О применении некоторых эвристик при исследовании задачи вершинной минимизации недетерминированных конечных автоматов методом ветвей и границ. Часть 1

М. Э. Абрамян, Б. Ф. Мельников

***Аннотация*–В работе рассматривается задача вершинной минимизации недетерминированных конечных автоматов. Один из способов ее решения состоит в анализе подмножества множества состояний двух канонических автоматов, построенных на основе исходного недетерминированного конечного автомата. При этом наиболее сложным этапом решения является нахождение минимального покрытия вспомогательной логической матрицы специальными подмножествами ее элементов со значением 1 (true). При выполнении некоторых дополнительных условий по найденному минимальному покрытию можно построить конечный автомат, эквивалентный исходному и имеющий минимальное количество вершин.**

Описан базовый алгоритм нахождения минимального покрытия, основанный на методе ветвей и границ, и дополнительные эвристики, которые можно комбинировать с базовым алгоритмом. Все рассматриваемые алгоритмы являются итерационными anytime-алгоритмами, позволяющими в любой момент времени получить лучшее на данном шаге псевдооптимальное решение. Алгоритмы реализованы на языке C# 6.0 для платформы .NET Framework. Приведены результаты численных экспериментов, демонстрирующие сравнительную эффективность описанных алгоритмов.

***Ключевые слова*–недетерминированные конечные автоматы, минимизация, метод ветвей и границ, эвристический алгоритм, программная реализация.**

I. Введение

Настоящая работа продолжает исследование алгоритма решения задачи вершинной минимизации недетерминированных конечных автоматов, основанного на методе ветвей и границ [1, 2]. Данная задача была поставлена еще в 1960-е годы, упоминание о ней имеется в [3]. В 1993 г. было доказано, что она является NP-трудной [4], поэтому для ее решения целесообразно использовать эвристические алгоритмы, обеспечивающие получение за приемлемое время работы результатов, близких к оптимальным. Среди таких алгоритмов можно выделить итерационные anytime-алгоритмы [5, 6], работающие в режиме реального времени и позволяющие в любой момент получить лучшее (на данном шаге) решение. Предметом рассмотрения данной работы, как и работ [1, 2], является алгоритм указанного типа.

Статья получена 24 июля 2020.

Михаил Эдуардович Абрамян, Южный федеральный университет (email: m-abramyan@yandex.ru).

Борис Феликсович Мельников, Университет МГУ – ППИ в Шэньчжэне (email: bf-melnikov@yandex.ru).

II. Постановка задачи в матричных терминах

Начнем с постановки задачи, сразу сформулировав ее в матричных терминах. Решение данной матричной задачи является основным этапом алгоритма вершинной минимизации, основанного на результатах работ [7–9]. На этом этапе анализируется специальное отношение #, связывающее подмножества множеств состояний X и Y двух канонических автоматов, построенных на базе исходного недетерминированного конечного автомата (см. также [1, п. 2–3]).

Рассматривается матрица A с логическими значениями 1 (true) и 0 (false), которая представляет собой матрицу, задающую отношение #; при этом множество X соответствует строкам матрица A, а множество Y – ее столбцам. В силу свойств отношения # матрица A не содержит нулевых и совпадающих строк и столбцов.

Будем называть *гридом* набор строк и столбцов данной матрицы, на пересечении которых содержатся только единицы. Подчеркнем, что грид может включать несмежные строки и/или столбцы исходной матрицы. Грид называется *полным*, если его нельзя расширить путем добавления новой строки или столбца. Если совокупность единичных элементов, соответствующих некоторому набору полных гридов, включает все единичные элементы исходной матрицы, то будем называть такой набор гридов *покрытием* матрицы, а число полных гридов, входящих в этот набор, – *размером* покрытия.

Задача заключается в нахождении покрытия исходной матрицы, имеющего минимальный размер. При выполнении некоторых дополнительных условий [7–9] по найденному минимальному покрытию можно построить конечный автомат, эквивалентный исходному и имеющий минимальное количество вершин.

III. Краткое описание базового алгоритма

Алгоритм решения задачи основан на методе ветвей и

границ (МВГ; см., например, [5]) и реализован на языке C# 6.0 для платформы .NET Framework.

