$	$A_2 \otimes B_1$	$A_1 \otimes B_2$
D	k_1X	O'

Полученная так таблица представляет покрывающий набор, доказательство см. [33]. Построенный так набор может быть не минимальным, даже если минимальны исходные наборы, см [33].

Временная сложность алгоритма без дополнительных действий

$$O((k_1l_1 + k_1l_2 + k_2l_1)(N + M - n)),$$

а с дополнительными действиями

$$O((k_1l_1 + k_1l_2 + k_2l_1)(N + M - n) + n\eta(k_1 + k_2)).$$

Необходимая память без дополнительных действий $O((k_1 + k_2)N + (l_2 + l_1)M)$, поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходный и вспомогательный наборы.

Для получения наборов для конфигураций, описанных выше, в качестве исходного SCA (или

PCA) выбирается указанный в соответствующем пункте (i) – (iv), а в качестве вспомогательного – набор из CA(2; n+1, n), где n – степень простого числа, с числом строк n_2, который является SCA(n^2; 2; (n, 1), n). Такой набор можно получить с помощью алгоритма, описанного в пункте 3.1.2. Построение повторяется r раз, в качестве исходного берется набор, полученный на предыдущем шаге, а в качестве вспомогательного – тот же вспомогательный набор SCA(n^2; 2; (n, 1), n). Временная сложность для проведения r итераций без дополнительных действий:

$$O(\sum_{i=1}^{r}((N + i(n^2 - n))(k_1 n^i + ik_1 n^{i-1} + k_2 n^i))),$$

а с дополнительными действиями:

$$O(\sum_{i=1}^{r}((N + i(n^2 - n))(n(k_1 + k_2)D_{i+1,n} + k_1 D_{i,n})) + \sum_{i=1}^{r-1}(n^2(n(k_1 + k_2)D_{i,n} + k_1 D_{i-1,n}))).$$

Необходимая память будет складываться из размеров самого большого исходного набора и вспомогательного набора:

$$O((N + (r - 1)(n^2 - n))(k_1 n^{r-1} + (r - 1)k_1 n^{r-2} + k_2 n^{r-1}) + (n + 1)n^2)$$

без дополнительных действий и

$$O((N + (r - 1)(n^2 - n))(n(k_1 + k_2)D_{r,n} + k_1 D_{r-1,n}) + (n + 1)n^2)$$

с дополнительными действиями.

В работе [33] описан метод, основанный на бинарных кодах, с помощью которого можно строить SCA($n^2 + r(n^2 - n)$; 2; (n^{r+1}, $(r + 1 + s)n^{r+1-s}$), n), то есть наборы из CA($2; n^{r+1} + C_s^{r+1} n^{r+1-s}$, n) с числом строк равным $n^2 + r(n^2 - n)$, где $r \geq s \geq 1$, n – степень простого числа. Однако нетрудно убедиться, что в подавляющем большинстве случаев с помощью уже описанных методов можно получить SCA того же размера для конфигураций, в которых число k_1 и сумма $k_1 + k_2$ будут больше или равны, чем у SCA, построенных этим методом.

3.1.7 Построение однородных покрывающих наборов глубины 3 с использованием уравновешенных схем [17]

Уравновешенная схема OD_λ(t, k, v) (ordered design) – это матрица размером $\lambda C_t^v t! \times k$ с элементами от 0 до v, в которой:

1. В каждом столбце содержится v различных значений;
2. Каждые t столбцов содержат ровно λ одинаковых строк, в которых значения в разных столбцах различны.

Если $\lambda = 1$, то соответствующие схемы обозначаются просто OD(t, k, v). Если q – степень простого числа, то существует OD(3, q+1, q+1) [17].

Данный алгоритм рекурсивного типа строит покрывающий набор из CA(3; q+1, q+1) для q, равному степени простого числа, используя OD(3, q+1, q+1), с числом строк равным $q^3 - q$, покрывающий набор A из CA(3; q+1, 2), с числом строк равным N. Число строк в итоговом наборе будет равно $q^3 - q + Nq(q + 1)/2 - (q^2 - 1)$, что меньше, чем у алгоритма, рассмотренного в пункте 3.1.3, который, однако, имеет более широкую область применения.

Краткое описание алгоритма.

1. Из набора A получаем еще $C_2^{q+1} - 1 = q(q + 1)/2 - 1$ вариантов этого набора путем замены значений элементов с 0 и 1 на значения от 0 до q, перебрав все возможные сочетания пар различных значений, кроме уже имеющихся 0 и 1.
2. Полученные в предыдущем пункте наборы (а также набор A) присоединяем друг к другу снизу, не включая строки вида (x, x, ..., x), где x от 0 до q. Получается набор с числом строк равным Nq(q+1)/2 - q(q+1).
3. Полученный в предыдущем пункте набор присоединяем снизу к матрице OD(3, q+1, q+1).
4. Добавляем q+1 строку вида (x, x, ..., x), где x от 0 до q.

Построенная так таблица будет (не обязательно минимальным) покрывающим набором из CA(3, q+1, q+1) для q, равного степени простого числа, доказательство см. в [17].

Временная сложность алгоритма оценивается как

$$O((q^3 - q + Nq(q + 1)/2 - (q^2 - 1))(q + 1)) = O(q^4 + Nq^3),$$

а необходимая память – как

$$O((q^3 - q + N)(q + 1)) = O(q^4 + Nq),$$

поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходный набор и матрицу OD(3, q+1, q+1).

3.1.8 Построение однородных покрывающих наборов с использованием совершенных хэш-семейств [17, 32]

Введем несколько вспомогательных определений.

Совершенное хэш-семейство (perfect hash family) PHF(N, t; k, m) – это набор из N (k, m)-хэш-функций (задающих отображения из множества K мощности k во множество M мощности m, обозначим их h_j), такой, что для каждого подмножества X мощности t множества K существует хэш-функция h_j инъективная (совершенная) на X. Это семейство можно задать с помощью матрицы размером k на N, в которой элемент [i, j] – это значение h_j-ой функции на i-ом элементе множества K. Любая матрица размера k на N, в которой в любой её подматрице размера $t \times N$ присутствует строка длины t, в которой все значения различны, задает некоторое совершенное хэш-семейство.

Разностная матрица (difference matrix) D(k, n; λ) – это матрица размера $n \times k\lambda$, с элементами из Z_k, в которой вектор разности любой пары строк содержит каждый элемент поля Z_k, точно λ раз. Например, если НОД((n − 1)!, k) = 1, то матрица $D_{ij} = ij \bmod k$ – это разностная матрица D(k, n; 1).

Данный алгоритм рекурсивного типа строит покрывающий набор из CA(t; k^{2p}, n) с помощью исходного набора из CA(t; k, n) размера N, если НОД((t − 1)t/2, k) = 1. Размер итогового набора будет равен $N((t-1)t/2+1)^p$. Этот же алгоритм позволяет строить PHF($N((t-1)t/2+1)^p$, t; k^{2p}, n) из PHF(N, t; k, n).

Краткое описание алгоритма.

1. Пусть матрица A – это исходная матрица, представляющая из себя PHF(N, t; k, n) (или исходный транспонированный набор из CA(t; k, n) размера N).
2. Пусть D – разностная матрица $D(k,\ (t-1)t/2+1;\ 1)$, $D_{ij} = ij \bmod k$, $0 \le i \le k-1,\ 0 \le j \le (t-1)t/2$.
3. Пусть A^x – это матрица того же размера, что и A, для которой $A^x_{ij} = A_{(i+x)j}$, $0 \le x,\ i \le k-1,\ 0 \le j \le N-1$.
4. Итоговая матрица собирается из матриц $B_{ij} = A^{Dij}$, $0 \le i \le k-1,\ 0 \le j \le (t-1)t/2$, как показано в таблице 6. Это будет PHF($N((t-1)t/2+1)$, t; k^2, m), если исходная матрица была PHF или транспонированный набор из CA(t; k^2, n), если исходная матрица была покрывающим набором, доказательство см. в [17]. Построенный набор не обязательно минимален.
5. Повторяя алгоритм p раз можно получить PHF или покрывающий набор для указанного выше вида конфигураций.

Таблица 6. Построение итоговой матрицы с использованием матрицы разностей.

$B_{0,\,0}$	$B_{0,\,1}$	...	$B_{0,\,(t-1)\,t/2}$
$B_{1,\,0}$	$B_{1,\,1}$	...	$B_{1,\,(t-1)\,t/2}$
...	...	...	...
$B_{k-1,\,0}$	$B_{k-1,\,1}$	...	$B_{k-1,\,(t-1)\,t/2}$

Временная сложность алгоритма –

$$O(k^p((t-1)t/2+1) + Nk^{2p}((t-1)t/2+1)^p) = O(Nk^{2p}t^{2p}),$$

а необходимая память –

$$O(k^p((t-1)t/2+1+N)) = O(k^p(t^2+N)),$$

поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходный набор и матрицу $D(k^p,\ (t-1)t/2+1;\ 1)$.

Существует алгоритм [32], позволяющий строить набор из CA(t; k, n) с помощью PHF(N_1, t; k, m) и исходного набора из CA(t; m, n) размера N_2. Размер итогового набора будет равен N_1N_2.

Краткое описание алгоритма.

1. Перебираем все строки исходного покрывающего набора (обозначим эту матрицу CA) от 0 до $N_2 - 1$. Пусть выбранная строка имеет номер j.

2. Перебираем все столбцы исходной PHF от 0 до $N_1 - 1$. Пусть выбран столбец l.

3. Перебираем все значения i от 0 до $k - 1$ и формируем строку длины k с двойным индексом (j, l) следующим образом: i-ым элементом строки становится элемент CA_{xj}, где x – это элемент PHF_{il}.

4. В итоговом наборе N_1N_2 строк и k столбцов.

Построенная так таблица будет покрывающим набором, доказательство см. в [17]. Однако построенный набор не будет минимальным.

Временная сложность алгоритма – $O(N_1N_2k)$, а необходимая память – $O(N_1k + N_2m)$, поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходный набор и матрицу PHF(N_1, t; k, m).

Используя первую рекурсивную схему можно получать наборы из CA(3; $(2v-1)^{2\wedge j}$, v) с числом строк $((2v-1)(q^3-q)+v)4^j$, где v>2, $v \equiv 0,\ 1 \bmod 3$, $q \geq v-1$ – степень простого числа и j – любое целое. А также набор из CA(3; $(2v-3)^{2\wedge j}$, v) с числом строк $((2v-1)(q^3-q)+v)4^j$, где v>2, $v \equiv 2 \bmod 3$ и такие же ограничения на q. Для этого в качестве исходного нужно взять набор из CA(3; 2v, v) с числом строк $(2v-1)(q^3-q)+v$, полученный с помощью метода, описанного в пункте 3.1.3. Если $v \equiv 0,\ 1 \bmod 3$, то для того, чтобы НОД((t–1)t/2, k) = НОД(3, k) был равен 1 (как требуется в схеме), то k=2v–1, а если $v \equiv 2 \bmod 3$, v>2, то k=2v–3. Также, используя эту схему, можно получать необходимые PHF для второй описанной рекурсивной схемы.

3.1.9 Построение однородных покрывающих наборов глубины более 2 с использованием рекурсивных конструкций (теорема Roux) [15–17, 32, 38–40]

Данный алгоритм рекурсивного типа строит покрывающий набор из CA(3; 2k, n), используя покрывающий набор из CA(3; k, n) и покрывающий набор из CA(2; k, n).

Если число строк в первом исходом наборе равно N_3, а во втором равно N_2, то в итоговом оно будет равно $N_3 + (n-1)N_2$. Для мощности t больше 3 этот алгоритм строит покрывающий набор из CA(t; 2k, n), используя t–1 исходный покрывающий набор соответственно из CA(t; k, n), ... CA(2; k, n).

Краткое описание алгоритма для $t = 3$.

1. Пусть есть набор CA(3; k, n), с числом строк равным N_3. Обозначим его элементы как $A^3_{i,j}$ $(1 \leq i \leq N_3,\ 1 \leq j \leq k)$.

2. Пусть есть набор CA(2; k, n), с числом строк равным N_2. Обозначим его элементы как $A^2_{i,j}$ $(1 \leq i \leq N_2,\ 1 \leq j \leq k)$.

3. Первые N_3 строк итогового набора (его элементы обозначим как $C_{i,j}$) из 2k столбцов строим следующим образом:

3.1 $C_{i,j} = A^3_{i,j}$, где $1 \leq i \leq N_3,\ 1 \leq j \leq k$,

3.2 $C_{i,j} = A^3_{i,j-k}$, где $1 \leq i \leq N_3,\ k+1 \leq j \leq 2k$.

4. Строки с $N_3 + 1$ до $N_3 + (n-1)N_2$ строим следующим образом:

4.1 $C_{i,j} = A^2_{(i-N_3-1) \bmod N_2+1,\ j,}$ где $N_3 + 1 \leq i \leq N_3 + (n-1)N_2,\ 1 \leq j \leq k$.

