Д. т. н. С. Ю. ЛУЗИН, к. т. н. О. Б. ПОЛУБАСОВ

Россия, г. Санкт-Петербург, АО "Авангард"
E-mail: luzin1@rol.ru

Дата поступления в редакцию
18.03 2003 г.

Оппонент *к. т. н. А. А. НИКОЛЕНКО*
(ОНПУ, г. Одесса)

ЭКОНОМИЧНЫЙ МЕТОД ПРИБЛИЖЕННОЙ МИНИМИЗАЦИИ ДНФ БУЛЕВЫХ ФУНКЦИЙ

Результаты, полученные с помощью предложенного метода, более точны, чем решения, полученные с помощью других эвристик.

К настоящему времени опубликовано значительное количество работ, посвященных проблемам минимизации булевых функций. Однако практика проектирования СБИС и постоянный рост степени интеграции требуют поиска новых, более эффективных моделей и алгоритмов. Более того, особое значение приобретает проблема оценки степени оптимальности получаемых решений, поскольку при современных размерностях задач использование переборных алгоритмов невозможно, а эвристические могут давать значительную погрешность.

Обычно при использовании эвристических алгоритмов относительная погрешность решения (отношение мощности найденного некоторым алгоритмом значения к мощности оптимума) является функцией размерности задачи. Поэтому при использовании эвристик для решения задач большой размерности встает дополнительный вопрос о величине отклонения от оптимума.

Большинство эвристик используют идею конкурирующих интервалов [1], однако применяют различные схемы поиска. Так, в работах [2, 3] используется поиск в глубину, когда осуществляется расширение некоторого терма до получения максимального интервала. Другие, например [4], используют поиск в ширину, осуществляя параллельное расширение интервалов таким образом, чтобы покрыть исходное множество минимальным числом непересекающихся интервалов. Поиск в ширину существенно увеличивает комбинаторную сложность, однако в среднем позволяет получать более точные решения.

Существенным недостатком большинства алгоритмов минимизации булевых функций, в т. ч. и приближенных, является необходимость попарного сравнения термов, что ограничивает быстродействие и предъявляет повышенные требования к необходимому объему памяти.

Таким образом, для повышения эффективности решения задач реальных размерностей необходима разработка новых подходов, обладающих минимальной комбинаторной сложностью и при этом гарантирующих получение оптимального или близкого к нему решения.

В работе [5] описан метод поразрядного выращивания, осуществляющий параллельное выращивание простых импликант без попарного сравнения. Как показано в [5], метод является асимптотически оптимальным в смысле быстродействия. В настоящей работе также используется поразрядное выращивание простых импликант, но не всех возможных, а только образующих кратчайшее (или близкое к нему) покрытие исходных импликант.

Жадный алгоритм

Пусть минимизируемая функция представлена в виде совершенной нормальной дизъюнктивной формы (СДНФ). Каждой элементарной конъюнкции поставлен в соответствие ее код, двоичный или десятичный. Например, конъюнкции $x_1x_2\bar{x}_3x_4$ соответствует двоичный код 1011, или десятичный 11. (Здесь и в дальнейшем меньшему номеру переменной соответствует меньший разряд двоичного кода, т. е. в кодах переменные записываются слева направо.) Все коды ранжированы в порядке возрастания.

Отметим, что в любом из минитермов некоторая переменная может принимать одно из трех значений: отсутствовать (—), присутствовать (1), присутствовать со знаком отрицания (0). Рассматривая только соседние термы, можно установить значение первой переменной в импликантах и отделить это значение от самих импликант, понизив ранг последних.

Это обстоятельство использовалось в [1] для поиска множества простых импликант методом "поразрядного выращивания". При этом множество импликант делилось на классы и подклассы, причем "выращенная" кодовая комбинация использовалась как код класса, каждый из которых содержал остатки кодов импликант. В предлагаемом методе кодом класса будет являться остаток кода импликант, а в каждом из классов будут выращенные кодовые комбинации, имеющие общий остаток.