Подробное описание алгоритма, которое сопровождается обзором соответствующих классов C#, приведено в [1, 2]. В данном пункте приводится его краткое описание, необходимое для того, чтобы понять сущность дополнительных эвристик, рассматриваемых далее.

Первой составляющей базового алгоритма является метод MakeGridRnd, предназначенный для генерации полного грида исходной матрицы, содержащего выбранные строку и столбец матрицы (в соответствии с определением грида на их пересечении должен находиться единичный элемент). После включения соответствующей строки и столбца в конструируемый грид запускается цикл, в котором делается попытка расширить грид путем добавления новой строки или столбца. Поиск подходящей строки начинается со строки, имеющий случайно выбранный номер, после чего все строки перебираются циклически, пока не будет обнаружена требуемая строка или не будут проанализированы все строки. Столбцы матрицы обрабатываются аналогично; при этом обработка строк и столбцов чередуется. Таким образом, реализованный алгоритм поиска полного грида определяется тремя исходными параметрами: номерами начальной строки и столбца, включаемых в грид, и датчиком случайных чисел (объектом типа Random).

Основной частью базового алгоритма является создание и обработка последовательности *подзадач*, каждая из которых включает набор Yes полных гридов для исходной матрицы (который, возможно, еще не является покрытием матрицы). Одновременно с построением последовательности подзадач subtasks строится и коллекция полных гридов grids, из которой берутся элементы для построения новых подзадач на основе имеющихся.

В соответствии с основной идеей МВГ с каждой подзадачей связывается не только набор полных гридов Yes, но и набор No, содержащий полные гриды, которые *не могут* включаться в эту подзадачу и в подзадачи, построенные на ее основе. Учет набора No позволяет избежать дублирования подзадач в процессе их генерации.

Новые подзадачи создаются путем *расщепления* существующей подзадачи на две новые. Это действие представляет собой *основной шаг* базового алгоритма и реализовано в виде метода MainStep. Расщепление выполняется с помощью нового полного грида, который, таким образом, играет роль *разделяющего элемента* МВГ. Разделяющий элемент выбирается из текущей коллекции полных гридов grids таким образом, чтобы он содержал максимальное и при этом ненулевое количество новых (т. е. еще не содержащихся в гридах из набора Yes исходной подзадачи) единичных элементов исходной матрицы. Для первой из двух создаваемых подзадач выбранный грид включается в набор Yes, для второй новой подзадачи этот грид включается в набор No. Если в существующей коллекции гридов grids выбрать разделяющий элемент не удалось, то запускается вспомогательный метод Substep, в котором делается попытка получить разделяющий элемент путем построения нового полного грида (методом MakeGridRnd). В случае, если данная попытка оказывается успешной, выполняется описанное выше расщепление исходной подзадачи на две новых и при этом созданный грид включается в коллекцию grids, если же получить разделяющий элемент не удалось, то исходная задача исключается из рассмотрения.

При успешном расщеплении исходной подзадачи вторая из них (для которой разделяющий элемент был включен в набор No) возвращается в набор subtasks подзадач, ожидающих обработки, а для первой подзадачи (набор Yes которой увеличился за счет добавления разделяющего элемента) проверяется, образует ли ее набор гридов Yes покрытие исходной матрицы. Если образует, то данное покрытие анализируется на минимальность и в случае, если покрытие является минимальным из ранее полученных, то оно рассматривается в качестве очередного псевдооптимального решения исходной задачи. Если набор Yes еще не образует полного покрытия, то подзадача возвращается в набор subtasks для последующей обработки.

Следует подчеркнуть, что даже для исходных матриц небольшого размера оказывается невозможным за разумное время выполнить обработку всех возникающих подзадач. Поэтому важным аспектом МВГ является *способ выбора* очередной подзадачи для обработки, повышающий шансы на обнаружение очередного псевдооптимального решения. Этот способ основан на понятии «веса» подзадачи: в каждый момент времени из набора подзадач выбирается подзадача, имеющая наименьший вес. В качестве весовых составляющих естественно рассмотреть следующие три характеристики подзадачи

1) количество единичных элементов исходной матрицы, еще не покрытых полными гридами из набора Yes данной подзадачи;

2) размер набора Yes подзадачи;

3) размер набора No подзадачи.