4.2 $C_{i,j} = A^2_{(i-N_3-1) \bmod N_2+1,\ j-k,} +_n((i-N_3-1) \operatorname{div} N_2 + 1),)$ где $N_3 + 1 \leq i \leq N_3 + (n-1)N_2,\ k+1 \leq j \leq 2k$, $+_n$ – сложение по модулю n.

Полученный набор будет покрывающим, доказательство см. [17]. Но он может не быть минимальным, даже если минимальны исходные наборы (см. [37]): существует набор для конфигурации (3; 18, 13) размера 3912, тогда как описанный метод дает покрывающий набор для конфигурации (3; 18, 13) размера 4225, используя в качестве исходных минимальные наборы из CA(3; 9, 13) (CAN(3; 9, 13)=2197) и из CA(2; 9, 13) (CAN(2; 9, 13)=169).

Краткое описание алгоритма для $t \geq 4$.

1. Пусть есть наборы A^t из CA(t; k, n), с числом строк равным N_t, ... , A^2 из CA(2; k, n), с числом строк равным N_2.

2. Обозначим через $B^{N_j}_{i}$ матрицу размером N_iN_j на k элементов, полученную из A^i повторением каждой строки N_j раз, где $2 \leq i,\ j \leq t-2,\ i + j = t$.

3. Обозначим через $C^{N}_{j\,i}$ матрицу размером N_iN_j на k элементов, полученную повторением матрицы A_jN_i раз, где $2 \le i,\ j \le t-2$, и $i+j=t$.

4. Обозначим через E_i матрицу размером N_iN_j на 2k элементов, где первые k столбцов – это матрица B_i^{Nt-i}, а следующие k столбцов – это матрица C^{Ni}_{t-i}, $2 \le i,\ j \le t-2$.

5. Первые $N_t + (n-1)N_{t-1}$ строк итогового набора строим аналогично пунктам 3 и 4 алгоритма для t=3, в качестве исходных возьмем матрицы A^t и A^{t-1}.

6. Следующие строки получаем присоединением снизу к полученному на шаге 5 набору матриц $E_{t-2}\ \ldots\ E_2$.

Полученный набор будет покрывающим, доказательство см. [15], но он может не быть минимальным, даже если минимальны исходные наборы, контрпример строится аналогично случаю t=3.

Временная сложность алгоритма – $O(2kN_{max}(n + N_{max}(t-3)))$, необходимая память – $O(2kN^2_{max})$ (где N_{max} – размер наибольшего исходного набора (N_t)), поскольку можно строить набор построчно, но при этом все время необходимо держать в памяти исходные наборы.

Для всех алгоритмов, приведённых в данном разделе, нет возможности задать ограничения на возможные комбинации значений параметров.

3.2. Методы построения неоднородных покрывающих наборов и наборов с переменной глубиной

3.2.1 Построение неоднородных наборов с постоянной глубиной при помощи близких по конфигурации однородных покрывающих наборов [16]

Данный редуцирующий алгоритм строит неоднородный покрывающий набор при помощи близкого по конфигурации однородного покрывающего набора. Например, набор из CA(2; 6, 3, 2, 5, 4, 3, 5) при помощи исходного набора из CA(2; 6, 5, 5, 5, 5, 5, 5) (то есть, CA(2; 6, 5)). Необходимо дальнейшее изучение классов частных случаев, в которых этот приём работает хорошо. В работе [16] приводятся некоторые соображения по этому поводу.

Краткое описание алгоритма.

1. Пусть необходимо построить неоднородный покрывающий набор A из CA(t; k, $n_1\ \ldots\ n_k$).

2. Пусть есть однородный покрывающий набор B из CA(t; k, n), построенный другими методами, где n выбрано так, чтобы набор B заведомо содержал в себе все строки набора A (самый тривиальный выбор: $n = \max(n_1\ \ldots\ n_k)$).

3. Заменяем те значения из B, которые не входят в область допустимых значений A, значениями, входящими в область допустимых значений A.

4. Выбрасываем из получившегося в пункте 3 набора все лишние строки так, чтобы он сохранял свойства покрывающего набора. В результате получаем неоднородный покрывающий набор из CA(t; k, $n_1\ \ldots\ n_k$).

Результирующий покрывающий набор зависит от метода “выбрасывания” лишних значений, то есть, вполне вероятно, что построенные для одной и той же конфигурации покрывающие наборы будут отличаться друг от друга. Гарантировать минимальность построенного таким методом набора нельзя, однако можно оценить, насколько он близок к минимальному [16].

Алгоритм требует детального анализа конфигурации покрывающего набора, а также нетривиальных эвристик. Временная сложность и затраты памяти зависят от частных случаев. Построение набора нельзя оптимизировать за счет указания ограничений на возможные комбинации значений, хотя такие ограничения можно ввести. Также нельзя оптимизировать расход памяти, поскольку все равно придется держать в памяти весь исходный набор.

3.2.2 Алгоритм “двойной проекции„[21]

Независимая пара (private pair).

Отметим символом * в покрывающем наборе элементы, которые не влияют на покрытие

всех кортежей размера t (то есть те места, при подстановке в которые любых значений матрица сохраняет свойства покрывающего набора).

Пусть C – покрывающий набор из CA(2; k, n_1, ... , n_k) с числом строк N. Пусть i и j – номера столбцов со значениями σ_i и σ_j соответственно. Тогда упорядоченная пара $\{(i, \sigma_i), (j, \sigma_j)\}$ будет независимой парой, если либо σ_i и σ_j оба равны символу *, либо пара (σ_i, σ_j) встречается в подматрице размером $N \times 2$, составленной из столбцов i и j, единственный раз.

Пусть $I = \{i_1, \ldots, i_s\}$ – множество столбцов. Строка из CA будет **независимой строкой над множеством I**, если каждая пара ее значений в столбцах $\{i, j\} \subseteq I$, где $i \neq j$, вместе с номерами этих столбцов дает независимую пару.

Данный редуцирующий алгоритм строит следующие неоднородные покрывающие наборы.

- Набор из CA(2; $k + 1$, $n_1 - 1$, ... , $n_k - 1$, s) с числом строк $N - 1$ с помощью набора из CA(2; k, n_1, ... , n_k) с числом строк N, где в исходном наборе существует независимая строка p над множеством I мощности s, а также значение x_i в строке p и столбце $i \in I$ не является символом * и встречается по крайней мере в n_i строках исходного набора.

- Набор из CA(2; $q + 1 + t$, $(q - t)^{q+1}$, s^t) размера $q^2 - t$ при помощи однородного набора из CA(2; q + 1, q), размера q^2, где q – степень простого числа, $1 \leq t \leq q$ и $1 \leq s \leq q - t$.

Краткое описание алгоритма.

1. Пусть есть набор C из CA(2; k, n_1, ... , n_k) размера N, удовлетворяющий условиям п.1 выше, $I = \{l_1, \ldots, l_s\}$ – множество столбцов, $(x_1, \ldots, x_k)$ – независимая строка p над I.

2. Разбиваем все строки набора C, кроме p, на s+1 класс C_1, ... , C_s и D. Класс C_1 содержит все строки (кроме p), в которых значение x_i стоит в столбце i. D содержит оставшиеся строки. Среди классов C_1, ... , C_s нет равных друг другу, поскольку p – независимая строка над I. Для каждого класса C_i выполняем шаги 3–5:

3. Для всех строк из C_i проставляем значение i–1 в новый столбец k+1.

4. Из условия, что x_i в строке p и столбце $i \in I$ не является символом * и встречается по крайней мере в n_i строках исходного набора, следует, что в C_i есть по крайней мере $n_i - 1$ строка. Выбираем $n_i - 1$ строку из C_i и заменяем в l_i-ом столбце значение x_{l_i} на любые допустимые значения так, чтобы в выбранных строках в l_i-ом столбце все значения были различны.

5. Для всех остальных строк из C_i заменяем значения в столбцах из I на символы *.

6. Для всех столбцов не из I (с индексом j) заменяем все x_{pj}, на символы *.

7. Удаляем строку p.

8. В столбец k+1 для строк из D проставляем *.

9. Переименовываем значения в каждом столбце, кроме k+1, чтобы там были значения от 0 до $n_i - 1$. Получили набор для первого вида конфигураций.

10. Для второго вида конфигураций в качестве исходного набора выбираем набор из CA(2; q + 1, q), с числом строк q^2 (построенного с помощью метода, описанного в пункте 3.1.2), где q – степень простого числа. В таком наборе любая строка – независимая над любым множеством столбцов.

11. Выбираем t строк, которые будем удалять – p_1, ... , p_t, таких, что первые их q элементы с индексом j от 0 до q будут равны j, а последние равны 0. Таких строк не более q.

12. В качестве I выбираем первые s столбцов. Не имеет смысла брать их более q – t, так как число возможных значений в добавляемом столбце не может быть более q – t.

13. Выполняем шаги 1–9 для выбранных p_1, ... , p_t строк с выбранным исходным

набором и множеством I. Это можно сделать за один проход по набору.

Полученные наборы будут покрывающими, доказательство см. [21], но могут не быть минимальными. Временная сложность алгоритма O(N(k+1)) и необходимая память O(Nk), поскольку все время необходимо держать в памяти исходные наборы.

3.2.3 Построение неоднородных наборов с глубиной t=2 при помощи комбинирования блоков из однородных покрывающих наборов и вспомогательных наборов [5]

Данный алгоритм рекурсивного типа строит покрывающий набор из CA(2; k, $n_1 \ldots n_k$), где $k > \max(n_1 \ldots n_k)$. Необходимы дальнейшие исследования для обобщения этого метода на случай CA(t; k, $n_1 \ldots n_k$), t>2, а также для построения наборов с переменной глубиной.

Краткое описание алгоритма.

1. Пусть необходимо построить неоднородный покрывающий набор A из CA(2; k, $n_1 \ldots n_k$). Пусть $n \geq \max(n_1 \ldots n_k)$ – это наименьшее целое число, являющееся степенью простого числа, и $k > n$, то есть число параметров достаточно велико.

2. Построим однородный A' из CA(2; n+1, n), число строк в нем будет равно n^2.

3. Первый тип вспомогательного набора $B(n^2 - 1,\ n + 1,\ n,\ d) = A'$ без первой строки и со столбцами, повторенными последовательно d раз. Число строк в таком наборе равно $n^2 - 1$, а число столбцов – $(n + 1)d$.

4. Второй тип вспомогательного набора $R(n^2 - n\ n,\ n,\ d) = A'$ без первых n строк, без первого столбца и с оставшимися столбцами, повторенными последовательно d раз. Число строк в таком наборе равно $n^2 - n$, а число столбцов – nd.

5. Третий тип вспомогательного набора I(c, d) – это набор, содержащий только единицы и состоящий из c строк и d столбцов.

6. Четвертый тип вспомогательного набора $N(n^2 - n,\ n,\ d)$ – это набор, содержащий набор двоек, размером $n \times d$, к которому по вертикали прибавлен набор троек размером $n \times d$, и т.д. набор, размером $n \times d$, с числом n. В наборе N(c, n, d) c строк и nd столбцов.

7. Строим итоговый набор из вспомогательных за $s = \log_n + {}_1k$ циклов. Если s=1, то из построенного на шаге 2 набора берем k столбцов, – это и будет итоговый набор.

8. Алгоритм выбора вспомогательных наборов и числа их повторов на каждом шаге цикла, описанный в работе [5], достаточно громоздкий, чтобы приводить его целиком.

Полученный набор будет покрывающим, доказательство см. [5], но не обязательно будет минимальным. В [5] приведены примеры, говорящие о том, что построенные этим методом наборы достаточно малы. Оценка размера получаемого набора – $\lceil \log_{n+1}(k) \rceil (n^2 - 1) + 1$.

Временная сложность алгоритма – $O(n^2 + k \cdot \log^2 k)$, необходимая память –

$$O((n^2(n+1) + k(\lceil \log_{n+1}(k) \rceil (n^2 - 1) + 1)) = \\ = O(n^3 + n^2 k \cdot \log_{n+1} k),$$

где $n \geq \max(n_1 \ldots n_k)$ – наименьшее такое число, являющееся степенью простого. Построение нельзя оптимизировать по памяти, поскольку набор нужно строить весь целиком, а не построчно. Этот алгоритм не позволяет расширить уже существующий набор строк.

Нет возможности задать ограничения на комбинации значений параметров, уменьшающие размер набора.

Все последующие методы не зависят от построения однородных покрывающих наборов и могут применяться для построения как неоднородных, так и однородных наборов.

3.2.4 Рекурсивное построение неоднородных наборов глубины 2 [17, 33, 41, 42]

Профиль набора

Отметим символом * в покрывающем наборе элементы, которые не влияют на покрытие всех кортежей размера t (то есть те места, при подстановке в которые любых значений матрица

сохраняет свойства покрывающего набора). Профилем матрицы размера $N \times k$ называется строка размера k, в которой значение d_i равно количеству символов * в i-м столбце матрицы.