Таким образом, на любом этапе, кроме первого и последнего, каждая из импликант характеризуется парой векторов: двоичным вектором β_j (или его десятичным эквивалентом), являющимся общим остатком импликант j-го класса, и троичным вектором $\alpha_{i,j}$ (выращенной комбинацией). Соответственно, на первом этапе имеется только множество двоичных векторов β_j, а на последнем — только множество троичных векторов $\alpha_{i,j}$.

В отличие от [1], будем определять смежность не пары термов, а пары классов. Два класса β_j и β_{j+1} смежны, если β_j — четно, и $\beta_{j+1}-\beta_j=1$ (если за четным классом следует нечетный класс с остатком кода, большим на единицу).

Метод заключается в проверке условия смежности для соседних классов и перераспределении компонент векторов. При этом на каждом шаге убирается младший разряд из кода остатка и добавляется в старший разряд выращенного троичного кода. Заметим, что на любом из этапов суммарное число компонент векторов не увеличивается.

Опишем правила перераспределения компонент векторов.

1. Если β_j четно и $\beta_{j+1}-\beta_j\neq 1$ (за четным классом не следует нечетный с остатком кода, большим на единицу), то младший разряд из β_j переходит в старший разряд всех выращенных кодовых комбинаций ($\alpha_{i,j}$) класса β_j:

$$\forall i\in\overline{1,k},\ \alpha'_{i,j}:=(0)\alpha_{i,j},$$
$$\beta'_j:=\beta_j/2.$$

2. Если β_j нечетно и $\beta_j-\beta_{j-1}\neq 1$ (перед нечетным классом нет четного класса, меньшего на единицу), то

$$\forall i\in\overline{1,k},\ \alpha'_{i,j}:=(1)\alpha_{i,j},$$
$$\beta'_j:=(\beta_j-1)/2.$$

3. Если β_j четно и $\beta_{j+1}-\beta_j=1$ (если за четным классом следует нечетный с остатком кода, большим на единицу), то образуется класс, остаток кода которого получается делением на 2 остатка кода четного класса: $\beta'_j:=(\beta_j-1)/1$. В этот класс переходят все выращенные кодовые комбинации обоих (четного и нечетного) классов, причем

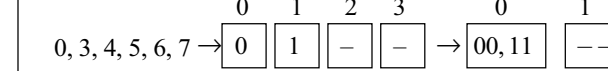
— если в смежных классах была пара одинаковых выращенных кодовых комбинаций ($\alpha_{i,j}=\alpha_{k,j+1}$), то в новом классе ($\beta'_j$) они представлены той же комбинацией с добавлением черточки в старший разряд: $\alpha'_{i,j}=(-)\alpha_{i,j}$;

— если некоторая выращенная кодовая комбинация поглощается выращенной кодовой комбинацией из смежного класса ($\alpha_{i,j}\backslash\alpha_{k,j+1}=\alpha_{i,j}$ или $\alpha_{i,j+1}\backslash\alpha_{k,j}=\alpha_{i,j+1}$), то в новый класс заносится поглощаемая кодовая комбинация с добавлением черточки в старший разряд (соответственно $\alpha'_{i,j}=(-)\alpha_{i,j}$ или $\alpha'_{i,j}=(-)\alpha_{i,j-1}$).

Непоглощаемые кодовые комбинации переходят из классов β_j и β_{j+1} в новый класс β'_j с добавлением в старший разряд, соответственно, 0 или 1:

$\alpha'_{i,j}:=(0)\alpha_{i,j}$ и $\alpha'_{i,j}:=(1)\alpha_{i,j}$).

Пример: $F=\Sigma$ 0, 3, 4, 5, 6, 7



→ –1 00, –11, 1– –

Примечание. В рамке — выращенные кодовые комбинации ($\alpha_{i,j}$), над рамками — общие для термов класса остатки кодов импликант (β_j).

Коррекция решения

Приведенный подход отличается высоким быстродействием и требует минимального объема памяти, однако полученное решение может значительно отличаться от оптимального. Точность решения можно повысить за счет перестройки термов с целью высвобождения взаимных пересечений.

