УДК 519.8

П.В. Леончик

АЛГОРИТМ ПОКРЫТИЯ РАЗРЕЖЕННЫХ БУЛЕВЫХ МАТРИЦ

Предлагается алгоритм решения задачи о наименьшем покрытии множества, известной в литературе как задача нахождения кратчайшего столбцового покрытия булевой матрицы. Сравнивается эффективность разработанного алгоритма, реализованного в программе Tie, с эффективностью алгоритма программы Espresso и алгоритма GANP. Приводятся результаты экспериментально-статистических испытаний алгоритма на стандартных примерах серий Benchmark, CLR и Stein, а также на псевдослучайных системах булевых функций.

Введение

Решение ряда логико-комбинаторных задач сводится к рассмотрению некоторых конечных множеств и поиску среди них подмножеств с заданными свойствами. Часто из всех удовлетворяющих подмножеств достаточно найти только одно подмножество, которое в каком-то смысле было бы лучшим. Хорошим примером таких задач является задача о наименьшем покрытии множества.

Минимизация булевых функций в классе дизъюнктивных нормальных форм (ДНФ) – одна из логических задач, которая сводится к задаче наименьшего покрытия множества [1]. По сути, алгоритм, представленный в данной статье, является этапом нахождения покрытия булевой матрицы Квайна в алгоритме минимизации системы булевых функций [2]. Алгоритм покрытия булевой матрицы ориентирован на разреженные булевы матрицы, возникающие при решении задачи минимизации булевых функций в классе ДНФ.

1. Задача о наименьшем покрытии множества

Пусть даны некоторое множество $A=\{a_1,a_2,\ldots,a_n\}$ и совокупность его подмножеств $B_1,B_2,\ldots,B_m$, причем $B_1\cup B_2\cup\ldots\cup B_m=A$. Для решения задачи *наименьшего покрытия множества* A требуется среди данных подмножеств выделить такую совокупность $B_{i_1},B_{i_2},\ldots,B_{i_k}$ с минимальным k, чтобы каждый элемент из A попал хотя бы в одно подмножество B_{i_j}, где $j=1,2,\ldots,k$, т. е. $B_{i_1}\cup B_{i_2}\cup\ldots\cup B_{i_k}=A$.

Удобно представить исходные данные в виде булевой матрицы *M*, строки которой соответствуют элементам множества A, а столбцы – подмножествам $B_1,B_2,\ldots,B_m$. В такой булевой матрице в *i*-й строке *j*-го столбца ставится единица, если элемент a_i множества A принадлежит подмножеству B_j, т. е. если $a_i\in B_j$. В этом случае говорят, что *i*-я строка покрывается *j*-м столбцом. При таком представлении данных задача может быть переформулирована в задачу нахождения *наименьшего* (по мощности) *столбцового покрытия булевой матрицы M*.

2. Алгоритм нахождения столбцового покрытия матрицы *M*

Для решения задачи покрытия используется алгоритм, состоящий из двух этапов поиска: первого покрытия и покрытий подматриц. Поиск первого покрытия исходной матрицы *M* производим с помощью *минимаксного* алгоритма [3]. Он представляет собой многошаговый процесс, где на каждом шаге выбирается строка матрицы *M* с наименьшим числом единиц и из покрывающих ее столбцов в решение включается тот, который покрывает наибольшее число непокрытых строк. По завершении работы минимаксного алгоритма некоторые из столбцов матрицы *M*, включенные в покрытие, могут оказаться избыточными, т. е. при их

удалении все строки матрицы *M* останутся покрытыми. После удаления избыточных столбцов получим некоторое решение, которое и будет считаться первым столбцовым покрытием булевой матрицы *M*.

На втором этапе поиска, разбивая матрицу *M* на подматрицы и пытаясь найти меньшее (по мощности) столбцовое покрытие каждой подматрицы в отдельности, улучшим общее столбцовое покрытие матрицы *M*. Очевидно, если X_m – покрытие матрицы *M*, то любая подматрица *W* матрицы *M* будет содержать покрытие $X_w \subseteq X_m$. Улучшая покрытие X_w, улучшаем соответственно и покрытие X_m.