Данный алгоритм рекурсивного типа строит покрывающие наборы глубины 2, если существует A из $CA(2; k, v_1, \ldots, v_k)$ размера N с профилем $(d_1, \ldots, d_k)$, а также для всех $i \in [1, k]$ существуют B_i из $CA(2; l_i, m_{i,1}, \ldots, m_{i,l_i})$ размера M_i с профилем $(f_{i,1}, \ldots, f_{i,l_i})$ такие, что $m_{ij} \leq v_i$ для всех $j \in [1, l_i]$. С помощью данного алгоритма при этом можно построить набор из $CA(2; l_1 + \ldots + l_k, m_{1,1}, \ldots, m_{1,l1}, \ldots, m_{k,1}, \ldots, m_{k,lk})$ с числом строк равным
$T = N + \max(M_i - d_i)$.

Краткое описание алгоритма.

1. Пусть есть неоднородный покрывающий набор A из $CA(2; k, n_1, \ldots, n_k)$ размера N с профилем $(d_1, \ldots, d_k)$.

2. Пусть для всех $i \in [1, k]$ есть неоднородные покрывающие наборы B_i из $CA(2; l_{i,mi,1}, \ldots, m_{i,l_i})$ размера M_i с профилем $(f_{i,1}, \ldots, f_{i,l_i})$ такие, что $m_{ij} \leq n_i$ для всех $j \in [1, l_i]$.

3. Первые N строк итогового набора из $T = N + \max_{i=1}^{k}(M_i - d_i)$ строк и $l_1 + \cdots + l_k$ столбцов (его столбцы обозначим как $C_{(i,j)}$, где $1 \leq i \leq k$, $1 \leq j \leq l_i$) строим следующим образом:

3.1. для всех $1 \leq i \leq k$ фиксируем i, теперь l_i – это константа, а A_i – это фиксированный i-ый столбец матрицы A;

3.2. далее для всех $1 \leq j \leq l_i$ столбец итогового набора $C_{(i,j)}$ делаем равным A_i, то есть повторяем A_i, l_i раз и получаем i-ый блок из строк с одинаковыми элементами. d_i строк в каждом i-ом блоке – это *.

4. Строки от N+1 до T в каждом блоке тоже заполним *.

5. В каждом i-ом блоке выберем, начиная сверху M_i строк, целиком состоящих из *. Такая строка будет состоять из l_i элементов.

6. В k-ой выбранной на предыдущем шаге строке j-ому элементу присвоим значение j-ого элемента k-ой строки B_i-ого набора.

7. Если остались строки, состоящие целиком из * или дублирующие уже построенные, удаляем их. Все оставшиеся * заменяем на произвольные значения, допустимые для конкретного столбца.

Полученный набор будет покрывающим, доказательство см. [41]. Он не обязательно будет минимальным, в работе [33] приведен способ уменьшения размера набора.

Временная сложность алгоритма – $O((N + M)L + \sum_{i=1}^{k}(M_i l_i))$, необходимая память – $O((N + M)L + \max_{i=1}^{k}(M_i l_i))$, где N – размер исходного набора, $M = \max_{i=1}^{k}(M_i - d_i)$, $L = \sum_{i=1}^{k} l_i$.

Построение нельзя оптимизировать по памяти, поскольку набор нужно строить весь целиком, а не построчно.

Нет возможности задать семантические ограничения, уменьшающие размер набора.

3.2.5 Оптимизационные алгоритмы

Далее будет рассматриваться семейство оптимизационных, так называемых жадных алгоритмов. Такой тип алгоритмов часто применяется для задач, у которых нет эффективных техник решения. Общий принцип жадных алгоритмов заключается в том, что каждый раз делается выбор, который кажется самым лучшим в данный момент, т.е. производится локально оптимальный выбор в надежде, что он приведет к оптимальному решению глобальной задачи [43]. Для алгоритмов построения покрывающих наборов это означает выбор следующей строки так, чтобы она покрывала как можно большее число ещё непокрытых пар, троек или других наборов значений.

Жадные алгоритмы не всегда приводят к оптимальному решению в общем случае [43]. При построении покрывающих наборов жадные алгоритмы также часто дают не минимальные наборы.

Достоинства жадных алгоритмов:

- построение однородных и смешанных покрывающих наборов для любой конфигурации параметров, некоторые алгоритмы

легко модифицировать для построения покрывающих наборов переменной глубины;

- неплохая временная асимптотика;
- в некоторых случаях возможно построение набора построчно, а не всего целиком;
- жадные алгоритмы могут использоваться как самостоятельно, так и совместно с другими техниками для минимизации получаемых наборов;
- возможность расширить алгоритм заданием ограничений на возможные комбинации значений параметров, уменьшающих набор.

Недостатки жадных алгоритмов:

- построенный набор не всегда минимальный, часто далёк от минимального; для большинства жадных алгоритмов, чем лучше их временная оценка, тем в общем случае построенный набор будет больше;
- в общем случае не существует метода оценки близости построенного набора к минимальному;
- иногда требуются существенные затраты памяти, поскольку необходимо хранить все непокрытые комбинации (пары, тройки и пр.) значений параметров.

Построенные с помощью жадных алгоритмов наборы одной и той же конфигурации могут различаться, поскольку некоторые значения при построении выбираются случайно [18].

Отдельной задачей изучения жадных алгоритмов является поиск тех классов конфигураций покрывающих наборов, для которых конкретный жадный алгоритм выдает минимальные или близкие к минимальному наборы. В данной статье подробный анализ этой темы проводиться не будет, и мы будем считать, что область применения у всех жадных алгоритмов включает в себя любые конфигурации однородных и неоднородных покрывающих наборов с постоянной глубиной.

Теперь рассмотрим некоторые известные жадные алгоритмы более детально.

3.2.5.1 Алгоритм, основанный на добавлении нового параметра (IPO) [13, 18, 22]

Краткое описание алгоритма.

1. Строим покрывающий набор глубины t для первых t параметров (полный перебор сочетаний значений параметров).
2. Добавляем следующий параметр (новый столбец).
3. Рост по горизонтали. Проставляем значения в новый столбец так, чтобы покрыть как можно больше кортежей мощности t, включающих значения нового параметра.
4. Рост по вертикали. Добавляем строки в исходный набор, чтобы покрыть оставшиеся кортежи мощности t.
5. Переходим к шагу 2 для следующего параметра.

Слегка модифицировав этот алгоритм, его можно применять для дополнения уже существующего покрывающего набора из $CA(t; k, n_1 \dots n_k)$ до набора из $CA(t; k+p, m_1 \dots m_k, \dots m_{k+p})$, где $n_i \le m_i$, $1 \le i \le k$, а также $\exists j\ 1 \le j \le k$, $n_j < m_j$ и/или $p > 0$.

В работе [13] описаны алгоритмы роста по горизонтали и по вертикали для t=2. Алгоритм роста по вертикали всегда добавляет минимальное количество тестов для покрытия кортежей мощности 2. Однако в этой статье присутствует ошибка во временной оценке роста по вертикали при t=2: $O(d^4k + d^2k^3)$, где k – количество параметров, $d = \max(n_1 \dots n_k)$, тогда как на самом деле $O(d^4k^2 + d^2k^3)$. В работе описаны два алгоритма роста по горизонтали: один имеет экспоненциальное время работы, но гарантирует минимальность полученного набора, другой полиномиальное – (d^5k^3), – но не гарантирует минимальность. В статье [13] ошибочно указано время (d^5k^2).

В работе [44] алгоритм видоизменен, рост по вертикали и по горизонтали объединены.

1. Пусть к исходному набору из $CA(2; k, n_1 \dots n_k)$ надо добавить еще один параметр с n_{k+1} возможными значениями и $n_{k+1} \le n_k$.

2. Построчно копируем столбец k в столбец k+1 по следующему правилу: если значение, стоящее в i-ой строке k-ого столбца меньше n_{k+1}, то ставим это же значение в i-ую строку k+1-ого столбца, иначе ставим на это место символ *. При копировании, если пара значений из последнего и добавляемого столбца встретилась в первый раз, то помечаем эту строку, как „обязательную“. Таким образом, мы покроем все пары с участием добавляемого столбца и всех столбцов, кроме последнего, а также все пары вида (x, x) с участием последнего и добавляемого столбца. Осталось покрыть еще $n_{k+1}n_k - n_{k+1}$ пар.

3. Для каждой непокрытой пары из столбцов (j, k+1) со значениями (a, b) делаем следующее:

3.1. Выбираем строку p, где в j-ом столбце стоит *, или строку p, не помеченную как „обязательная“, где в j-ом столбце стоит значение a. Если такой строки нет, то добавляем новую строку, где в j-ом столбце стоит a, а в k+1-ом стоит b. В остальных ее столбцах ставим *. Переходим к следующей паре (шаг 3).

3.2. Если строка p нашлась, и в j-ом ее столбце стоит *, ставим туда a. Помечаем строку p как „обязательную“. Ставим значение b в строку p в k+1-ой столбец. Пусть предыдущее значение было равно b'. Если это *, переходим к следующей паре (шаг 3). Иначе для всех пар, которые перестали быть покрыты после замены b на b' (пары из столбцов (h, k+1) со значениями (x, b'), где $1 \le h \le k-1$, x – значение в строке p в столбце h) выполняем шаг 3.

В работе [44] использована неправильная оценка роста покрывающего набора, из-за чего временная сложность посчитана неправильно. Правильное значение временной оценки – $O(d^6k^3\log k)$, необходимой памяти – $O(kd^2\log k + d^2)$, поскольку необходимо держать в памяти весь набор и непокрытые пары для последних двух параметров.

При t>2 алгоритм роста по вертикали из работы [13] модифицирован в работе [22].

1. Непокрытые кортежи мощности t будем обозначать следующим образом: $((p_{k1}, w_1), \ldots (p_{k(t-1)}, w_{t-1}), (p_i, u))$, где i – порядковый номер добавляемого параметра, u - значение этого параметра; p_{kj} – порядковый номер уже добавленного параметра, w_j – значение этого параметра; $k_j < i$, $1 \le j \le t-1$.

2. Набор исходных строк набора обозначим T, а набор новых строк T'. T' изначально пуст.

3. Перебираем все непокрытые кортежи: выбираем следующий кортеж $((p_{k1}, w_1), \ldots, (p_{k(t-1)}, w_{t-1}), (p_i, u))$.

4. Если T содержит строку, где на месте с номером p_{kj0} стоит специальный символ *, обозначающий любое значение, а на тех остальных местах с номерами p_{kj} ($j <> j_0$) стоят либо соответствующие значения w_j, либо тоже символы *, в любом сочетании, то заменяем все символы * в такой строке на соответствующие номерам этих мест значения w_j.

5. Иначе добавляем в T' строку, где на местах p_{kj} стоят значения w_j, а на остальных местах символы *.

6. Переходим на шаг 3, если не все непокрытые кортежи перебраны.

Этот алгоритм добавит необходимое количество тестов для покрытия кортежей мощности t, однако гарантировать минимальность не может. Размер полученного набора будет зависеть от порядка перебора непокрытых кортежей мощности t (шаг 3).

Модифицированный полиномиальный (не гарантирующий минимальность) алгоритм роста по горизонтали для t>2 [22] выглядит следующим образом.

1. Пусть к исходному набору, размер которого – N надо добавить столбец p_i с возможными значениями $v_1, \ldots, v_{ni}$.

2. Если N меньше, чем возможное число значений для нового столбца – n_i, то добавляем к j-ой строке исходного набора значение v_j, $1 \le j \le N$. Построение окончено.

3. Иначе перебираем строки, в которые еще не добавлен столбец.

4. Для каждой строки перебираем все значения столбца p_i.

5. Для каждой пары (расширяемая строка, значение) вычисляем, сколько кортежей мощности t покрыто в результате расширения этой строки этим значением.

6. Выбираем такую пару (расширяемая строка, значение), чтобы число покрытых кортежей было наибольшим. Проставляем в выбранную строку выбранное значение.

7. Убираем из непокрытых кортежей мощности t все, которые становятся покрыты.

8. Если не все строки перебраны, возвращаемся на шаг 5.

В работе [22] шаги 3–7 выглядят сложнее. Там предполагается, что строится однородный набор, благодаря чему возможно использование эвристик, существенно ускоряющих построение набора и уменьшающих объем используемой памяти.

Пусть необходимо расширить однородный покрывающий набор из CA(t; k–1, n) с количеством строк равных r до покрывающего набора из CA(t; k, n). Расширением строки i (длиной k–1) значением v будем называть i-ую строку из исходного набора, к которой слева добавлено значение v. Расширенная строка имеет длину k. Поскольку исходный набор – покрывающий, то все кортежи мощности t в нем уже покрыты, и осталось покрыть только кортежи с участием добавляемого столбца. Расширение строки значением всегда покрывает C_{t-1}^{k-1} кортежей с участием последнего столбца. В работе [22] используются две вспомогательные матрицы, хранящие информацию о покрытых кортежах:

- $T_{i,v}$ – матрица, в которой записано количество кортежей, во-первых, покрытых в результате расширения строки i значением v, а во-вторых, покрытых предыдущими расширениями. Число новых кортежей, покрытых расширением строки i значением v, будет равно $C_{t-1}^{k-1} - T_{i,v}$ (поскольку набор однородный). Размер матрицы T равен r на n.