Пересекающиеся термы образуются при расширении интервалов, имеющих смежные остатки кода импликанты, но несовпадающие выращенные коды, в случае поглощения одного из выращенных кодов другим. При этом мощность пересечения пары равномощных интервалов равна половине мощности интервала. В случае интервалов различной мощности мощность пересечения будет равна половине мощности меньшего интервала.

Перестроить интервалы, высвободив при этом пересечения, можно, если в смежных классах имеются пересекающиеся термы. Однако это условие является только необходимым.

Пусть имеем $\alpha_{i,j}$ терм из класса β_j, $\alpha_{l,j+1}$ терм из класса β_{j+1}, при этом классы β_j и β_{j+1} смежны, и

$$|\alpha_{i,j}\cap\alpha_{l,j+1}|=|\alpha_{i,j}/2|=|\alpha_{l,j+1}/2|.$$

Термы $\alpha_{i,j}$ и $\alpha_{l,j+1}$ склеиваются с высвобождением пересечений, если

$$\alpha_{i,j}\setminus(\alpha_{i,j}\cap\alpha_{l,j+1})\subset\bigcup_k\alpha_{k,j}$$

и

$$\alpha_{l,j+1}\setminus(\alpha_{i,j}\cap\alpha_{l,j+1})\subset\bigcup_k\alpha_{k,j+1}.$$

В этом случае вместо двух интервалов $(0)\alpha'_{i,j}$ и $(1)\alpha'_{l,j+1}$ получается один интервал $(-)\alpha'_{i,j}$, где $\alpha'_{i,j}=\alpha'_{i,j}\cap\alpha'_{l,j+1}$. Отметим, что поиск пары интервалов для подобного склеивания необходимо осуществлять в случае невозможности расширения их на данном шаге.

Введение процедуры коррекции несколько снижает быстродействие алгоритма, однако существенно повышает точность решения. При этом процедура носит локальный характер, поскольку осуществляется попарное сравнение только термов смежных классов. Кроме того, постоянное увеличение мощности классов сопровождается уменьшением суммарного количества термов.

Для снижения затрат, связанных с поиском пар термов для проведения коррекции, следует предварительно отсортировать термы в классах в порядке возрастания кодов импликант, а для одинаковых кодов импликант — по кодам разности. Помимо снижения комбинаторной сложности, это может повысить и точность решения, поскольку повышается вероятность того, что одинаковые термы в различных классах будут перестраиваться синхронно (идентично). Максимизация количества одинаковых остатков кодов импликант — залог их склеивания на последующих этапах и минимизации общего числа импликант.

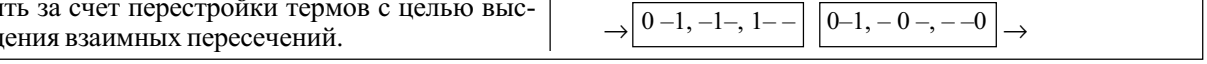
Пример: $F=\Sigma$ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14

Поскольку $--1\backslash(--1\cap 0---)=1-1\subset 1--$

и

$0---\backslash(--1\cap 0---)=0-0\subset ---0$,

то



Поскольку –1–\(–1– ∩ – –0)=–11⊂ (0–1∪1– –)
и
– –0\(–1– ∩ – –0)= – 00 ⊂ – 0–,
то

$$\rightarrow \boxed{0–1, –10,\ 1– –}\ \boxed{0\ –1, –10, – 0 –} \rightarrow – 0–1, – –10,\ 01– –, 1–0–$$

Минимизация функции, заданной набором троичных векторов

В случае если характеристическое множество функции содержит слишком много элементов, задать ее в виде совершенной ДНФ практически сложно, а иногда даже невозможно. В связи с этим несомненный интерес представляют методы, позволяющие минимизировать сокращенную ДНФ без предварительного получения совершенной ДНФ и всех простых импликант.

Пусть функция задана набором троичных векторов. Введем операцию расщепления термов по некоторой переменной: в каждом из троичных векторов удалим компоненту, соответствующую некоторой переменной, и разобьем множество на два класса. В первый класс попадут термы со значением ведущей переменной "0", во второй — со значением "1". Термы со значением переменной "–" попадут в оба класса.

