ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
2019 Управление, вычислительная техника и информатика № 48

УДК 519.17
DOI: 10.17223/19988605/48/9

Д.И. Черемисинов, Л.Д. Черемисинова

ПОИСК ПОДСХЕМ В КМОП СХЕМЕ ИЗ ТРАНЗИСТОРОВ МЕТОДОМ GRAPH MINING

Graph Mining – одно из направлений интеллектуального анализа данных, в котором объемные комплексные данные представлены в виде графов, а анализ ведется для того, чтобы получить новые знания. Задачей Graph Mining является обнаружение в графе типовых шаблонов. Общепринятым видом таких шаблонов являются часто встречающиеся подграфы. Рассматривается задача поиска часто встречающихся подграфов в большом графе и обсуждается применение этой задачи в преобразовании плоской КМОП (комплементарная структура металл-оксид-полупроводник) схемы из транзисторов в формате SPICE в иерархическую схему.
Ключевые слова: кластеризация графа; поиск часто встречающихся подграфов; изоморфизм подграфов; цветовое кодирование графа; КМОП схемы из транзисторов.

В последние годы наблюдается повышенный интерес к разработке алгоритмов интеллектуального анализа данных, работающих на графах. Такие графы возникают естественно в ряде различных областей, таких как обнаружение сетевых атак, семантический web, поведенческое моделирование, перепроектирование СБИС, анализ социальных сетей и классификация химических соединений. Когда данные сложны и имеют много взаимосвязей, на первый план выходит задача поиска знаний об их структуре. Важную роль в методах интеллектуального анализа для данных большого объема, моделируемых графами, играют алгоритмы поиска часто встречающихся подграфов. Задача поиска часто встречающихся подграфов состоит в нахождении в заданном графе всех тех подграфов, которые встречаются в нем с частотой большей, чем заданное значение. Идентификация часто встречающихся графов / подграфов в базе данных или в одном большом графе может использоваться для задач классификации, кластеризации графов, построения индексов при поиске подграфов.

По аналогии с программированием преобразование иерархической схемы электронного устройства в схему, состоящую исключительно из примитивных элементов, естественно назвать компиляцией. Обратный процесс, в результате которого из плоской транзисторной схемы строится иерархическая транзисторная схема, можно назвать декомпиляцией. Декомпилятор [1] как программа, восстанавливающая иерархическое описание схем, является одним из инструментов верификации лейаута, или перепроектирования (reengineering), схем [2]. Так же как при декомпиляции программ, целью декомпиляции схемы является замена представления схемы на низком (транзисторном) уровне более высокоуровневым ее представлением. В отличие от программного процесса декомпиляция схем не является языковой трансформацией, хотя декомпилятор использует определенный формат (формальный язык представления данных) схем. Языком, используемым декомпилятором схем, как правило, является формат SPICE (Simulation Program with Integrated Circuit Emphasis) для обмена электрическими схемами [3]. Формат SPICE позволяет описывать как схемы транзисторного уровня, так и иерархические.

В формате SPICE электрические схемы состоят из элементов, которые соединены друг с другом цепями. Главной частью описания схемы в формате SPICE является список транзисторов, в котором для каждого вывода транзистора указано имя цепи, соединяющей его с остальными частями схемы. Например, схема статического КМОП инвертора имеет по одному *n*-МОП и *p*-МОП транзистору, для каждого из четырех выводов (сток, затвор, исток, подложка) которых приведены имена цепей (рис. 1). Удобной моделью для поиска подсхем в исходной транзисторной схеме является помеченный неориентированный двудольный граф. Одну долю графа составляют вершины, соответствующие выводам

элементов и портам схемы (выводам всей электрической схемы), а другую – вершины, соответствующие цепям – соединениям между выводами.