Каждый столбец $k_i \in X_m$ будет являться тем столбцом, на основе которого происходит выбор подматрицы W_i. Если для некоторой подматрицы W_i найдется покрытие $X_{w_i}^*$, которое окажется меньше существующего покрытия X_{w_i}, то замена покрытия X_{w_i} на покрытие $X_{w_i}^*$ приведет к изменению всего покрытия X_m матрицы *M*. Новое покрытие X_m матрицы *M* состоит из нового множества столбцов, для которых построение новых подматриц и нахождение для них новых покрытий, в свою очередь, будет приводить к улучшению покрытия X_m матрицы *M*. Этот процесс будет продолжаться до тех пор, пока будут находиться подматрицы, для которых можно улучшить решение.

3. Алгоритм выбора подматрицы *W* из матрицы *M*

Строчная подматрица *W* матрицы *M* представляет собой подмножество строк $C_w \subset C$, где *C* – множество строк матрицы *M*, со всеми столбцами, покрывающими строки C_w.

Введем понятие *критического* столбца некоторой строки *c*. Столбец $k \in X_m$ является критическим для строки *c*, если после удаления столбца *k* из покрытия X_m строка *c* окажется непокрытой. *Критическим множеством столбцов* некоторой строки *c* называется такое множество столбцов покрытия $X_c \subset X_m$, если после удаления из покрытия X_m всех столбцов X_c строка *c* окажется непокрытой.

Подматрицу *W* выбираем по следующему алгоритму:

Шаг 1. Первым в подматрицу *W* включаем столбец $k \in X_m$, выбираемый произвольно.

Шаг 2. Для столбца k находим множество строк C_1, для которых столбец k является критическим. Такие строки всегда найдутся, так как покрытие X_m безызбыточно.

Шаг 3. Находим множество столбцов K_1, покрывающих все строки множества C_1.

Шаг 4. Находим множество строк C_2, которые покрываются столбцами множества K_1.

Шаг 5. Находим множество столбцов K_2, покрывающих строки множества C_2 и являющихся столбцами покрытия X_m.

Шаг 6. Если некоторый столбец $k^* \in K_2$ является критическим столбцом строки $c \notin C_2$, то столбец k^* удаляется из множества K_2.

Шаг 7. Создаем множество строк $C_3 \subseteq C_2$, которые покрываются всеми столбцами множества K_2.

Множество строк C_3 и будет составлять множество строк C_w подматрицы *W*, а множество всех столбцов, покрывающих множество строк C_w, – множество столбцов K_w. При этом множество столбцов K_2 будет составлять покрытие X_w подматрицы *W*.

Продемонстрируем работу алгоритма на примере матрицы *M*, представленной ниже:

M

	A	***B***	***C****	***D***	***E****	***F***	***G****	***H***	***I****	***J***	***K****	***L***	***M***	***N****	***O****	***P***	***Q****
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
2	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
3	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
6	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
8	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0
10	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
12	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
13	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1
14	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1
15	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0

На *первом* этапе алгоритма нахождения столбцового покрытия матрицы *M* найдем первое покрытие матрицы *M* с помощью минимаксного алгоритма.

Шаг 1. Одной из строк с минимальным числом единиц (1–7, 10, 11, 15) является строка 1. Из покрывающих ее столбцов максимальное число строк покрывает столбец *O*. Включаем столбец *O* в покрытие и удаляем его из матрицы со всеми строками, которые он покрывает.

Шаг 2. Из оставшихся строк минимальное число единиц имеют несколько строк. Одна из них строка 2, ее покрывают два столбца *I* и *K*. Каждый из этих столбцов покрывает по две строки. Включаем в покрытие любой из них (например, столбец *K*).

Шаг 3. Из оставшихся строк находим строку 3 и включаем в покрытие столбец *C*.

Шаг 4. Из оставшихся строк находим строку 5 и включаем в покрытие столбец *E*.

Шаг 5. Из оставшихся строк находим строку 6 и включаем в покрытие столбец *G*.

Шаг 6. Из оставшихся строк находим строку 13 и включаем в покрытие столбец *Q*.

Шаг 7. Для покрытия оставшихся двух непокрытых строк 8 и 12 включаем любые покрывающие их столбцы (например, столбцы *I* и *N*).

По завершении минимаксного алгоритма будет получено покрытие матрицы *M* $X_m = \{O,K,C,E,G,Q,I,N\}$.

На *втором* этапе алгоритма найдем все тупиковые покрытия подматрицы *W*. Алгоритм выбора подматрицы *W* из матрицы *M* продемонстрируем на примере матрицы *M*, столбцы которой, помеченные символом *, являются первым покрытием, найденным с помощью минимаксного алгоритма.