Действительно, малая величина характеристики (1) означает, что подзадача может быть быстро достроена до какого-либо решения исходной задачи, малая величина (2) повышает шансы получить решение малого размера, а малая величина (3) означает, что имеется больше вариантов для построение новых подзадач. Численные эксперименты, приведенные в [1], показали, что лучшие результаты достигаются, если в качестве веса использовать характеристику (3) (или, при условии включения в вес всех характеристик, использовать для характеристики (3) больший весовой коэффициент). Поэтому в базовом алгоритме при вычислении веса подзадачи использовался один из двух следующих наборов весовых коэффициентов: (0, 0, 1) и (1, 1, 10).

IV. Тестирование базового алгоритма

При тестировании реализованного алгоритма использовались два метода: StepRun(steps) и TimeRun(seconds). Метод StepRun выполняет указанное в параметре steps количество вызовов метода MainStep. Продолжительность метода TimeRun (и тем самым количество вызовов метода MainStep) определяется параметром seconds следующим образом: работа метода TimeRun прекращает-

ся, если за указанное число секунд не будет найдено ни одного нового псевдооптимального решения.

Базовый алгоритм и его модификации тестировались на двух наборах из 100 матриц. Первый набор содержал матрицы размера 15×25, которые генерировались случайным образом путем добавления к исходной нулевой матрице единичных элементов, соответствующих 20 случайно выбранным гридам. Второй набор содержал матрицы размера 30×40, построенные на основе 35 случайных гридов. Гриды выбирались таким образом, чтобы они не были попарно вложены; кроме того, также проверялось, чтобы построенные матрицы удовлетворяли дополнительным условиям: не содержали нулевых и совпадающих строк и столбцов. Таким образом, если под *размером оптимального решения* понимать минимальный размер покрытия матрицы полными гридами, то для первого набора матриц размер оптимального решения оценивался сверху величиной 20, а для второго набора – величиной 35. Каждая из матриц обрабатывалась с помощью методов StepRun(500), StepRun(5000) и TimeRun(10), т. е. для нахождения псевдооптимального решения либо использовалось фиксированное число вызовов метода MainStep (500 или 5000), либо определялось время (10 секунд), после которого обработка матрицы завершалась, если в течение этого времени не было найдено ни одного нового псевдооптимального решения.

В табл. 1 и 2 приведены результаты численных экспериментов для базового алгоритма. Табл. 1 содержит данные о размере полученных псевдооптимальных решений, а табл. 2 – данные о времени работы алгоритма (в секундах). Для каждого элемента данных указываются три характеристики: среднее значение для обработанных 100 матриц и (в скобках) минимальное и максимальное значение. В качестве основного здесь и далее рассматривался вариант алгоритма с набором весовых коэффициентов (0, 0, 1); кроме того, в данном пункте для сравнения приводятся результаты для варианта, соответствующего набору (1, 1, 10).

Расчеты проводились на компьютере с процессором AMD A10-6700 3,70 GHz.

Таблица 1. Размер наилучшего псевдооптимального решения для различных режимов

	StepRun(500)	StepRun(5000)	TimeRun(10)
15×25	17,38 (14-22)	15,85 (13-19)	15,11 (12-18)
15×25, набор (1, 1, 10)	20,47 (17-25)	19,04 (15-24)	17,93 (14-22)
30×40	58,04 (43-74)	38,92 (30-47)	34,86 (27-46)
30×40, набор (1, 1, 10)	54,63 (44-69)	51,49 (41-66)	49,03 (40-63)

Таблица 2. Время обработки матриц (в секундах)

	StepRun(500)	StepRun(5000)	TimeRun(10)
15×25	0,06 (0,05-0,11)	0,83 (0,53-1,37)	12,35 (10,02-22,25)
15×25, набор (1, 1, 10)	0,10 (0,05-0,20)	1,32 (0,51-2,29)	14,48 (10,02-22,37)
30×40	0,13 (0,11-0,17)	2,04 (1,53-2,50)	18,55 (10,55-29,94)
30×40, набор (1, 1, 10)	0,12 (0,08-0,27)	1,80 (1,11-4,23)	17,97 (10,26-35,54)

Результаты численных экспериментов свидетельствуют о том, что, реализованный алгоритм позволяет быстро определить хорошие псевдооптимальные решения исходной задачи. В частности, для набора весовых коэффициентов (0, 0, 1) в случае матриц размера 15×25 уже в методе StepRun(5000) (0,83 секунды) удается достичь верхней границы оптимального решения (равной 20 гридам) для *всех* 100 матриц, а для матриц размера 30×40 указанное значение (35 гридов) достигается *в среднем* в методе TimeRun(10) (19 секунд), хотя в этом случае имеет место достаточно большой разброс полученных псевдооптимальных значений (от 27 до 46). Результаты также демонстрируют важность удачного выбора весовых коэффициентов для подзадач, особенно для матриц большого размера.