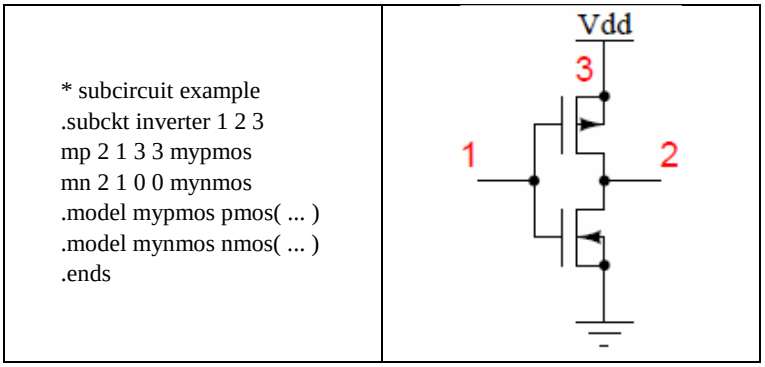

Рис. 1. Описание КМОП инвертора в формате SPICE и его электрическая схема
Fig. 1. Description of the CMOS inverter in SPICE format and its electrical circuit

На рисунках этот граф компактнее представляется в виде его реберного графа. Для графа G реберным называется граф $L(G)$, любая вершина которого представляет ребро графа G и две вершины графа $L(G)$ смежны тогда и только тогда, когда соответствующие им ребра смежны в G. Пометками (раскраской) элементов транзисторного уровня являются названия выводов транзисторов. На рис. 2 в графе КМОП схемы транзисторного уровня у транзисторов не показан вывод подложки, зеленым обозначен вывод «сток», красным – «исток», синим – «затвор».

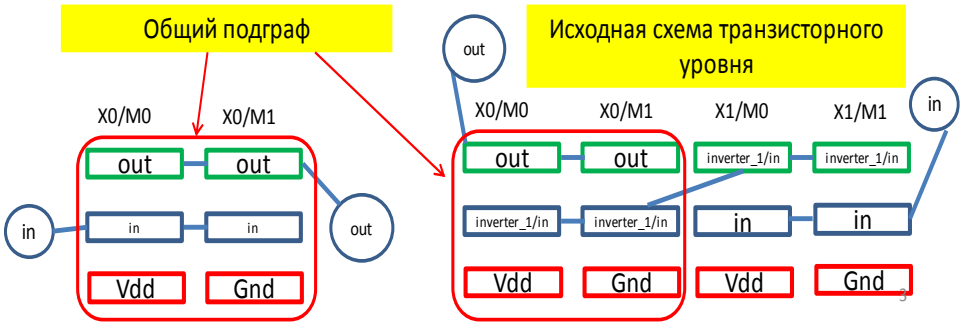

Рис. 2. Декомпиляция – поиск и извлечение подсхемы
Fig. 2. Decompiling – subcircuit recognition and extraction

Современные цифровые КМОП схемы содержат до миллиарда примитивных элементов на транзисторном уровне. КМОП – набор полупроводниковых технологий построения интегральных микросхем и соответствующая ей схемотехника микросхем. Для построения иерархического структурного описания нужно выделить наборы взаимосвязанных транзисторов в схеме в качестве отдельных компонентов, т.е. найти подсхемы в исходной плоской схеме (см. рис. 2). После замены подсхем из транзисторов элементами описание схемы становится двухуровневым.

Если искомые подсхемы библиотечных элементов заданы, т.е. известна библиотека подсхем, то образцы графов (шаблоны) для поиска формируются из описания этих библиотечных элементов. Задача декомпиляции в этом случае сводится к задаче поиска в исходном графе подграфов, изоморфных заданному [4].

В дальнейшем задача декомпиляции рассматривается только для случая, когда библиотека подсхем не известна. Если нужно распознать библиотеку подсхем, то задача в такой постановке поставлена некорректно, так как любой фрагмент, содержащий целое число транзисторов, может считаться частью библиотечной подсхемы. Нужны дополнительные критерии, позволяющие выделять подсхемы. Как и в подходе Graph Mining, далее рассматривается задача выделения в качестве подсхем фрагментов транзисторной КМОП схемы, которые являются часто встречающимися подграфами.