- $Cov_{\bigwedge,v}$ – булевская матрица, в которой в элементе $\bigwedge$, v стоит «true», если кортеж $\bigwedge$, v уже покрыт. $\bigwedge$ – обозначение кортежа мощности $t-1$ без участия последнего столбца (все такие кортежи можно пронумеровать, число таких кортежей равно $n^{t-1}C_{t-1}^{k-1}$), v – одно из возможных значений от 0 до $n-1$. Размер матрицы Cov равен $n^tC_{t-1}^{k-1}$.

Алгоритм, описанный в работе [22], эффективно манипулирует этими структурами данных, и его временная сложность для расширения покрывающего набора из CA(t; k – 1, n) с количеством строк равных r до покрывающего набора из CA(t; k, n) оценивается как $O(r^2/n^{t-1}C_{t-1}^{k-1} + rn^tC_{t-1}^{k-1})$. Необходимая память оценивается как $O(rn + n^tC_{t-1}^{k-1})$.

Общее время работы алгоритма для построения набора с нуля оценивается как $O(N^2/n^{t-1}C_t^k + Nn^tC_t^k)$, где N – число строк итогового набора. Эти оценки получены в работе [22] с учетом допущений, которые авторы посчитали разумными, опираясь на эмпирические данные. Если ввести еще одно допущение, что алгоритм строит наборы близкие к минимальным и учесть, что число строк близкого к минимальному набору $N \le n^t\log(n^tC_t^k)$, временную сложность можно оценить как

$$O(n^{2t}\log^2(n^tC_t^k)/n^{t-1}C_t^k + n^{2t}\log(n^tC_t^k)C_t^k).$$

Необходимую память как

$$O(n^{t+1}\log(n^tC_t^k) + n^tC_{t-1}^{k-1}).$$

В работе [22] также указана эвристика, которая позволяет строить достаточно малые наборы, если $n < 10$. Эвристика существенно сокращает шаги алгоритма, поддерживающие матрицу $T_{i,v}$ в актуальном состоянии. Вместо обхода матрицы $Cov_{\bigwedge,v}$ и проставления правильных значений в $T_{i,v}$, в эту матрицу проставляется „среднее“значение. Если используется такая эвристика, то временная сложность для расширения покрывающего набора из CA(t; k–1, n) с количеством строк

равных r до покрывающего набора из CA(t; k, n) оценивается как $O(r^2n + rC_{t-1}^{k-1} + rn^tC_{t-1}^{k-1})$. Необходимая память оценивается как O(rn). Общее время работы алгоритма с эвристикой для построения набора с нуля оценивается как $O(N^2nk + NC_t^k + Nn^tC_t^k)$, где N – число строк итогового набора.

В более общем случае неоднородного набора, где указанные выше эвристики применить нельзя, временная оценка модифицированных алгоритмов для добавления одного параметра с количеством возможных значений n_{k+1} в набор из CA(t; k, $n_1 \ldots n_k$) будет отличаться на порядок:

- рост по вертикали: $O(t(m_{k+1}^{2t}(k+1) + m_{k+1}^t (k+1)^2))$,
- рост по горизонтали: $O(m_{k+1}^{2t}(k+1)^2 n_{k+1})$, где $m_{k+1} = \max(n_1 \ldots n_{k+1})$.

Суммарная временная оценка:

- рост по вертикали: $O(t(d^{2t}k^2 + d^tk^3))$,
- рост по горизонтали: $O(d^{2t+1}k^3)$, где $d = \max(n_1 \ldots n_k)$.

Для всего алгоритма IPO: $O(tk^6d^t(d^t + k + + d^tk))$.

Расход памяти будет складываться из размера набора, числа непокрытых кортежей мощности t, а также числа кортежей, покрываемых на каждом шаге. Наибольшее количество памяти алгоритм будет потреблять, когда будет добавляться последний столбец: размер набора, число непокрытых и покрываемых кортежей мощности t будут наибольшими. Размер набора – $O(kd^t \log k)$ [18], число непокрытых и покрываемых кортежей – $O(d^tC_{t-1}^{k-1})$, память необходимая для них, это $O(td^tC_{t-1}^{k-1})$. Суммарная необходимая память – $O(d^t(k \cdot \log k + tC_{t-1}^{k-1}))$, где $d = \max(n_1 \ldots n_k)$.

Построение нельзя оптимизировать по памяти, поскольку набор нужно строить весь целиком, а не построчно.

Исходный алгоритм не предусматривает задания ограничений на возможные комбинации значений параметров, но его несложно модифицировать, чтобы при построении учитывать семантические ограничения, уменьшающие размер набора.

3.2.5.2 Алгоритмы, основанные на выборе наилучших строк из кандидатов [4, 17, 45–47]

Данное семейство алгоритмов оптимизационного типа эффективно применять для дополнения уже существующего покрывающего набора до большего по размеру (в ширину или высоту) набора, включающего в себя исходный набор.

Данное семейство алгоритмов – единственный, известный на сегодняшний день метод построения наборов переменной глубины с нуля или дополнения готовых меньших по размеру наборов. Для реализации построения наборов переменной глубины необходимо изобрести соответствующие эвристики (см. ниже).

Для конкретных конфигураций область применения той или иной эвристики устанавливается экспериментально. Для построения набора с нуля см. работы [4, 17, 23, 45–49]. Для дополнения готового набора – никаких материалов обнаружено не было.

Краткое описание семейства алгоритмов [17].

1. Пусть необходимо построить покрывающий набор из CA(t; k, $n_1 \ldots n_k$).
2. Повторять N_1 раз шаги 3–13.
3. Пока есть непокрытые построенным набором кортежи мощности t выполнять шаги 4–11.
4. Повторять построение следующей строки набора N_2 с помощью шагов 5–11.
5. Последовательно выбираем параметры, пока не останется ни одного не выбранного, выполняя шаги 6–9.
6. Используя правило PR_1 (то есть, эвристический критерий отбора лучшего), из всех k параметров выбираем множество параметров P, в которые будем в первую очередь вставлять значения.
7. Используя правило PR_2, из всех параметров множества P выбираем один параметр – p_i, в который будем в первую очередь вставлять значения. Возможные значения параметра p_i, это числа от 0 до n_i.

8. Используя правило VR_1 выбираем из всех значений параметра p_i множество значений V, которые будем считать наиболее подходящими.

9. Используя правило VR_2, выбираем из множества V одно значение v и проставляем параметру p_i значение v.

10. После последовательного повторения шагов 5–9 для всех параметров получается строка – тест T.

11. Если эта строка покрывает больше кортежей мощности t, чем построенная в прошлый раз, то оставляем её. Возвращаемся к шагу 4.

12. После шагов 3–11 получается покрывающий набор CA.

13. Если число строк в этом наборе меньше, чем в построенном в прошлый раз, то оставляем этот набор. Возвращаемся к шагу 2.

14. После N_1 раз повторенного построения получаем набор для конфигурации $(t; k, n_1 \ldots n_k)$.

Поскольку выбор строки (теста T), покрывающего наибольшее число кортежей мощности t (шаги 5–10), – это NP-полная задача [12], то необходимы эвристики PR_1, PR_2, VR_1 и VR_2 чтобы решить эту задачу за полиномиальное время. В работе [17] доказан логарифмический рост покрывающего набора относительно числа параметров k. Основываясь на этом доказательстве, была выдвинута гипотеза о том, что для эффективного построения покрывающего набора, необязательно в шагах 5–10 строить строку покрывающую наибольшее число кортежей мощности t, а достаточно построить строку покрывающую *среднее число* кортежей мощности t. Назовем это число δ. В работе [17] оно вычисляется $\delta = \sum_{1 \le p1 < \ldots < pt \le k} \delta_{p1,\ldots,pt}$, где $\delta_{p1,\ldots,pt} = r_{p1,\ldots,pt}/(n_{p1} \cdot \cdots \cdot n_{pt})$, $r_{p1,\ldots,pt}$ – число непокрытых кортежей мощности t, включающих в себя значения p_1-ого ... p_t-ого параметра, $\delta_{p1,\ldots,pt}$ – называется локальной плотностью, δ – глобальной.

Возможные варианты эвристических критериев отбора лучшего для PR_1 :

1. Лучшим считается тот параметр, у которого больше возможных значений. Этот критерий обозначим MAX_N.

2. Лучшим считается тот параметр, который включен в большее число непокрытых на данном этапе кортежей мощности t. Например, пусть есть непокрытые пары $(v_{2,1}, v_{3,2})$, $(v_{1,2}, v_{2,2})$ и $(v_{2,2}, v_{3,1})$, где $v_{i,j}$ – это j-ое значение i-ого параметра. Тогда наилучшим параметром для выбора в соответствии с этим критерием будет 2-ой (i=2), – он встречается 3 раза. Этот критерий учитывает только непокрытые кортежи, оставшиеся после того, как зафиксированы предыдущие параметры. Обозначим этот критерий P_IN_MAX_UC.

3. Лучшим считается тот параметр, который включен в определенное, равное σ (не обязательно максимальное), число непокрытых кортежей мощности t. Этот критерий более гибкий и позволяет учитывать (при правильном выборе σ) не только уже зафиксированные параметры, но и ещё не зафиксированные (см. [17]). Это критерий σ.

4. Лучший параметр выбирается случайным образом. Обозначим этот критерий RAND.

Поскольку под критерий лучшего согласно PR_1 может подойти несколько параметров, то необходим PR_2 для выбора одного из множества.

Возможные варианты эвристических критериев отбора лучшего для PR_1 :

1. Взять первый параметр по порядковому номеру. Это критерий BY_ORD.

2. Взять случайным образом.

3. Для разрешения коллизий 1-го и 3-го вариантов PR_1 можно в качестве PR_2 использовать 2-ой вариант PR_1 (то есть, быть жадным ещё раз), а затем один из первых двух критериев PR_2 для того, чтобы выбрать единственный вариант.

Возможные варианты эвристических критериев отбора лучшего для VR_1 :

1. Лучшим считается то значение, которое покрывает наибольшее число непокрытых на данном этапе кортежей мощности t для уже зафиксированных параметров. Обозначим такой критерий V_IN_MAX_UC.

2. Лучшим считается то значение, которое покрывает определенное, равное δ (не обязательно максимальное), число непокрытых кортежей мощности t. Этот критерий обозначим δ.

3. Лучшее значение выбирается случайным образом.

VR_2 выбирается аналогично PR_2.

Большинство существующих на данный момент коммерческих и доступных свободно инструментов, генерирующих комбинаторные тестовые наборы, используют именно жадные алгоритмы для построения покрывающих наборов. См. таблицу 7, в которой показаны использующиеся эвристики.

Замечания к таблице 7.

* На первом проходе цикла выбора параметров всегда выбирается такой, у которого значение входит в наибольшее число непокрытых на данном этапе кортежей мощности t.

** Для t параметров и t значений для этих параметров действуют правила P_IN_MAX_UC и V_IN_MAX_UC, соответственно, выбираются t параметров и значения в них такие, что .они содержат параметры и значения входящие в наибольшее число непокрытых кортежей. Для всех остальных пар (параметр, значение) строится вектор $(p_0, \ldots p_{t-1})$, где p_i – число непокрытых кортежей (без учета уже зафиксированных вначале $t-1-i$ значений) мощности t, в которые данная пара входит. Выбирается лучшая пара, у которой такой вектор будет иметь большее значение. Вектора сравниваются лексикографически. Если находится несколько пар с одинаковыми векторами, то лучший выбирается случайно.

*** Нет эвристики, т.е. выбирается строка, покрывающая наибольшее число кортежей мощности t. Это в общем случае требует экспоненциального времени.

Некоторые вариации жадных алгоритмов (ATGT [8], PICT [26]) для построения каждой строки перебирают не параметры и значения, а непокрытые кортежи мощности t – t параметров и t значений. Несколько выбранных кортежей формируют строку теста. Эти алгоритмы используют следующие эвристики для перебора кортежей:

1. Случайный перебор. В этом случае, алгоритм перезапускается несколько раз и выбирается лучшее решение.

2. Перебор в лексикографическом порядке. Этот вариант не дает хороших результатов.

3. Кортеж t_1 считается лучше (выбирается раньше) кортежа t_2, если пары параметр-значение, которые присутствуют в кортеже t_1, встречаются реже в уже покрытых кортежах, чем пары параметр-значение кортежа t_2.

4. Кортеж t_1 считается лучше кортежа t_2, если наименее использованная в предыдущих тестах пара параметр-значение кортежа t_1 встречается реже в уже сгенерированных тестах, чем наименее использованная в предыдущих тестах пара параметр-значение кортежа t_2.