V. Первая модификация базового алгоритма: отсечение подзадач

Базовый вариант алгоритма решения основной задачи, описанный в п. II, допускает ряд модификаций. Для исследования эффективности подобных модификаций можно было порождать на основе класса Task, содержащего реализацию базового алгоритма, производные классы. Однако такой способ затрудняет анализ вариантов, основанных на *комбинациях* модификаций. Поэтому был использован другой подход, при котором все возможные модификации алгоритма вносятся в методы исходного класса Task, а в его конструкторе предусматривается параметр algOptions, определяющий комбинацию тех модификаций (дополнительных эвристик), которые должны использоваться при текущем запуске. Параметр algOptions имеет перечислимый тип, позволяющий задавать набор битовых флагов.

В базовом варианте алгоритма отсечение подзадачи проводилось в единственном случае: если в методе Substep за указанное количество попыток не удавалось сформировать новый грид, который можно было бы добавить в текущую подзадачу. Следует заметить, что для матриц большого размера подобная ситуация возникает сравнительно редко. В то же время часто возникает ситуация, когда в процессе построения подзадачи количество добавленных в нее гридов (т. е. размер набора Yes)

становится равным или большим уже найденного размера псевдооптимального решения. В данной ситуации анализировать эту подзадачу (и, в частности, порождать на ее основе новые подзадачи) не имеет смысла. Более того, можно сразу прервать обработку подзадачи, извлеченной из набора subtasks, если она содержит OptSize – 1 гридов, где OptSize – размер уже найденного псевдооптимального решения. Отсечение подзадачи следует выполнить и в том случае, когда после добавления к ней очередного грида размер набора Yes становится равным OptSize – 1 и при этом подзадача еще не является решением исходной задачи, т. е. не определяет покрытие матрицы. Таким образом, действия по отсечению следует выполнять не только в методе Substep, но и в начале метода MainStep (при извлечении очередной подзадачи из набора subtasks), а также в конце этого метода при выполнении завершающих действий после добавления к подзадаче нового грида. Для описанной эвристики будем использовать обозначение CutOff.

Естественно ожидать, что за счет применения эвристики CutOff будет обработано большее число подзадач, и это повысит шансы на нахождение лучших псевдооптимальных решений. Действительно, для режима StepRun наблюдается ускорение алгоритма (табл. 3), а для режима TimeRun – увеличение количества обработанных подзадач (табл. 4).

В то же время численные эксперименты показывают, что добавление к базовому алгоритму дополнительной эвристики отсечения не приводит к его улучшению (табл. 5).

Данное обстоятельство может быть объяснено тем, что в процессе обработки «неоптимальных» подзадач (которые были бы досрочно отсечены при использовании соответствующей эвристики) набор grids пополняется новыми полными гридами, наличие которых повышает шансы построения псевдооптимальных решения при обработке последующих подзадач. Это предположение подтверждается информацией об итоговом размере набора grids для различных вариантов алгоритма (табл. 6).

Таблица 3. Время обработки матриц (в секундах)

	StepRun(500)	StepRun(5000)
15 × 25	0,06 (0,05-0,11)	0,83 (0,53-1,37)
15 × 25, CutOff	0,04 (0,03-0,09)	0,30 (0,25-0,39)
30 × 40	0,13 (0,11-0,17)	2,04 (1,53-2,50)
30 × 40, CutOff	0,12 (0,09-0,16)	1,09 (0,83-1,39)

Таблица 4. Количество завершенных/отсеченных подзадач

	TimeRun(10)
15 × 25	7516 (1281-25559) / 2844 (0-7905)
15 × 25, CutOff	8 (5-12) / 58804 (28364-122771)
30 × 40	2513 (1106-4352) / 49 (0-391)
30 × 40, CutOff	13 (8-20) / 5688 (2706-23972)