Шаг 1. Включаем в подматрицу *W* столбец Q, который входит в покрытие.

Шаг 2. Для столбца Q находим множество строк C_1, для которых этот столбец является критическим ($C_1 = \{13,14\}$).

Шаг 3. В множество K_1 включаем все столбцы, покрывающие строки множества C_1 ($K_1 = \{D,H,L,M,P,Q\}$).

Шаг 4. Находим множество строк C_2, которые покрываются столбцами из множества K_1 ($C_2 = \{1,4,7,9,10,13,14\}$).

Шаг 5. Находим множество столбцов K_2, покрывающих строки множества C_2 и являющихся столбцами покрытия X_m ($K_2 = \{C,E,G,K,O,Q\}$).

Шаг 6. Удаляем из множества K_2 столбцы C,E,G, так как они являются критическими для строк 3, 5, 6 соответственно, а эти строки не принадлежат множеству C_2 ($K_2 = \{K,O,Q\}$).

Шаг 7. Находим множество строк $C_3 \subseteq C_2$, которые покрываются всеми столбцами множества K_2 ($C_3 = \{1,9,13,14\}$). Строки 1, 9, 13, 14 со всеми покрывающими их столбцами и будут составлять подматрицу *W*, представленную ниже:

W

	D	***F***	***H***	***K****	***L***	***M***	***O****	***P***	***Q****
1	0	0	0	0	0	0	1	1	0
9	0	1	0	1	1	0	0	0	0
13	0	0	1	0	0	0	0	1	1
14	1	0	0	0	1	1	0	0	1

Подматрица *W* содержит покрытие $X_w = \{K, O, Q\}$, которое является частью общего покрытия $X_m = \{O, K, C, E, G, Q, I, N\}$. Решая задачу покрытия для подматрицы *W* путем поиска всех минимальных покрытий, найдем покрытие $X_{new} = \{L, P\}$. Очевидно, что покрытие подматрицы *W* новым покрытием X_{new} улучшает общее покрытие X_m матрицы *M*.

4. Поиск всех минимальных покрытий подматрицы

Следует отметить, что для улучшения решения покрытия подматрицы достаточно найти одно минимальное покрытие, однако часто возникает ситуация, когда для некоторой подматрицы не удается улучшить решение. В таком случае из всех минимальных покрытий подматрицы надо выбрать какое-то одно, которое может оказаться предпочтительнее, чем существующее решение. Каждому из минимальных покрытий подматрицы X_{w_i} присваивается *внешний коэффициент* $\Omega(X_{w_i})$, который равен числу строк множества $C \not\subset C_w$ матрицы *M*, покрываемых столбцами покрытия X_{w_i}, где C_w – множество строк подматрицы *W*. Предпочтение отдается минимальному покрытию X_{w_i} с максимальным коэффициентом Ω.

Алгоритм поиска всех минимальных покрытий подматрицы *W* представляет собой обход дерева покрытий (рис. 1), на каждом шаге которого покрывается еще непокрытая строка. Для сокращения обхода дерева покрытий применяется процедура определения избыточности, что приводит к сокращению времени поиска минимального покрытия. Проиллюстрируем работу алгоритма на более наглядном примере подматрицы *W**:

*W**

	a	***b***	***c***	***d***	***e***	***f***	***g***
1	1	0	1	0	0	0	0
2	1	1	0	0	1	0	0
3	0	1	0	1	0	0	0
4	0	0	1	1	0	1	0
5	0	0	0	0	1	0	1
6	0	0	0	0	0	1	1

В начале алгоритма для покрытия строки 1 включаем в покрытие столбец *a*. Переходим к строке 3 (так как строка 2 уже покрыта столбцом *a*) и для ее покрытия включаем столбец *b*. Переходим к строке 4. Пробуем включить в покрытие столбец *c,* но он не может быть включен, так как при этом столбец *a* окажется избыточным, поскольку покрывает только строки 1 и 2, которые уже покрыты столбцами *c* и *b*. Поэтому, чтобы покрыть строку 4, попытаемся включить в покрытие следующий покрывающий ее столбец *d*. Этот столбец тоже не может быть включен в покрытие, потому что окажется избыточным столбец *b*. Для покрытия строки 4 включаем в покрытие столбец *f*. Осталось покрыть только строку 5, которую покрываем столбцом *e.* Таким образом найдено первое безызбыточное покрытие $\{a, b, f, e\}$. После нахож-

дения безызбыточного покрытия идем в обратном направлении. Последняя покрытая строка 5 была покрыта столбцом *e.* Удаляем столбец *e* из покрытия и покрываем строку 5 столбцом *g*. Следующее найденное покрытие – $\{a,b,f,g\}$.