5. Кортеж t_1 считается лучше кортежа t_2, если наиболее использованная в предыдущих тестах пара параметр-значение кортежа t_2. встречается реже в уже сгенерированных тестах, чем наиболее использованная в предыдущих тестах пара параметр-значение кортежа t_2. Этот, на первый взгляд, обратный предыдущему критерий не всегда дает обратный порядок кортежей.

6. Кортеж t_1 считается лучше кортежа t_2, если его пары параметр-значение

Таблица 7. Эвристики инструментов генерации покрывающих наборов.

	PR_1	PR_2	VR_1	VR_2	N_1	N_2
AETG [4, 45]	RAND*	–	V_IN_MAX_UC	RAND	≤ 50	≤ 50
TCG [47]	MAX_N	BY_ORD	V_IN_MAX_UC	BY_ORD	1	$\max(n_1 \ldots n_k)$
DDA [46]	Δ	BY_ORD	Δ	BY_ORD	1	1
CTS [16, 19]	P_IN_MAX_UC**	RAND	V_IN_MAX_UC**	RAND	1	1
Jenny[48]	RAND	–	***	–	1	1

более равномерно распределены в предыдущих тестах. Для каждой пары считается среднеквадратичное отклонение. Этот критерий основывается на тех же соображениях, что используются в алгоритме DDA [46]: каждая новая строка-тест должна покрывать среднее число кортежей (δ).

Последние четыре критерия можно использовать двумя способами: выбирать кортежи независимо друг от друга или при выборе следующего кортежа учитывать уже использованные в построенной части тестовой строки пары параметр-значение. Второй способ показывает более хорошие результаты, но имеет большие временные затраты.

Инструменты, использующие жадные алгоритмы, для которых не было найдено информации об использующихся эвристиках [49]: CATS, TestCover.com, CaseMaker, Pro-test.

В работе [23] проведен анализ использующихся в промышленных инструментах эвристик, и было принято решение в качестве PR_1 использовать P_IN_MAX_UC, в качестве PR_2 – RAND, в качестве VR_1 – V_IN_MAX_UC, в качестве VR_2 – δ, $N_1 = 5$, $N_2 = 1$. Кроме того, вместо повторения построения очередного теста, в работе используется эвристический алгоритм увеличения количества покрываемых кортежей каждым вновь построенным тестом. Иными словами, вместо шага 11 приведенного выше алгоритма, выполняется эвристический поиск локального экстремума в строке. В ряде частных случаев такая методика позволяет строить меньшие покрывающие наборы за меньшее время, нежели перечисленные в таблице 3 промышленные инструменты. Используя этот метод удалось построить наименьшие известные на 2007 год покрывающие наборы для конфигураций, где количество значений параметров равно 2, а глубина 5–8 [37].

Данное семейство алгоритмов имеет полиномиальное время работы, которое зависит от применяемых эвристик. В общем случае время работы можно оценить так:

$$O(k(f(PR_1) + f(PR_2))d(f(VR_1) + f(VR_2))),$$

где $d = \max(n_1, \ldots, n_k)$, $f(PR_1)$, $f(PR_2)$, $f(VR_1)$, $f(VR_2)$ – функции времени работы соответствующих эвристик.

Расход памяти рассчитывается аналогично алгоритму IPO: $O(d^t(k \cdot \log k + tC_{t-1}^{k-1}))$.

Набор можно строить построчно, однако необходимо хранить все уже построенные строки.

Алгоритм допускает задание ограничений на возможные комбинации значений параметров. В работах [8, 24–26] показано, как можно модифицировать практически любой жадный алгоритм из данного семейства для того, чтобы учитывать такие ограничения. Кроме того, показано, как их можно задавать в виде логических предикатов, а также сравниваются эффективность использования различных логических решателей SAT[8, 24, 25].

3.2.6 Решение оптимизационной задачи

Далее будут рассматриваться алгоритмы, основанные на решении следующей оптимизационной задачи: пусть у нас есть множество Σ возможных решений, а также функция оценивания $c(S)$, определённая для всех $S \in \Sigma$. Оптимальное решение отвечает возможному решению с минимальной функцией оценивания. Трансформация T – это действие, которое применяется к S для получения нового возможного решения.

Задачу построения покрывающего набора можно свести к рассмотренной так: Σ –

множество всех наборов (возможно, с непокрытыми комбинациями), S – один из наборов, c(S) – число непокрытых комбинаций, если c(S)=0, то S – покрывающий набор. T – изменение значений элементов набора S.

3.2.6.1 Поиск экстремума (Hill Climbing) [17]

Данный алгоритм оптимизационного типа эффективно применять только для улучшения (то есть, уменьшения) существующих тестовых наборов для любых конфигураций. В качестве начального решения выбирается уже существующий тестовый набор.

Краткое описание алгоритма.

1. Повторяем построение набора не менее N_1 раз или до тех пор, пока не построим покрывающий набор – шаги 2–7.
2. Случайным образом выбирается начальное решение. Оно становится текущим решением S.
3. Производится случайная трансформация текущего решения. Получаем решение S′.
4. Если $c(S') \leq c(S)$, то S′ принимается за текущее решение.
5. Если выполнено условие останавливающей эвристики, то переходим к шагу 6, иначе возвращаемся к шагу 3.
6. После шагов 2–5 получаем решение S – не обязательно покрывающий набор.
7. Если построенный набор покрывающий и его размер меньше, чем размер построенного в прошлый раз, то оставляем его. Возвращаемся к шагу 2.
8. После N_1 раз (или более) повторенного построения получаем искомый набор.

В общем случае алгоритм имеет экспоненциальные временную оценку и затраты по памяти.

Построение нельзя оптимизировать по памяти, поскольку набор нужно строить весь целиком, а не построчно.

Исходный алгоритм не предусматривает задания ограничений, уменьшающих набор или ускоряющих его построение.

В работе [2] алгоритм модифицирован для работы с покрывающими наборами переменной глубины. Он успешно применяется для небольших значений параметров конфигурации покрывающих наборов.

3.2.6.2 Поиск с ограничениями (Tabu Search) [17, 50, 51]

Данный алгоритм оптимизационного типа эффективно применять в тех же случаях, что и алгоритм поиска экстремума. Однако, за счёт того, что есть возможность задавать ограничения, алгоритм может работать немного быстрее для реальных задач [50].

Алгоритм повторяет идею поиска экстремума с некоторыми отличиями.

1. Повторяем построение набора N_1 раз для конфигурации Q с числом параметров k, для которой известны верхняя и нижняя границы размера набора (N_{min}, N_{max}).
2. Случайным образом выбираем начальное решение – набор, размер которого находится в пределах ((N_{min}, N_{max}). Оно становится текущим решением S.
3. Производится трансформация текущего решения следующим образом.

3.1. Случайным образом выбираем непокрытый кортеж мощности t из множества непокрытых кортежей. Обозначим его UT.

3.2. Выбираем строки из решения S такие, что изменения только одного элемента строки покрывают UT. Если таких строк нет, выбираем одну строку случайным образом.

3.3. Для каждой выбранной строки заменяем элемент (или элементы) так, чтобы UT был покрыт и вычисляем стоимость нового решения (с заменённым элементом), то есть число непокрытых кортежей мощности t.

3.4. Из всех решений выбираем решения с наименьшей стоимостью и не противоречащих используемым ограничениям.

3.5. Если таких решений несколько, выбираем решение S′ случайно. S′ принимается за текущее решение.

4. Если S' – покрывающий набор, то запоминаем S', затем выбираем новое случайное начальное решение с числом строк на одну меньшим, чем у S', и возвращаемся к шагу 3. Если число циклов по построению покрывающего набора превысило N_2, то переходим к шагу 5, иначе возвращаемся к шагу 3.

5. Если у нас есть покрывающий набор, построенный на предыдущих шагах, то оставляем его, иначе выбираем новое случайное начальное решение с числом строк на одну большим, чем у S', и возвращаемся к шагу 3.

6. После шагов 2–5 получаем решение S – покрывающий набор.

7. Если размер построенного набора меньше, чем построенного в прошлый раз, то оставляем его. Возвращаемся к шагу 2.

8. После N_1 раз повторенного построения получаем искомый покрывающий набор.

В работе [50] описано следующее ограничение (см. шаг 3.4): если решение S' становилось текущим в предыдущих T циклах – шаги 3–4 ($1 \le T \le 10$), то оно не может стать текущим. Это условие позволяет алгоритму не зациклиться.

Также можно задавать и другие ограничения. Они устанавливают больший вес для изменений, которые достигают некоторых желательных свойств.

В общем случае алгоритм имеет экспоненциальную временную оценку. Если известны верхняя и нижняя оценки размера набора N_{min}, N_{max}, то временная сложность равна $O(N_1 k N_{max} d(k + \log T) + dk N_{max}(T + k^2))$, где d – максимальное число допустимых значений параметров, затраты по памяти – $O(kN_{max}T)$, где T – число хранимых наборов для реализации ограничений. Оптимизировать использование памяти нельзя, поскольку набор нужно строить весь целиком, а не построчно.

Эти оценки существенно превышают оценки для оптимизационных алгоритмов других семейств (например, потому что N_1 должно быть достаточно большим), поэтому алгоритм целесообразно использовать только в исследовательских целях.

3.2.6.3 Симуляция отжига (Simulated Annealing) [17, 39, 51–53]

Данный алгоритм оптимизационного типа эффективно применять в тех же случаях, что и алгоритм поиска экстремума. Алгоритм защищён от зацикливания в локальном минимуме, что теоретически позволяет найти лучшее решение, чем простой поиск экстремума. Алгоритм имеет вероятностную основу, необходимо провести дальнейшие исследования (собрать статистику) для того, чтобы понять, для каких конфигураций он работает наилучшим образом.

В работе [39] симуляция отжига эффективно применяется для построения исходных покрывающих наборов и наборов специального вида для последующего использования их в рекурсивных конструкциях. Эти рекурсивные конструкции описаны в пунктах 3.1.6-3.1.9 настоящей работы. Такая комбинация позволяет получать достаточно небольшие (по сравнению с другими методами) покрывающие наборы за хорошее время.

Краткое описание алгоритма.

Это обобщение алгоритма поиска экстремума, которое позволяет с некоторой вероятностью выбирать очередное решение с худшим значением оценочной функции. Худшее решение выбирается с вероятностью $e^{-(c(S')-c(S))/KT}$, где c() – оценочная функция, K – константа, а T – изменяемый в ходе работы параметр, который называют ***степенью нагрева.*** Это позволяет алгоритму не „застрять“в локальном минимуме, а продолжить поиск глобального. Степень нагрева уменьшается небольшими шагами, позволяя достичь состояния равновесия посредством последовательности преобразований текущего решения. Обычно уменьшение степени нагрева производится в виде $T = \alpha T$, где α – число, немного меньше 1. Как только подходящее условие останова достигается, текущее решение принимается за приближенное решение задачи.

В общем случае алгоритм имеет экспоненциальную временную оценку. Если известны верхняя и нижняя оценки границ размера

набора N_{min}, N_{max}, то его временная сложность – $O(N_1k + k^3N_{max})$, затраты по памяти – $O(kN_{max})$. Построение нельзя оптимизировать по памяти, поскольку набор нужно строить весь целиком, а не построчно.

Эти оценки существенно превышают оценки для алгоритмов других семейств, поэтому алгоритм целесообразно использовать только в исследовательских целях.

Исходный алгоритм не предусматривает задания ограничений, уменьшающих набор или ускоряющих его построение.

3.2.6.4 Великий потоп (Great Deluge) [17, 54, 55]

Этот алгоритм принадлежит к семейству алгоритмов достижения предельных значений (threshold accepting). Данный метод похож на алгоритм симуляции отжига, с той разницей, что вместо использования вероятности при принятии решения с худшим значением оценочной функции, это решение принимается тогда, когда его цена не более некоторого текущего значения, называемого уровнем воды. По мере работы алгоритма уровень воды постепенно снижается, ограничивая количество возможных трансформаций. Часто данный метод показывает более быструю сходимость к решению [54, 55].

Описанные в разделе 3.2.6 алгоритмы из-за плохих временных оценок почти не используются в промышленных инструментах. Однако они часто позволяют построить наборы, более близкие к оптимальным, чем жадные алгоритмы, и поэтому чаще используются в исследовательских целях. При выборе оценочной функции другого вида их можно использовать для построения покрывающих наборов переменной глубины.

3.2.7 Генетические алгоритмы [17, 51]

Основная идея генетических алгоритмов – управление популяцией решений и ее развитие с использованием двух операций: мутации и скрещивания. Мутации вносят локальные изменения в какое-либо решение из популяции, а скрещивание объединяет часть одного решения с частью другого решения. Выживание решения в новом поколении определяется полезностью данного решения относительно других решений в поколении.