Таблица 5. Размер наилучшего псевдооптимального решения для различных режимов с применением и без применения эвристики CutOff

	StepRun(500)	StepRun(5000)	TimeRun(10)
15 × 25	17,38 (14-22)	15,85 (13-19)	15,11 (12-18)
15 × 25, CutOff	18,44 (15-23)	17,32 (13-23)	15,88 (13-20)
30 × 40	58,04 (43-74)	38,92 (30-47)	34,86 (27-46)
30 × 40, CutOff	58,57 (45-74)	42,66 (30-59)	41,34 (30-53)

Таблица 6. Размер набора grids на завершающем шаге алгоритма для различных режимов с применением и без применения эвристики CutOff

	StepRun(500)	StepRun(5000)	TimeRun(10)
15 × 25	46 (40-51)	65 (51-75)	84 (58-109)
15 × 25, CutOff	25 (19-30)	28 (23-34)	39 (31-51)
30 × 40	67 (56-80)	117 (102-139)	150 (133-167)
30 × 40, CutOff	60 (47-75)	61 (47-78)	64 (49-79)

VI. Вторая модификация базового алгоритма: генерация начального набора полных гридов

Весьма естественной дополнительной эвристикой является *предварительная подготовка* набора полных гридов grids, используемого при построении подзадач. Понятно, что чем более представительным является данный набор, тем выше шансы построения решения, близкого к оптимальному.

Для генерации начального набора полных гридов реа-

лизован алгоритм, основанный на ранее описанном методе MakeGridRnd. Данный метод вызывается указанное число раз для каждого элемента исходной матрицы со значением 1, и полученные в результате полные гриды добавляются в набор grids. Для того чтобы исключить дублирование созданных гридов они сохраняются в виде упорядоченного множества (структуры SortedSet). Данная эвристика будет обозначаться в виде MakeGr(N), где N указывает число вызовов метода MakeGridRnd для каждого единичного элемента исходной матрицы.

В табл. 7 приведены данные о размере созданных начальных наборов полных гридов для различных режимов. Следует заметить, что при наличии сформированного начального набора grids метод Substep в ходе работы алгоритма практически не вызывается.

Применение начального набора полных гридов улучшают полученные псевдооптимальные решения, особенно для матриц большого размера (табл. 8).

При наличии большого набора полных гридов увеличивается время каждого шага и, соответственно уменьшается количество шагов, выполненных за указанное время (из-за необходимости перебора всех гридов для выбора очередного разделяющего элемента). Соответствующая информация приведена в табл. 9 и 10.

Заметим, что при наличии эвристики MakeGr отсечение подзадач, реализуемое эвристикой CutOff, приводит к дополнительному (хотя и незначительному) улучшению среднего псевдооптимального решения. В частности, если для эвристики MakeGr(100) и режима TimeRun(10) в случае матриц размера 15×25 среднее псевдооптимальное решение равно 14,76 (см. табл. 8), то при использовании комбинации эвристик MakeGr(100)+CutOff размер среднего псевдооптимального решения уменьшился до 14,59. Для матриц размера 30×40 при тех же наборах параметров размер среднего псевдооптимального решения 29,97 уменьшился до 29,91.

Таблица 7. Размер начального набора grids для различных режимов

	MakeGr(1)	MakeGr(2)	MakeGr(10)	MakeGr(100)
15 × 25	93 (47-167)	127 (58-281	180 (63-517)	191 (63-566)
30 × 40	473 (227-665)	808 (326-1225)	2165 (515-4373)	4334 (582-13107)

Таблица 8. Размер наилучшего псевдооптимального решения для различных режимов с применением эвристики MakeGr

	StepRun(500)	TimeRun(10)
15 × 25	*17,38 (14-22)*	*15,11 (12-18)*
15 × 25, MakeGr(1)	16,12 (13-19)	15,05 (12-18)
15 × 25, MakeGr(2)	15,81 (13-19)	14,87 (12-18)
15 × 25, MakeGr(10)	15,59 (13-18)	14,77 (12-17)
15 × 25, MakeGr(100)	15,59 (13-18)	14,76 (12-17)
30 × 40	*58,04 (43-74)*	*34,86 (27-46)*
30 × 40, MakeGr(1)	33,78 (25-42)	32,50 (25-39)
30 × 40, MakeGr(2)	31,98 (24-40)	31,05 (24-37)
30 × 40, MakeGr(10)	30,30 (24-37)	29,76 (24-35)
30 × 40, MakeGr(100)	30,21 (24-37)	29,97 (24-37)