АЛГОРИТМ (рекурсивный)

Построить множество простых импликант
{
1. Получить множество входных импликант.
2. Отбросить поглощаемые импликанты.
3. Если осталось меньше двух импликант => ОТВЕТ.
Иначе
{
1. Выбрать ведущую переменную (см. примечание).
2. Расщепить множество входных импликант по ведущей переменной на классы "0" и "1" (термы с "–" попадут в оба класса).
3. Построить множество простых импликант для класса "0".
4. Построить множество простых импликант для класса "1".
5. Слить два класса в один (используя склеивание и высвобождение).
=>ОТВЕТ.
}
}

Примечание. В качестве ведущей целесообразно выбирать переменную с меньшим числом "–"; при одинаковом числе "–" следует выбрать переменную с более равномерным распределением "0" и "1". При этом в разных ветвях выбор ведущих переменных может быть неодинаковым.

Пример: Пусть $F=\bar{x}_1\bar{x}_4 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_4 \vee \bar{x}_3\bar{x}_4$
Набор соответствующих троичных векторов:
0 – – 0, – 00 –, – –11, 0 –1–, 00 – –.

В качестве ведущей выбираем переменную x_2, осуществляем расщепление и удаляем поглощаемые термы:

$$\boxed{0 – 0, 00 –, – 0 –}\ \boxed{0 – 0, 00 –, – –1, 0 – –}$$

Поскольку 0– –\(0– –∩0–0)=0 –1⊂– –1,

$$\boxed{0 – 0, – 0 –}\ \boxed{0 – 0, – –1} \rightarrow 0 – – 0, – 00 –, – –11$$

Заключение

Описанный в работе метод не гарантирует получения кратчайшей ДНФ. Тем не менее, благодаря систематической коррекции частичного решения, результаты, полученные с помощью данного метода, более точны, чем решения, полученные с помощью других эвристик. Так, например, сравнение с методами, реализованными в пакете Abel фирмы Data I/O, продемонстрировало, что преимущество в точности сказывается уже на простых примерах.

Проверка быстродействия программных средств осуществлялась на специально сгенерированных тестовых примерах, ядро которых представляет собой функцию вида

$F=x_1 \vee x_2 \vee \ldots \vee x_n$.

На подобных примерах легко контролируется точность решения, т. к. оптимальный результат заранее известен. (При добавлении термов, не содержащих терма $\bar{x}_1 \cdot \bar{x}_2 \cdot \ldots \cdot \bar{x}_n$, результат оптимизации $F'=F$, в противном случае $F'=1$.)

Представленная **таблица** позволяет судить о быстродействии алгоритма и его программной реализации. Сравнение с другими программами затруднительно, поскольку известные системы имеют существенные ограничения по числу независимых переменных (входов устройства).

Количество переменных	Число термов	Время, с
100	100	3
150	150	6
200	200	14
250	250	27
300	300	46
350	350	71
400	400	106
450	450	152
500	500	225

Отметим, что, в отличие от известных подходов, данный метод позволяет распараллелить процесс минимизации функции, что позволяет создавать эффективные реализации метода для многопроцессорных систем.

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. Закревский А. Д. Логический синтез каскадных схем.— М.: Наука, 1981.

2. Rhyne T. V., Noe P. S., Mc Kinney, Pooch U. W. A new technique for fast minimization of switching functions // IEEE Trans. Comput.— 1977.— Vol. C-26.— P. 757—764.

3. Perkins Sh. R., Rhyne T. V. An algorithm for identifying and selecting the prime implicants of multiple-output boolean function // IEEE Trans. Comput.— Aid. Des. Integr. Circuit and Syst.— 1988.— Vol. 7, N 11.— P. 1215—1218.

4. Hong S. J., Cain R. G., Ostapko D. L. MINI: A heuristic approach for logic minimization // IBM J. Res. Dev.— Sept. 1974.— Vol. 18.— P. 443—458.

5. Лузин С. Ю. Асимптотически оптимальный метод получения простых импликант // Автоматика и выч. техника.— 2000.— Вып. 1.— С. 80—84.