Для случая покрывающих наборов популяция решений – это множество наборов (не обязательно покрывающих) с заданным размером N. N – параметр для очередного запуска алгоритма. Размер популяции поддерживается постоянным, равным M. На каждом шаге алгоритма происходят скрещивание и мутация. Скрещивание – выбор двух случайных наборов и рекомбинация их строк для получения двух новых наборов, обладающих свойствами обоих родителей. Мутация – полученные новые наборы изменяются, например, по правилам шага 3 алгоритма поиска с ограничениями, или по каким-либо другим. Потом происходит выбраковка, – остаются только M лучших наборов, у которых функция полезности наилучшая, например, число непокрытых кортежей мощности t наименьшее.

Применение генетических алгоритмов для поиска покрывающих наборов на данный момент слабо исследовано. В работ [51] Stardom опубликовал первые результаты по применению генетических алгоритмов для поиска покрывающих наборов. Временная оценка такого алгоритма – $O(N_1Mk^2(N_{max} + d^2))$, где N_1 – число запусков алгоритма, $d = \max(n_1 \ldots n_k)$, N_{max} – верхняя оценка размера набора. Необходимая память – $O(kMN_{max})$.

В работе [56] описан „миметический“генетический алгоритм для конфигураций вида (3, k, 2). В качестве начальной популяции выбирается M наборов (M делится на 4), в которых число 0 и 1 в каждом столбце одинаковое (или отличается на 1, если число строк нечетно), 0 и 1 проставляются случайным образом. M наборов делятся на 4 множества с одинаковым числом наборов. Две пары множеств скрещиваются по следующему правилу: случайным образом выбирается строка с номером i, и затем $C_{mn} = A_{mn}(D_{mn} = B_{mn})$, если $m \le i$; и $C_{mn} = B_{mn}(D_{mn} = A_{mn})$, если $m > i$, где A_{mn} и B_{mn} – это родители, а C_{mn} и D_{mn} – потомки. Для мутации используется алгоритм поиска локального минимума непокрытых троек в наборе, основанный на алго-

ритме симуляции отжига. В процессе мутации получается несколько наборов. В качестве окончательной мутации выбирается набор с наименьшим количеством непокрытых троек и наиболее далекий от исходного, где расстояние измеряется количеством различных элементов наборов (подобно расстоянию Хэмминга). После проведения скрещиваний и мутаций остается M наборов с наименьшим числом непокрытых троек. Этот алгоритм позволил для нескольких конфигураций построить наборы меньшие, чем известные до этого. Однако в работе [56] не приведены сравнения по быстродействию алгоритма с другими техниками, потенциально способными дать настолько же качественные наборы, например, симуляцией отжига и поиском с ограничениями. Вопрос о том, можно ли получить настолько же хорошие результаты другими методами за меньшее время и с меньшим расходом памяти, остается открытым.

Опубликованные результаты говорят не в пользу генетических алгоритмов, поскольку для большинства конфигураций время работы генетического алгоритма, необходимое для генерации покрывающего набора больше, чем время, необходимое симуляции отжига или поиску с ограничениями для той же конфигурации.

3.2.8 Поиск с возвратом [27]

Данный тип алгоритмов использует определенные правила для последовательного перебора всевозможных значений, стоящих в ячейках набора, для заданной конфигурации $(t; k, n_1, \ldots, n_k)$, $n_1 \geq \cdots \geq n_k$ и размера набора N. Поскольку время работы и необходимая память для таких алгоритмов достаточно велики при построении набора с нуля, их целесообразно использовать только в исследовательских целях, для небольших значений параметров конфигурации или для дополнения уже существующего набора, при условии, что известна верхняя граница его размера N.

Краткое описание алгоритма.

1. Начинаем построение набора с некоторого исходного набора, в общем случае это может быть полный перебор t первых параметров (у которых наибольшее число возможных значений).
2. Перебираются все остальные ячейки матрицы $k \times N$. Используя эвристику CH, выбирается очередное значение в очередной ячейке набора.
3. Шаг 2 повторяется, пока набор не будет построен полностью, либо сработает останавливающая эвристика SH.
4. Если полученный набор – покрывающий, то алгоритм завершается, иначе набор откатывается в предыдущее состояние, и алгоритм возвращается к шагу 2.

Предложенные в работе [27] эвристики, ограничивающие выбор ячейки и значения таковы.

- Все векторы, представляющие из себя строки и столбцы набора, должны находиться в лексикографическом порядке. Иными словами, пусть i_1 и i_2 – номера строк, и $i_1 < i_2$, тогда вектор из значений ячеек $(v_{i1,1}, \ldots, v_{i1,k}) \leq (v_{i2,1}, \ldots, v_{i2,k})$. Аналогично для столбцов.
- Пусть все возможные значения для каждого из столбцов j упорядочены от 0 до n_j, и M_j – наибольший порядковый номер из значений, уже проставленных в столбце j. Тогда следующее значение в столбце j может иметь порядковый номер от 0 до $M_j + 1$.
- Выбирается число s, $1 \leq s \leq t$. Для каждого кортежа мощности s вычисляется частота его появления в некотором множестве столбцов. Частоты для одного и того же множества столбцов не должны отличаться более, чем на 1.
- Выбранные значения должны удовлетворять заданным пользователем ограничениям на возможные комбинации значений параметров.
- Если несколько значений для очередной ячейки не удовлетворяют перечисленным выше ограничениям, то есть несколько способов выбрать одно из них:

- выбрать лексикографически первое;
- выбрать случайное;
- выбрать значение, покрывающее наибольшее число непокрытых кортежей мощности t. В работе [27] указано, что этот критерий эффективен лишь для некоторых классов конфигураций, однако требует больших вычислительных затрат, нежели предыдущие два.

Предложенная в работе [27] останавливающая эвристика такова. Пусть есть множество X – непокрытые кортежи мощности t, содержащие столбец c, и множество Y кортежей мощности t, содержащие столбец c, в которых значения в столбце c еще не проставлены. Вычисляем множество кортежей, соседних к X, N(X), то есть, кортежей из Y, отличающихся от X только значением в столбце c. Если мощность множества N(X) меньше мощности множества X, то каким бы образом не были проставлены значения в ячейках кортежей из Y, все кортежи X покрыть не получится. Требуется возврат на предыдущий шаг.

Все эти эвристики можно использовать одновременно, и получаемый набор будет покрывающим, доказательство см. [27].

Необходимое время оценить достаточно сложно, поскольку нет данных о том, насколько предложенные эвристики сократят перебор. В общем случае, можно оценить время работы так: $O(MNkf(CH)f(SH))$, где $M \geq k(N - N_{init})$ – число перебранных наборов, f(CH), f(SH) – время работы соответствующих эвристик, N_{init} – размер исходного набора. Необходимая память: $O(Nk^2(N - N_{init}) + d^t C_k^t)$, $d = \max(n_1 \ldots n_k)$, поскольку необходимо держать в памяти все наборы во всей ветке выбора, а также информацию обо всех кортежах мощности t.

Алгоритм предусматривает задание ограничений на возможные комбинации, а также позволяет их использовать в эвристиках, что сокращает время работы алгоритма.

3.2.9 Оптимизация заданного покрывающего набора [28]

Данный гибридный алгоритм, который можно одновременно отнести к редуцирующим и оптимизационным, эффективно применяется [28] для улучшения (уменьшения) существующих тестовых наборов для любых конфигураций. Идея алгоритма заключается в поиске элементов исходного набора из $CA(t; k, n_1, \ldots, n_k)$, не влияющих на покрытие кортежей мощности t. Эти элементы можно заменить на символ *, означающий, что на это место можно поставить любое допустимое значение, и набор не перестанет быть покрывающим. Пусть N – число строк в исходном наборе.

Краткое описание алгоритма.

1. Все элементы исходного набора считаем кандидатами на замену на символ *.
2. Из k столбцов исходного набора выбираются все возможные упорядоченные сочетания из t столбцов $(c_1, \ldots, c_t)$. Таких сочетаний будет C_t^k. Для каждого сочетания столбцов выполняем шаги 3–7.
3. Проходим по всем строкам исходного набора. Для каждой строки r значения, стоящие в столбцах $(c_1, \ldots, c_t)$ строки r образуют вектор $(v_1, \ldots, v_t)$. Если этот вектор встретился первый раз, то элементы набора в строке r и столбцах $(c_1, \ldots, c_t)$ убираем из кандидатов на замену на *.
4. После завершения итераций шагов 2–3, все элементы, оставшиеся в кандидатах на замену на *, заменяем на *.
5. Если после шага 4 в наборе оказались строки, состоящие целиком из *, удаляем их.
6. Строка, в которой оказалось наибольшее число * ставится на последнее место в наборе. Запоминаем это число.
7. Для всех остальных строк, содержащих *, идем по всем элементам строки r, где стоит * (пусть этот элемент стоит в столбце c). Если значение, стоящее в столбце c в последней строке, это *, то заменяем элемент строки r в столбце c на случайное значение. Иначе заменяем его на значение, стоящее в столбце c в последней строке.
8. Случайным образом переставляем все строки, кроме последней. Получаем новый исходный набор. Возвращаемся к шагу 2.

9. Если после повторений шагов 2–8 M_1 раз ни одна строка не была удалена, или число * в последней строке не увеличилось, то сохраняем получившийся набор.

10. В качестве нового набора выбирается полученный на предыдущем шаге набор, в котором последняя строка становится первой, и случайная строка, в которой есть * становится второй. В первой и второй строке все * заменяются на случайные допустимые значения. Возврат на шаг 2.

11. Шаги 2–10 повторяются M_2 раз. При каждом повторе остается только меньший из сохраненного и полученного наборов.

Временная оценка – $O(M_2(M_1(C_t^k tN + Nk + N - -1) + 6k))$, необходимая память – $O(Nk + d^t)$, $d = \max(n_1 \ldots n_k)$. Построение нельзя оптимизировать по памяти, поскольку набор нужно строить весь целиком, а не построчно. Алгоритм не предусматривает задания ограничений на возможные комбинации значений параметров.

3.3. Результаты обзора

В таблицах 8–10 собраны все описанные методы построения покрывающих наборов, указаны конфигурации, для которых их целесообразно использовать.

Использованные обозначения в заголовке таблиц:

- **M** – является ли построенный набор минимальным;
- **E** – можно ли применить алгоритм для расширения исходного набора;
- **S** – допускает ли алгоритм добавление семантических ограничений.

Использованные обозначения в теле таблиц:

- $d = \max(n_1 \ldots n_k)$, для конфигураций вида $(t;\ k,\ n_1 \ldots n_k, \ldots)$;
- N_e – число запусков алгоритма;
- N_{max} – верхняя оценка получаемого размера набора.

Для рекурсивного алгоритма построения однородных наборов глубины t=2:

- $k_1 = 1,\ k_2 = m - 1,$ N – число строк в исходном покрывающем наборе из CA(2; m, n), полученном другими методами;
- $k_1 = n,\ k_2 = 1,\ N = n^2$;
- $k_1 = l,\ k_2 = 1,\ N = (l + 1)(n - 1) + 1,$ для чисел n и l существует особый (n,l)-исходный вектор покрытия;
- n=3, $(k_1 = 14,\ k_2 = 1,\ N = 15)$, $(k_1 = 60,\ k_2 = 14,\ N = 21)$, $(k_1 = 220,\ k_2 = 114,$ N=27), $(k_1 = 1092,\ k_2 = 220,$ N=33).

Прямые алгоритмы построения целесообразно применять для создания покрывающих наборов с небольшим числом параметров и для некоторых частных случаев. Их основным достоинством является то, что они дают близкие к минимальным (или даже минимальные) наборы за хорошее время. Основными недостатками являются крайне узкая область применеия и отсутствие возможности отсеивания строк наборов, не удовлетворяющих ограничениям на возможные комбинации.

Рекурсивные алгоритмы хорошо дополняют прямые для особых классов конфигураций исходных наборов. Например, метод Roux дает близкий к минимальному набор из CA(3; 20, 9) с числом строк равным 1377, используя набор из CA(3; 10, 9) с числом строк равным 729 и набор из CA(2; 10, 9) с числом строк равным 81. Обе исходные конфигурации принадлежат классу $(t;\ p^k + 1,\ p^k)$. Однако для других классов конфигураций или, если исходные наборы не минимальные, результирующий набор может оказаться существенно больше, чем построенный, например, жадным алгоритмом [37]. Рекурсивными алгоритмами надо пользоваться осторожно и только в некоторых частных случаях. Достоинство рекурсивных алгоритмов в том, что они для частных случаев дают покрывающие наборы близкие к минимальным за хорошее время, недостатками являются узкая область применения и отсутствие возможности задавать ограничения.

Таблица 8. Алгоритмы построения однородных наборов.