Таблица 9. Время обработки матриц (в секундах)

	StepRun(500)
15 × 25	*0,06 (0,05-0,11)*
15 × 25, MakeGr(1)	0,12 (0,08-0,20)
15 × 25, MakeGr(2)	0,16 (0,09-0,42)
15 × 25, MakeGr(10)	0,23 (0,11-0,58)
15 × 25, MakeGr(100)	0,41 (0,19-0,83)
30 × 40	*0,13 (0,11-0,17)*
30 × 40, MakeGr(1)	1,11 (0,41-1,97)
30 × 40, MakeGr(2)	2,06 (0,63-4,15)
30 × 40, MakeGr(10)	5,56 (1,25-13,00)
30 × 40, MakeGr(100)	11,85 (2,28-37,86)

Таблица 10. Количество выполненных шагов

	TimeRun(10)
15 × 25	*53354 (28608-137318)*
15 × 25, MakeGr(1)	38119 (25732-77819)
15 × 25, MakeGr(2)	31642 (17900-74182)
15 × 25, MakeGr(10)	26249 (10736-59881)
15 × 25, MakeGr(100)	26434 (10464-60524)
30 × 40	*35645 (22136-53049)*
30 × 40, MakeGr(1)	6441 (2581-20615)
30 × 40, MakeGr(2)	3705 (1599-14475)
30 × 40, MakeGr(10)	1703 (467-5260)
30 × 40, MakeGr(100)	1161 (176-4774)

VII. Заключение

В работе рассмотрены дополнительные эвристики, которые можно комбинировать с базовым алгоритмом нахождения минимального покрытия матрицы полными гридами. Численные эксперименты показали, что эвристика CutOff, обеспечивающая отсечение подзадач, не приводит к улучшению результатов, если ее использовать изолированно, в то время как эвристика MakeGr, связанная с генерацией начального набора полных гридов, существенно улучшает результаты. Кроме того, имеет место улучшение результатов, если использовать комбинацию этих эвристик.

В следующей части будут рассмотрены другие эвристики и обсуждены способы более точной оценки качества получаемых псевдооптимальных решений.

Библиография

[1] Абрамян М.Э., Мельников Б.Ф. Исследование задачи вершинной минимизации недетерминированных конечных автоматов с помощью метода ветвей и границ // Cloud of Science. 2020. Т. 7, № 2. С. 297–319.

[2] Абрамян М.Э. Об одном подходе к реализации метода ветвей и границ для оптимизационных задач // Информационные подходы в моделировании и управлении: подходы, методы, решения. Сборник научных статей II Всероссийской научной конференции с международным участием. Часть 1. Тольятти, 2019. С. 56–64.

[3] Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Т. 1. – М. : Мир, 1978.

[4] Jiang T., Ravikumar B. Minimal NFA problems are hard // SIAM J. Comput. 1993. Vol. 22. No. 6. P. 1117–1141.

[5] Melnikov B.F. Multiheuristic approach to discrete optimization problems // Cybernetics and Systems Analysis. 2006. Vol. 42. No. 3. P. 335–341.

[6] Melnikov B.F., Melnikova E.A., Pivneva S.V., Dudnikov V.A., Davydova E.V. Geometric and game approaches for some discrete optimization problems // CEUR Workshop Proceedings. 2018. Vol. 2212. P. 312–321.

[7] Melnikov B. Once more on the edge-minimization of nondeterministic finite automata and the connected problems // Fundamenta Informaticae. 2010. Vol. 104. No. 3. P. 267–283.

[8] Мельников Б.Ф. Регулярные языки и недетерминированные конечные автоматы : монография. – М. : РГСУ, 2018.

[9] Melnikov B. The complete finite automaton // International Journal of Open Information Technologies. 2017. Vol. 5. No. 10. P. 9–17.