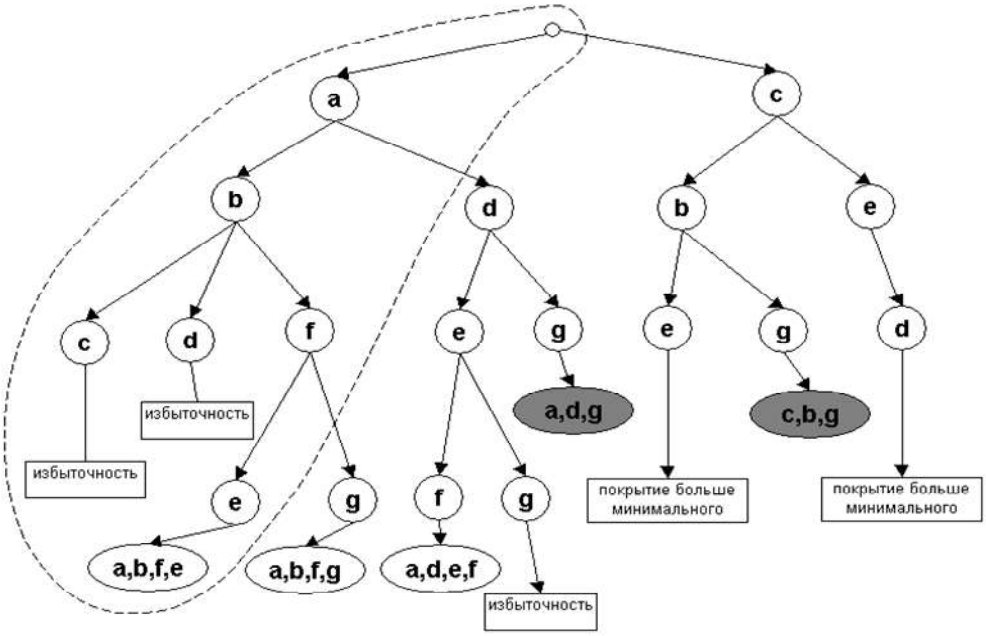

Рис. 1. Дерево перебора покрытий

Описанным способом последовательно находим следующие покрытия матрицы, которые будем получать в таком порядке: $\{a,b,f,e\}$, $\{a,b,f,g\}$, $\{a,d,e,f\}$, $\{a,d,g\}$. Последнее найденное покрытие $\{a,d,g\}$ содержит всего три столбца, поэтому при поиске следующих покрытией нет необходимости выбирать покрытия, содержащие более трех столбцов. Для данной подматрицы находим еще одно покрытие $\{c,b,g\}$, содержащее три столбца. Среди всех покрытий матрицы два покрытия $\{a,d,g\}$, $\{c,b,g\}$ являются минимальными (рис. 1).

5. Процедура определения избыточности

Перед началом обхода дерева покрытий каждому столбцу подматрицы присваивается *коэффициент избыточности* ψ, который равен числу строк, покрываемых столбцом. После каждого включения столбца k_i в покрытие выполняется процедура определения избыточности, которая пересчитывает коэффициент избыточности для столбцов S_{k_i}, покрывающих те же строки, что и столбец k_i. Коэффициент избыточности столбца $k_j \in S_{k_i}$ равен числу строк, которые покрываются столбцом k_j и при этом не покрыты столбцами, включенными в покрытие. Если для некоторого включенного в покрытие столбца k_j коэффициент избыточности равен нулю, то столбец k_i нецелесообразно включать в покрытие, так как оно окажется избыточным. На рис. 2 отображена часть дерева перебора покрытий из рис. 1, обведенная пунктирной линией. На этом дереве показан поиск первого безызбыточного покрытия $\{a,b,f,e\}$ подматрицы *W**, состоящий из нескольких шагов:

Шаг 1. Присваиваем коэффициенту избыточности всех столбцов число 2, равное числу строк, покрываемых каждым столбцом.