Алгоритм	Область применения	Класс алгоритма	Временная сложность и затраты по памяти	M	E	S
Булевский	(2; k, 2)	Прямой	O(k), Память: $k\log_2 k$	+	-	-
Аффинный	(t; n+1, n), n – степень простого числа; (3; 2k+2, 2k),	Прямой	$n^t + (n+1)\log_2(n+1)+$ $+O(\log_p^4 n)$ Память: $O(n) + O(\log_p^3 n)$	+	-	-
Действия групп	(3; 2k, n+1), n – степень простого числа	Прямой	$O(2k((2k-1)(n^3-n)+n+1)+n^3)$ Память: $O(2k(2k-1)+n^4-n^2)$	-	-	-
	(2; 4, n), где $n \equiv 2 \pmod 4$		$O(4(n+1)n)$, Память: $O(4(n+1)+n^2)$,			
	(2; l, g), (g, l)-исходный вектор покрытия, (2; l+1, g), особый (g, l)– исходный вектор покрытия		$O(l(l(g-1)+1))$ Память: $O(l+(g-1)g)$			
Понижение глубины	(t–1; k–1, n) из (t; k, n)	Редуцирующий	O(Nk), Память: O(Nk)	-	-	-
Умножение	(t; k, n_1n_2) из (t; k, n_1) и (t; k, n_2), N_1, N_2 – число строк в исходных наборах	Рекурсивный	$O(k(1+N_1+N_2))$ Память: $O(k(1+N_1+N_2))$	-	-	-
Однородный, рекурсивный, t=2	(2; $k_1n^r + rk_1n^{r-1} + k_2n^r$, n), r ≥ 1, n – степень простого числа	Рекурсивный	$O(\sum_{i=1}^{r}((N+i(n^2-n))$ $(k_1n^i + ik_1n^i + k_2n^i)))$ Память:$O((N+(r-1)(n^2-n))$ $(k_1n^{r-1} + (r-1)k_1n^{r-2} + k_2n^{r-1})+$ $+(n+1)n^2)$	-	-	-
	(2; $n(k_1+k_2)D_{r+1,n} + k_1D_{r,n}$, n), r ≥ 1, n – степень простого числа, $D_{r,t} = \sum_{i=1}^{\lfloor (r+1)/2 \rfloor} C_{i-1}^{r-i}t^{r-i}$		$O(\sum_{i=1}^{r}((N+i(n^2-n))$ $(n(k_1+k_2)D_{i+1,n} + k_1D_{i,n}))$ $+\sum_{i=1}^{r-1}(n^2(n(k_1+k_2)D_{i,n} + k_1D_{i-1,n})))$ Память:$O((N+(r-1)(n^2-n))$ $(n(k_1+k_2)D_{r,n} + k_1D_{r-1,n}) + (n+1)n^2)$			
Ordered design	(3; n+1, n+1) из OD(3, n+1, n+1) и (3; n+1, 2) размера N, n – степень простого числа	Рекурсивный	$O((n^3-n+Nn(n+1)/2(n^2-1))$ $(n+1))$ Память: $O((n^3-n+N)(n+1))$	-	-	-
Семейство совершенных хэш-функций	(t; k^{2p}, n) из (t; k, n) размера N, НОД((t–1)t/2, k)=1 (3; $(2v-1)^{2\wedge j}$, v), $v \equiv 0,\ 1 \bmod 3$, (3; $(2v-3)^{2\wedge j}$, v), $v \equiv 2 \bmod 3$, v>2, q ≥ v − 1–степень простого числа, j–любое целое	Рекурсивный	$O(k^p((t-1)t/2+1)+$ $+k^{2p}((t-1)t/2+1)^pN)$ Память: $O(k^p((t-1)t/2+1+N))$	-	-	-
	(t; k, n) из PHF $(N_1, t; k, m)$ и (t; m, n) размера N_2		$O(N_1N_2k)$ Память: $O(N_1k+N_2m)$			
Теорема Roux	(3; 2k, n) из (3; k, n) и (2; k, n)(t; 2k, n) из (t; k, n),..., (t–2; k, n), t ≥ 4	Рекурсивный	$O(2N_{tmax}k(n+N_{tmax}(t-3)))$ Память: $O(2kN_{tmax}^2)$, N_{tmax} – число строк в наибольшем исходном наборе	-	-	-
IPO для наборов	Любые конфигурации, для которых нет более эффективных методов. Дополнение (t; k, n) до (t; k+p, n), где p>0	Жадный	$O(n^{2t}\log^2(n^tC_t^k)/n^{t-1}C_t^k + n^{2t}$ $\log(n^tC_t^k)C_t^k)$ Память: $O(n^{t+1}\log(n^tC_t^k) + n^tC_{t-1}^{k-1}$	-	+	+

Жадные алгоритмы универсальны, поэтому инструментов построения покрывающих они используются в большинстве известных наборов [49]. Они обладают следующими

Таблица 9. Алгоритмы построения неоднородных наборов.

Алгоритм	Область применения	Класс алгоритма	Временная сложность и затраты по памяти	M	E	S
Выбрасывание лишних значений	Необходимы дальнейшие исследования.	Редуцирующий	Зависит от размеров исходного набора	-	+	+
Двойная проекция	(2; n+1+t, $(n - t^{n+1}, s^t)$, n – степень простого числа, $1 \le t \le n$ и $1 \le s \le n - t$	Редуцирующий	O(N(k+1)) Память:O(Nk)	-	+	-
Комбинирование блоков	(2; k, n_1 ... n_k) и $k > \max(n_1 \ldots n_k)$ Необходимы дальнейшие исследования.	Рекурсивный	$O(n^2 + k \log_2 k)$, Память: $O(n^2(n+1) + k(\lceil \log_{n+1} k \rceil (n^2 - 1) + +1))$, где $n \ge d$ и $n = p^m$, p – простое число, $k \ge 1$	-	-	-
Неоднородный, рекурсивный, t=2	(2; $l_1 + \ldots + l_k$, $m_{1,1}, \ldots, m_{1,l1}, \ldots, m_{k,1}, \ldots, m_{k,lk}$), из (2; k, $v_1, \ldots, v_k$) с профилем (d_1, ..., d_k), и (2; l_i, $m_{i,1}, \ldots, m_{i,li}$) с профилем ($f_{i,1}, \ldots, f_{i,li}$), $i \in [1, k]$ и $m_{ij} \le v_i$, $j \in [1, l_i]$	Рекурсивный	$(N + M)L + \sum_{i=1}^{k}(M_i l_i)$ Память:$(N + M)L + \max_{i=1}^{k}(M_i l_i)$, $M = \max_{i=1}^{k}(M_i - d_i)$, $L = \sum_{i=1}^{k} l_i$, где M_i – число строк в исходных наборах	-	-	-
IPO	Дополнение (t; k, $n_1 \ldots n_k$) до (t; k + p, $m_1, \ldots m_{k+p}$), где $n_i \le m_i$, $i \in [1, k], \exists j : j \in [1, k]$, $n_j < m_j$ и/или p>0	Жадный	$O(tk^6 d^t (d^t + k + d^t k))$ Память: $O(d^t(k \log k + tC_{t-1}^{k-1}))$	-	+	+
Оптимизация исходного набора	Уменьшение строк в исходном наборе (t; k, $n_1 \ldots n_k$) размера N	Редуцирующий Оптимизационный	$O(N_e(M(C_t^k tN + Nk + N - 1) + +6k))$, M – число перезапусков для поиска улучшений. Память: $O(Nk+d^t)$	-	+	-

важными для практического применения особенностями:

- возможность отсеивать строки наборов по ограничениям на возможные комбинации;
- возможность применения их для построения наборов переменной глубины;
- возможность расширять уже существующий тестовый набор.

Главным недостатком жадных алгоритмов является обратная зависимость между близостью результирующего набора к минимальному и временем работы алгоритма. Необходимо дальнейшее изучение области применения жадных алгоритмов для оптимизации.

Необходимо также дальнейшее изучение области применения алгоритмов выбрасывания лишних значений и комбинирования блоков, а также обобщение их на целые классы конфигураций. Методы дают очень неплохие результаты, однако реализация метода комбинирования блоков в инструменте TConfig [5] имеет узкую область применения, а реализация метода выбрасывания лишних значений в виде инструмента невозможна без предварительной формализации и алгоритмизации. Некоторые исследования этой проблемы проведены в работах [16, 19].

Можно существенно расширить класс конфигураций покрывающих наборов, для которых существует хорошее решение, за счет комбинирования методов. Для выявления эффективной комбинации методов, необходимо анализировать исходную конфигурацию набора. В работе [16] предложено решение, основанное на комбинировании прямых, рекурсивных и оптимизационных алгоритмов – пакет CTS. Перспективным направлением исследований выглядит дополнение использующейся в CTS техники комбинирования алгоритмов.

Принципы комбинирования методов на основе анализа исходных конфигураций могут выглядеть следующим образом.

- Использовать прямые алгоритмы в сочетании с рекурсивными для тех конфигураций,

Таблица 10. Методы, легко модифицируемые для построения наборов переменной глубины.

Алгоритм	Область применения	Класс алгоритма	Временная сложность и затраты по памяти	M	E	S
Жадный, Построчный	Любые конфигурации, для которых нет более эффективных методов. Расширение исходного набора	Жадный	$O(k(f(PR_1)+f(PR_2))d(f(VR_1)+f(VR_2)))$, $f(PR_1)$, $f(PR_2)$, $f(VR_1)$, $f(VR_2)$ – функции времени работы эвристик. Память: $O(d^t(k\log k + tC_{t-1}^{k-1}))$	-	+	+
Поиск экстремума	Теоретические исследования, поиск минимального набора	Жадный, задача оптимизации	Экспоненциальные время и память	-	-	-
Поиск с ограничениями	Теоретические исследования, поиск минимального набора	Жадный, задача оптимизации	$O(N_e k N_{max} d(k+\log T+ +O(dkN_{max}(T+k^2)))$ Память: $O(kN_{max}T)$, T - число хранимых наборов для реализации tabu	-	-	+
Симуляция отжига	Теоретические исследования, поиск минимального набора	Жадный, задача оптимизации	$O(N_e k + O(k^3 N_{max}))$, Память: $O(kN_{max})$	-	-	-
Великий потоп	Теоретические исследования, поиск минимального набора	Жадный, задача оптимизации	$O(N_e k + O(k^3 N_{max}))$, Память: $O(kN_{max})$	-	-	-
Генетические алгоритмы	Теоретические исследования, поиск минимального набора	Генетический	$O(N_e M k^2 (N_{max}+d^2))$, Память: $O(MkN_{max})$, M – размер популяции	-	-	-
Поиск с возвратом	Теоретические исследования, поиск минимального набора, расширение исходного набора	Поиск с возвратом	$O(MNkf(CH)f(SH))$, $M \geq k(N-N_{init})$ – число перебранных наборов, $f(CH)$, $f(SH)$ – функции времени работы эвристик. Память: $O(Nk^2(N-N_{init})+d^tC_k^t)$, N_{init} – размер исходного набора.	-	+	+

для которых это возможно, поскольку результирующие наборы будут построены за малое время и окажутся близкими к минимальным. Примеры конфигураций: (2; k, n), и n раскладывается в произведение степеней простых чисел $n = p_1^{w1}p_2^{w2}\ldots p_r^{wr}$ таких, что $k \leq (p_i^{wi}+1)p_i^{wi}+1$, где p – простое. Строим r наборов A_i из CA(2; k, p_i^{wi}) и перемножаем их рекурсивным алгоритмом умножения. Если $k \leq p_i^{wi}+1$, то минимальный A_i получается построением с помощью поля Галуа. Если $p_i^{wi}+1 < k \leq (p_i^{wi}+1)p_i^{wi}+1$, то строим набор B_i из CA(2; $\lceil k/p^w \rceil - 1$, p_i^{wi}) с помощью поля Галуа, а затем строим A_i из B_i рекурсивным методом для однородных наборов мощности 2 (см. пункт 3.1.4).

- Рекурсивные и жадные алгоритмы лучше не комбинировать, поскольку, вероятнее всего, проще будет получить меньшего размера набор для такой конфигурации другими методами. Иными словами, пример конфигурации (2; k, n) далеко не всегда хорошо обобщается, поскольку, хотя любое n раскладывается в произведение степеней простых чисел $n = p_1^{w1}p_2^{w2}\ldots p_r^{wr}$, для $k > (p_i^{wi}+1)p_i^{wi}+1$, а также для многих $k \leq (p_i^{wi}+1)p_i^{wi}+1$, в общем случае набор A_i из CA(t; k, p_i^{wi}) быстрее строить жадным алгоритмом, который даст не минимальный набор. Перемножение хотя бы с одним не минимальным набором может дать набор, существенно больший, чем набор из CA(2; k, n), построенный с нуля, например, жадным

алгоритмом [37].

- Дополнение построенных прямыми методами (вместе с рекурсивными) минимальных (или близких к минимальным) наборов с конфигурацией для меньшего числа параметров и значений параметров. Например, эффективно будет взять минимальный набор из CA(2; k, n) и дополнить его с помощью алгоритма IPO [13] до набора из CA(2; $k+m$, 2n), где m – небольшое число добавляемых параметров. Начиная с какого-то m построение будет эффективней делать с помощью жадного алгоритма построчного добавления, например, используя эвристику DDA [46].