Михаил Эдуардович АБРАМЯН,
доцент Южного федерального университета, Ростов-на-Дону (http://sfedu.ru/),
email: m-abramyan@yandex.ru,
mathnet.ru: personid=20226,
elibrary.ru: authorid=17911,
scopus.com: authorId=8291167000,
ORCID: orcidID=0000-0002-2802-6144.

Борис Феликсович МЕЛЬНИКОВ,
профессор университета МГУ – ППИ в Шэньчжэне (http://szmsubit.ru/),
email: bf-melnikov@yandex.ru,
mathnet.ru: personid=27967,
elibrary.ru: authorid=15715,
scopus.com: authorId=55954040300,
ORCID: orcidID=0000-0002-6765-6800.

On the application of some heuristics in the study of the state minimization problem for nondeterministic finite automata by the branch and bound method. Part 1

M. E. Abramyan, B. F. Melnikov

***Abstract*–We consider the problem of state minimization of nondeterministic finite automata. One of the ways to solve it is to analyze a subset of the set of states of two canonical automata constructed on the basis of the initial nondeterministic finite automaton. In this case, the most difficult stage of the solution is to find the minimum cover of the auxiliary logical matrix by special subsets of its elements with the value 1 (true). If some additional conditions are satisfied, then using the found minimum cover makes it possible to construct a finite automaton that is equivalent to the initial one and has the minimum number of states.**

We describe a basic algorithm for finding the minimum cover based on the branch and bound method and additional heuristics that can be combined with this basic algorithm. All the algorithms under consideration are iterative anytime algorithms that yield the best current-step quasi-optimal solution at any time. The algorithms are implemented in C# 6.0 for the .NET Framework. The results of numerical experiments are presented that demonstrate the comparative efficiency of the described algorithms.

***Keywords*–nondeterministic finite automata, minimization, branch and bound method, heuristic algorithm, implementation.**

REFERENCES

[1] Abramyan M.E., Melnikov B.F. Investigation of the problem of state minimization of nondeterministic finite automata using the branch and bound method // Cloud of Science. 2020. Vol. 7, No. 2. P. 297–319 (in Russian).

[2] Abramyan M.E. On one approach to the implementation of the branch and bound method for optimization problems // Information approaches in modeling and control: approaches, methods, solutions. Collection of scientific papers of the II Russian scientific conference with international participation. Part 1. Togliatti, 2019. P. 56–64 (in Russian).

[3] Aho A., Ullman J. The theory of parsing, translation, and compiling. Vol. 1. – M. : Mir Publ, 1978 (in Russian).

[4] Jiang T., Ravikumar B. Minimal NFA problems are hard // SIAM J. Comput. 1993. Vol. 22. No. 6. P. 1117–1141.

[5] Melnikov B.F. Multiheuristic approach to discrete optimization problems // Cybernetics and Systems Analysis. 2006. Vol. 42. No. 3. P. 335–341.

[6] Melnikov B.F., Melnikova E.A., Pivneva S.V., Dudnikov V.A., Davydova E.V. Geometric and game approaches for some discrete optimization problems // CEUR Workshop Proceedings. 2018. Vol. 2212. P. 312–321.

[7] Melnikov B. Once more on the edge-minimization of nondeterministic finite automata and the connected problems // Fundamenta Informaticae. 2010. Vol. 104. No. 3. P. 267–283.

[8] Melnikov B.F. Regular languages and nondeterministic finite automata: a monograph. – M. : RGSU Publ., 2018 (in Russian).

[9] Melnikov B. The complete finite automaton // International Journal of Open Information Technologies. 2017. Vol. 5. No. 10. P. 9–17.

Mikhail Eduardovich ABRAMYAN,
Associate Professor of Southern Federal University, Rostov-on-Don, Russia (http://sfedu.ru/),
email: m-abramyan@yandex.ru,
mathnet.ru: personid=20226,
elibrary.ru: authorid=17911,
scopus.com: authorId=8291167000,
ORCID: orcidID=0000-0002-2802-6144.

Boris Feliksovich MELNIKOV,
Professor of Shenzhen MSU – BIT University, China (http://szmsubit.ru/),
email: bf-melnikov@yandex.ru,
mathnet.ru: personid=27967,
elibrary.ru: authorid=15715,
scopus.com: authorId=55954040300,
ORCID: orcidID=0000-0002-6765-6800.