Шаг 2. Включив в покрытие столбец a, процедура определения избыточности пересчитывает коэффициет избыточности для столбцов $S_a = \{b,c,e\}$.

Шаг 3. Включаем в покрытие столбец b.

Шаг 4. При включении в покрытие столбца c оказывается, что $\psi(a)=0$. Поэтому удаляем из покрытия столбец c и возвращаемся по дереву на один уровень вверх (к вершине b).

Шаг 5. При включении в покрытие столбца d оказывается, что $\psi(b)=0$. Удаляем из покрытия столбец d.

Шаг 6. Включаем в покрытие столбец f.

Шаг 7. Включив в покрытие столбец e, получаем первое безызбыточное покрытие $\{a,b,f,e\}$.

Процедура определения избыточности позволяет сократить время поиска минимального покрытия подматрицы.

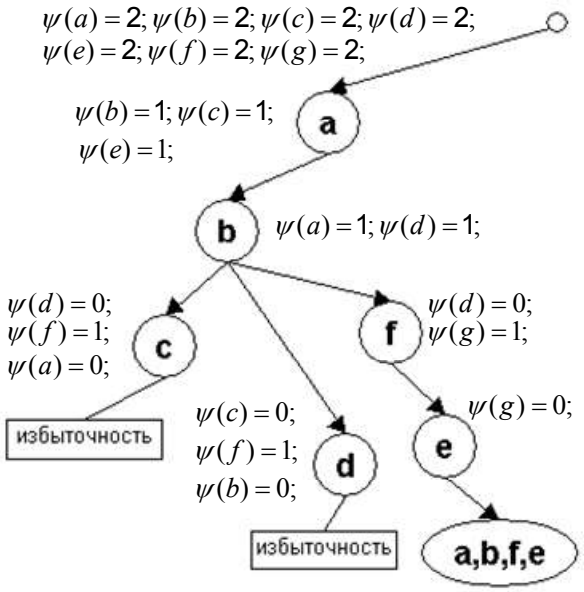

Рис. 2. Процедура определения избыточности

6. Экспериментально-статистические испытания алгоритма

Алгоритм покрытия булевой матрицы, являющийся частью алгоритма минимизации булевых функций в классе ДНФ, реализован в программе Tie [2]. Сравнение эффективности алгоритма покрытия проводилось на персональном компьютере с процессором Intel Pentium 4 с тактовой частотой 1,7 ГГц, 1024 Мб оперативной памяти. Программа Tie реализована на языке Си и протестирована на операционных системах Windows 98/2000/Xp. Объем исполняемого файла около 1 Мб.

Программа Tie была подвергнута многократным испытаниям на известных примерах серии Benchmark [4] и псевдослучайных булевых функциях, полученных с помошью параметрически управляемых генераторов [5], в экспериментах использовались примеры серий CLR и Stein [6].

В табл. 1 представлены стандартные примеры серии Benchmark для систем булевых функций в классе ДНФ. В процессе решения создается булева матрица Квайна $m \times n$, на которой и проверяется эффективность решения задачи покрытия. Для сравнения используется модифицированная программа Espresso-Qm (программа Espresso [7–9] с ключом Dqm), находящая все простые импликанты системы функций, как и программа Tie. В табл. 1 используются следующие обозначения: m – число строк матрицы; n – число столбцов матрицы; p – процентное соотношение числа единиц в матрице к общему числу элементов матрицы; k – число столбцов покрытия; t – время, затраченное на решение задачи покрытия.

Программа Tie, в отличие от программы Espresso, позволяет сохранять результаты минимизации без завершения работы программы [2]. С целью демонстрации эффективности разра-

ботанного алгоритма (по качеству и скорости получаемого покрытия) для примеров приводятся два решения. Например, для матрицы test3, состоящей из 3543 строк и 40 680 столбцов, программой Tie за 52,9 с найдено решение, состоящее из 460 столбцов, а за 7256,4 с найдено решение из 433 столбцов.