- Использование алгоритма выбрасывания лишних значений из построенных прямыми (вместе с рекурсивными) методами минимальных наборов с конфигурацией для большего числа параметров и их значений. Пусть у нас есть минимальный набор из CA(2; k, n). Из него методом выбрасывания лишних значений можно эффективно получить набор из CA(2; k, n_1, ... , n_k), где $n_i \le n - m$, где $m \ge 0$ – небольшое число.

- Если размер набора, построенного с помощью быстрых методов – многократной рекурсии или жадного алгоритма с быстро работающей эвристикой – слишком велик, то его можно попробовать уменьшить с помощью метода оптимизации исходного набора [28] или оптимизационных методов (например, симуляции отжига). Для некоторых случаев может оказаться быстрее сгенерировать большой набор, а потом потратить какое-то время на его уменьшение.

Обзор показал, что ни один из распространённых инструментов, строящих покрывающие наборы, не использует в полной мере все известные методы для оптимального построения наборов близких к минимальным. В результате, во многих частных случаях известные инструменты дают наборы, далекие от минимальных, или же строят их дольше, с большими затратами памяти, чем это делает алгоритм, использующий эвристику для данного частного случая. Пакет CTS использует комбинированный подход, но это решение может быть существенно улучшено.

Возникает необходимость в создании нового инструмента, который бы анализировал исходную конфигурацию покрывающего набора и выбирал близкий к оптимальному алгоритм для этой конфигурации. Инструмент должен комбинировать алгоритмы для достижения лучших результатов.

4. ЗАКЛЮЧЕНИЕ

В данной работе проведён анализ областей применения различных техник и методов построения покрывающих наборов. Для рассматриваемых методов были выявлены их достоинства и недостатки, а также были оценены временная сложность алгоритмов и необходимый объем памяти. Были описаны прямые, рекурсивные, оптимизационные, генетические алгоритмы и алгоритмы поиска с возвратом. Также были описаны эвристики, позволяющие сократить покрывающие наборы, и области применимости этих эвристик.

В результате проведённого анализа методов построения покрывающих наборов был сделан вывод о том, что анализ конфигурации набора и комбинирование методов построения позволят расширить классы конфигураций покрывающих наборов, для которых существуют эффективные решения. Ни один из существующих инструментов (кроме CTS [16]) не анализирует конфигурацию набора и не использует достоинства многих известных алгоритмов и эвристик для эффективного построения покрывающих наборов. Решение, предложенное в пакете CTS, использует далеко не все эффективные комбинации алгоритмов.

Одним из возможных перспективных направлений развития является анализ исходной конфигурации покрывающего набора и выбор близкого к оптимальному алгоритма или комбинации алгоритмов для этой конфигурации. В работе были сформулированы принципы комбинирования различных алгоритмов для построения наименьшего покрывающего набора

с наименьшими затратами по времени и по памяти.

СПИСОК ЛИТЕРАТУРЫ

1. *Зеленов С.В., Зеленова С.А.* Автоматическая генерация позитивных и негативных тестов для тестирования фазы синтаксического анализа. Труды Института Системного Программирования РАН. 8(1):41–58. 2004.

2. *Hoffman D., Sobotkiewicz L., Wang Hong-Yi, Strooper P., Bazdell G., Stevens B.* Test Generation with Context Free Grammars and Covering Arrays. Testing: Academic and Industrial Conference – Practice and Research Techniques, Windsor. UK. 2009. P. 83–87

3. *Ammann P., Offutt J.* Using Formal Methods to Derive Test Frames in Category-Partition Testing. Safety, Reliability, Fault Tolerance, Concurrency, and Real Time Security: Proc. 9-th Ann. Conf. Computer Assurance (COMPASS '94). 1994. P. 69–79.

4. *Cohen D.M., Dalal S.R., Fredman M.L., Patton G.C.* The AETG System: an approach to testing based on combinatorial design. IEEE Transactions on Software Engineering. 23(7): 437-444. 1996.

5. *Williams A.W.* Determination of Test Configurations for Pairwise Interaction Coverage. Proc. of 13-th Intl. Conf. on Testing Communicating Systems (TestCom). 2000. P. 59–74.

6. *Williams W., Probert R.L.* A measure for component interaction test coverage. Proc. ACS/IEEE Intl. Conf. on Computer Systems and Applications. 2001. P. 301–311.

7. *Bryce R., Colbourn C.J.* One-Test-at-a-Time Heuristic Search for Interaction Test Suites. Proc. of Genetic and Evolutionary Computation Conference (GECCO). Search-based Software Engineering track (SBSE). London, England, July 2007. P. 1082–1089.

8. *Calvagna A, Gargantini A.* Combining Satisfiability Solving and Heuristics to Constrained Combinatorial Interaction Testing. Proc. of 3-rd Intl. Conf. on Tests and Proofs, Zurich, Switzerland. 2009. P. 27–42.

9. Документация по сервису WM Keeper Light http://webmoney.ru/rus/about/demo/light/index.shtml

10. *Patton G.C.* DAT (Defect Analysis Team) 1986-1990 Overview. Internal Bellcore Technical Memo. 1991.

11. *Cohen D.M., Dalal S.R., Parelius J., Patton G.C.* The Combinatorial Design Approach to Automatic Test Generation. IEEE Software, 13(5): 83-87. 1996.

12. *Seroussi G., Bshouty N.H.* Vector sets for exhaustive testing of logic circuits. IEEE Trans. Information Theory, 34(3): 513-522. 1988.

13. *Lei Y., Tai K.C.* In-parameter order: A test generation strategy for pairwise testing. In Proc. 3rd IEEE High Assurance System Engineering Symposium. 1998. P. 254–161.

14. *Godbole A.P., Skipper D.E., Sunley R.A.* t-Covering arrays: upper bounds and Poisson approximations. Combinatorics, Probability and Computing, 5: 105-118. 1996.

15. *Martirosyan S., Van Trung T.* On t-covering arrays. Designs, Codes and Cryptography 32(1-3): 323-339. 2004.

16. *Hartman A., Raskin L.* Problems and algorithms for covering arrays. Discrete Math. 284(1-3): 149-156. 2004.

17. *Colbourn C.J.* Combinatorial aspects of covering arrays. Le Matematiche (Catania), 58: 121-167. 2004.

18. *Grindal M., Offutt A.J., Andler S.F.* Combination testing strategies: A survey. Software Testing, Verification, and Reliability, 15(3): 167-199. 2005.

19. *Hartman A.* Software and Hardware Testing Using Combinatorial Covering Suites. Proc. of Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications. 2005. P. 266–327.

20. *Colbourn C.J., Keri G., Rivas Soriano P.P., Schlage-Puchta J.-C.* Covering and radius-covering arrays: Constructions and classification. Discrete Appl. Math., 158(11): 1158-1180. 2010.

21. *Colbourn C.J.* Strength two covering arrays: Existence tables and projection. Discrete Mathematics 308(5-6): 772-786. 2008.

22. *Forbes M., Lawrence J., Lei Y., Kacker R.N., Kuhn D.R.* Refining the in-parameter-order strategy for constructing covering arrays. J. Res. Nat. Inst. Stand. Tech., 113(5): 287-297. 2008.

23. *Константинов А.В.* Автоматизация построения тестов с использованием комбинаторных методов. Дипломная работа, Московский Государственный Университет им. М.В. Ломоносова. 2007.

24. *Cohen M.B., Dwyer M.B., Shi J.* Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach. IEEE Trans. Softw. Eng. 34(5): 633-650. 2008.

25. *Cohen M.B., Dwyer M.B., Shi J.* Exploiting Constraint Solving History to Construct Interaction Test Suites. Testing: Academic and Industrial Conference Practice and Research Techniques – MUTATION. 2007. P. 121–132.

26. *Czerwonka J.* Pairwise testing in real world. In: 24-th Pacific Northwest Software Quality Conference. 2006.

27. *Yan J, Zhang J.* A backtracking search tool for constructing combinatorial test suites. J. Syst. Softw., 81(10): 1681–1693. 2008.

28. *Nayeri P., Colbourn C.J., Konjevod G.* Randomized postoptimization of covering arrays. Springer, LNCS 5874: 408–419. 2009.

29. *Edelman A.* The mathematics of the Pentium division bug. SIAM Review 39(1): 54–67. 1997.

30. *Greene C.* Sperner families and partitions of a partially ordered set. In Combinatorics. M. Hall Jr. J. van Lint, eds. Dordrecht, Holland. 1975. P. 277–290.

31. *Лидл Р., Нидеррайтер Г.* Конечные поля. В 2-х т. М.: Мир. 1988.

32. *Chateauneuf M., Kreher D.* On the state of strength-three covering arrays. Journal of Combinatorial Designs, 10(4): 217–238. 2002.

33. *Colbourn J, Martirosyan S.S., Mullen G.L., Shasha D., Yucas J.L., Sherwood G.B.* Products of Mixed Covering Arrays of Strength Two. J. Combin. Design, 14(2): 124–138. 2006.

34. *Meagher K., Stevens B.* Group construction of covering arrays. J. Combin. Design, 13(1): 70–77. 2005.

35. *Yin J.* Constructions of difference covering arrays. J. Combinatorial Theory (A) 104(2): 327–339. 2003.

36. *Харари Ф.* Теория графов. М.: Мир. 1973.

37. Таблицы наименьших известных покрывающих наборов http://www.public.asu.edu/ ccolbou/src/tabby/catable.html

38. *Sloane N.* Covering arrays and intersecting codes. Journal of Combinatorial Designs, 1(1): 51–63. 1993.

39. *Cohen M.B., Colbourn C.J., Ling A.C.H.* Constructing Strength Three Covering Arrays with Augmented Annealing. Discrete Mathematics 308(13): 2709–2722. 2008.

40. *Colbourn C.J., Martirosyan S.S., Van Trung T., Walker II R.A.* Roux-type Constructions for Covering Arrays of Strengths Three and Four. Designs, Codes and Cryptography, 41(1): 33–57. 2006.

41. *Stevens B., Mendelsohn E.* New recursive methods for transversal covers. Journal of Combinatorial Designs, 7(3): 185–203. 1999.

42. *Stevens B., Ling A., Mendelsohn E.* A direct construction of transversal covers using group divisible designs. Ars Combin. 63: 145–159. 2002.

43. *Кормен Т.Х., Лейзерсон Х.В., Ривест Р.Л., Штайн К.* Алгоритмы: построение и анализ. М.: Вильямс. 2006. Глава 16. Жадные алгоритмы. С. 1296.

44. *Calvaga A., Gargantini A.* IPO-s: incremental generation of combinatorial interaction test data based on symmetries of covering arrays. Proc. of IEEE Int. Conf. on software testing verification and validation workshops, Denver, Colorado, USA. 2009.

45. *Cohen M., Dalal S.R., Fredman M.L., Patton G.C.* The Automatic Efficient Test Generator (AETG) System. Proc. of 5-th Intl. Symposium on Software Reliability Engineering, 6-9 Nov, 1994. P. 303–309.

46. *Colbourn C.J., Cohen M.B., Turban R.C.* A deterministic density algorithm for pairwise interaction coverage. Proc. of the IASTED Intl. Conference on Software Engineering, February, 2004. P. 242–252.

47. *Tung Y.-W., Aldiwan W.S.* Automating test case generation for the new generation mission software system. In Proc. IEEE Aerospace Conf., 2000, P. 431–437.

48. *Jenkins B.* Tool for pairwise testing. http://burtleburtle.net/bob/math/jenny.html. 2005.

49. Pairwise Testing, Combinatorial Test Case Generation. http://www.pairwise.org/tools.asp

50. *Nurmela K.* Upper bounds for covering arrays by tabu search. Discrete Applied Math., 138(1–2): 143–152. 2004.

51. *Stardom J.* Metaheuristic and the search for covering and packing arrays. Master's thesis, Simon Fraser University. 2001.

52. *Stevens B.* Transversal Covers and Packings, Ph. D. Thesis, Mathematics, University of Toronto. 1998.

53. *Cohen M.B., Colbourn C.J., Ling A.C.H.* Augmenting Simulated Annealing to Build Interaction Test Suites. In Proc Intl. Symposium Software Requirements Engineering, (ISSRE 2003). P. 394–405.

54. *Dueck G.* New Optimization Heuristic – The Great Deluge Algorithm and the Record-To-Record Travel. Journal of Computational Physics, 104: 86–92. 1993.

55. *Dueck G., Scheuer T.* Threshold Accepting: A general purpose optimization algorithm appearing superior to simulating annealing. Journal of Computational Physics, 90: 161–175. 1990.

56. *Rodriguez-Tello E., Torres-Jimenez J.* Memetic Algorithms for Constructing Binary Covering Arrays of Strength Three. Proc. of Artificial Evolution. 2009. P. 86–97.