Таблица 1

Сравнение эффективности алгоритма покрытия на стандартных примерах серии Benchmark

Пример	*m*	*n*	*p*	Espresso-Qm		Tie	
				k	*t,* с	*k*	*t,* с
bench1	403	4688	0,513	125	2,14	123/121	1,13/3,23
exam	625	4694	1,072	65	4,45	65/63	0,509/1,349
ex1010	1471	25203	0,268	–	–	256/237	7,65/35163,2
ex5	7620	2532	3,225	68	7,56	66/65	1,087/3,85
max1024	3232	1278	0,627	264	0,88	263/260	0,161/4,77
test1	365	2100	0,635	111	0,07	111/110	0,35/57,59
test2	7122	106949	0,082	–	–	900/860	313,3/41710,5
test3	3543	40680	0,168	478	3121,25	460/433	52,9/7256,4
test4	1516	6139	1,276	105	34,12	99/92	23,6/14579,8

Результаты испытаний программ Espresso и Tie на псевдослучайных системах булевых функций представлены в табл. 2. Для создания примеров использовался генератор псевдослучайных булевых функций [2]. Генератор, формирующий систему СДНФ, управляется следующими параметрами: *x* – число переменных; *y* – число функций в системе; *z* – процентное соотношение количества единиц в значениях данной системы булевых функций. В процессе минимизации строится булева матрица $m \times n$, на которой тестируется разработанный алгоритм.

Из табл. 2 видно, что программа Espresso-Qm не смогла найти решение для систем функций $f(x_1, x_2, ..., x_n)$, где число столбцов матрицы Квайна $n > 25000$. Для этих примеров приведено время решения задачи покрытия (в скобках) и общее время минимизации. В табл. 2 даны результаты решения примеров программой Espresso, для того чтобы показать возможное решение примеров, для которых программа Espresso-Qm не нашла решения. Следует отметить, что программа Espresso создает упрощенную матрицу Квайна, поэтому для нее указано общее время минимизации.

Таблица 2

Сравнение эффективности алгоритма покрытия на псевдослучайных системах булевых функций

Пример (*fx_y_z*)	*m*	*n*	*p*	Espresso		Espresso-Qm		Tie	
				k	*t,* с	*k*	*t,* с	*k*	*t,* с
f8_8_50	1040	1273	0,552	233	0,14	220	0,81	222/217	0,04/0,12
f9_8_50	2129	3066	0,277	453	0,78	425	21,34	431/413	0,265/17,86
f12_8_50	16872	34859	0,037	3366	31,34	–	–	3158	13,68(12,91)
f10_8_60	5026	9744	0,157	833	1,43	798	815,31	811/744	1,355/63,03
f10_12_50	6319	10148	0,113	1002	1,21	1020	1266,15	991/967	0,76/8,82
f11_11_30	6974	8564	0,056	1949	4,48	1910	101,06	1921/1894	0,880/53,07
f12_4_50	8405	15414	0,061	2100	9,65	1965	6840,95	1970/1908	3,11/33,03
f12_4_75	12434	38418	0,090	1633	9,26	–	–	1524	9,52(7,07)
f13_2_35	5955	7830	0,059	2159	6,10	2039	91,88	2044/2029	3,75/19,74
f14_4_25	17354	22129	0,017	7044	87,25	6713	516,61	6770/6701	1,98/89,47
f14_16_50	133624	324825	0,006	16299	668,96	–	–	16196	99,58(39,58)
f15_12_40	161323	356640	0,003	31594	4317,0	–	–	30770	91,88(59,49)
f16_8_50	267869	819668	0,002	50844	18365,0	–	–	48282	288,52(171,17)
f17_1_50	66694	155177	0,008	15721	2193,0	–	–	14844	79,04(53,84)
f18_6_25	410619	651541	0,001	–	–	–	–	138597	189,48(52,23)
f19_8_15	672498	882519	0,001	–	–	–	–	284890	214,56(60,48)
f22_4_20	3528659	5450563	0,000078	–	–	–	–	1351632	4538,0

В табл. 3 представлены результаты экспериментов на сериях примеров CLR и Stein, где m – число строк в матрице; n – число столбцов в матрице; GANP – решение, полученное алгоритмом GANP [10]; t_{GANP} – время, затраченное на поиск решения алгоритмом GANP; Tie – решение, полученное алгоритмом Tie; t_{tie} – время, затраченное на поиск решения алгоритмом Tie.

Таблица 3

Сравнение эффективности алгоритма покрытия на примерах серий CLR и Stein

Пример	m	n	GANP	t_{GANP}, с (Pentium, 100 ГГц)	Tie	t_{tie}, с (Pentium-4, 1,7 ГГц)
CLR.9	255	126	26	15,32	26	0,351
CLR.10	511	210	25	27,0	28	0,086
CLR.11	1023	330	23	274,6	26	1,426
CLR.12	2047	495	23	979,7	29	3,738
CLR.13	4095	715	23	4615,4	29	26,58
Stein.27	117	27	18	1,2	18	0,032
Stein.45	330	45	30	12,6	30	0,234
Stein.81	1080	81	61	16,6	61	0,198
Stein.135	3015	135	104	212,4	104	0,379
Stein.243	9801	243	198	536,1	198	34,73

Из табл. 3 видно, что алгоритм Tie показал хорошие результаты на серии примеров Stein, однако не нашел оптимального решения на примерах серии CLR. Это связано со специфическими свойствами матриц из примеров серии CLR.

На основании проведенных наблюдений установлено, что свойства матриц из примеров серии CLR значительно отличаются от свойств матриц, строящихся в процессе минимизации булевых функций. Так, доля единиц в матрицах серии CLR составляет около 25 %, тогда как доля единиц в матрицах, образованных при минимизации булевых функций, составляет от 0,0023 до 3,22 % (в среднем 0,6 %), т. е. матрицы Квайна являются разреженными.

Заключение

При решении задачи минимизации систем булевых функций в классе ДНФ строятся матрицы Квайна, обладающие свойством разреженности, на которые и ориентирован разработанный алгоритм. Программная реализация разработанного алгоритма покрытия и его экспериментальные испытания подтверждают высокую практическую эффективность как на серии примеров Benchmark, так и на псевдослучайных системах булевых функций. Предложенный алгоритм можно применить для решения других логических задач, которые могут быть сведены к задаче наименьшего покрытия множества.

Список литературы

1. Сапоженко, А.А. Минимизация булевых функций в классе дизъюнктивных нормальных форм / А.А. Сапоженко, И.П. Чухров // Итоги науки и техники. – М., 1987. – Т. 25. – С. 68–116.

2. Леончик, П.В. Минимизация систем булевых функций в классе дизъюнктивных нормальных форм / П.В. Леончик // Информатика. – 2006. – № 1(9). – С. 88–96.

3. Закревский, А.Д. Основы логического проектирования. Комбинаторные алгоритмы дискретной математики. Кн. 1 / А.Д. Закревский, Ю.В. Поттосин, Л.Д. Черемисинова. – Минск: ОИПИ НАН Беларуси, 2004.

4. Yang, S. Logic synthesis and optimization benchmarks user guide version 3.0. Technical Report / S. Yang. – Microelectronics Center of North Carolina, 1991.

5. Закревский, А.Д. Генераторы псевдослучайных логико-комбинаторных объектов / А.Д. Закревский, Н.Р. Торопов // Логическое проектирование: сб. науч. тр. – Минск: Ин-т техн. кибернетики НАН Беларуси, 1999. – С. 49–63.

6. Beasley, J.E. OR-Library: Distributing test problems by electronic mail / J.E. Beasley // Journal of the Operational Research Society. – 1990. – Vol. 41, № 11. – P. 1069–1072.

7. Logic Minimization Algorithms for VLSI synthesis / R.K. Brayton [et al.]. – Boston: Kluwer Academic Publishers, 1984.

8. ESPRESSO-II: a New Logic Minimizer for PLA / R.K. Brayton [et al.] // IEEE Pros. Custom Integrated Circuits Conf. – Rochester, 1984.

9. Rudel, R. Multiple-Valued Minimization for PLA Optimization / R. Rudel, A. Sangiovanni Vincentelli // IEEE Trans. Computer-Aided Des. – 1987. – Vol. CAD-6, № 5. – P. 727–751.

10. Еремеев, А.В. Генетический алгоритм для задачи о покрытии / А.В. Еремеев // Дискретный анализ и исследование операций. – 2000. – Сер. 2. – Т. 7, № 1. – С. 47–60.

Поступила 12.12.06

Объединенный институт проблем
информатики НАН Беларуси,
Минск, Сурганова, 6
e-mail: pleonchik@gmail.com

P.V. Liavonchyk

ALGORITHM OF SPARSE BOOLEAN MATRIX COVERING

The algorithm for solving a minimum set covering problem is presented. The problem is known as finding the minimum column covering a Boolean matrix. The comparative analysis of the effectiveness of the algorithm realized in the programme Tie, the algorithm of the programme Espresso and the algorithm GANP is suggested. The results of statistical experimental testing of the algorithms on a standard set of Benchmark series, pseudorandom Boolean function systems and on a set of CLR and Stein series are presented.