Алгоритм наивного поиска часто встречающихся подграфов состоит из двух операций. Первая операция разыскивает все подграфы-кандидаты для данного графа G, а вторая подсчитывает частоту встречаемости каждого найденного подграфа-кандидата. Поиск подграфов-кандидатов начинается с графов, состоящих из одной вершины. Новые вершины и ребра добавляются итеративно к уже созданным подграфам для создания новых кандидатов. Для каждого кандидата рассчитывается частота встречаемости. Подграф-кандидат считается часто встречаемым, если его частота встречаемости не меньше некоторого граничного значения. Такой алгоритм имеет огромную вычислительную сложность. Поиск подграфов-кандидатов требует перебора множества всех подмножеств (булеана) множества вершин графа G (число операций $O(2N)$, где N – число вершин в графе G, которое для современных КМОП схем имеет порядок 10^9. Подсчет частоты встречаемости одного подграфа-кандидата S имеет еще большую сложность ($O(2^{N^2} N^n)$, где N и n – число вершин в графах G и S).

1. Разбиение графа на подграфы

Задача разбиения графа на подграфы (graph partition) [5] состоит в представлении исходного графа $G = (V, E)$ в виде множества подмножеств вершин Sep $V = \{V_1, V_2, ..., V_n\}$, $V_i \subseteq V$ по определенным правилам. По условию задачи поиска подсхем в КМОП схеме требуется, чтобы $\bigcup_{i=1}^{n} V_i = V$, т.е. все вершины исходного графа должны быть распределены по подмножествам, причем все $V_i \neq \varnothing$. В зависимости от дополнительных условий, накладываемых на блоки разбиения V_i, возникает несколько задач этого типа. Например, в случае k-блочного разбиения множество вершин делят на не более чем k отдельных блоков и так, чтобы число ребер, соединяющих блоки, было минимальным. В задаче равномерного разбиения графа стремятся получить блоки примерно одинакового размера и минимизировать связи между блоками.

Задача разбиения графа на подграфы имеет множество применений. Например, эффективная реализация параллельных программ обычно требует решения проблемы разбиения графа, в котором вершины представляют вычислительные задачи, а ребра представляют обмен данными. В этом случае k-блочное разбиение графа вычислений может использоваться для назначения задач в компьютере с k процессорами. Поскольку каждый блок разбиения содержит одинаковое количество вычислительных задач, то решение задачи обеспечивает балансировку загрузки этих k процессоров, а так как при решении задачи минимизируется мощность множества разрезающих ребер, то одновременно обеспечивается и сокращение межпроцессорного обмена данными.

В последнее время проблема разбиения графов нашла новые применения в области кластеризации для обнаружения сообществ в социальных и биологических сетях. При поиске сообществ в социальных сетях делается попытка извлечь структуру из графа социальной сети, разделив его вершины на непересекающиеся подграфы (сообщества), такие что связи между вершинами подграфа включены в множество его ребер, а число соединений между подграфами мало.

Задача k-блочного разбиения графа часто решается рекурсивным делением его множества вершин пополам [5, 6]. То есть сначала получаем 2-блочное разбиение множества V, а затем разбиваем каждую из полученных частей пополам и так далее. После $\log k$ операций деления граф G разбивается на k частей. Таким образом, задача k-блочного разбиения графа сводится к последовательности 2-блочных разбиений. Эта схема широко используется из-за своей простоты, хотя она не всегда приводит к оптимальному разбиению.

Задача разбиения на подграфы может быть расширена на гиперграфы, где ребро может соединять более двух вершин [7]. Гиперребро не разрезается, если все вершины находятся в одном блоке, и разрезается ровно один раз в противном случае, независимо от того, сколько вершин находится с каждой стороны. Эта постановка задачи возникает в задаче поиска подсхем в схеме из транзисторов, возникающей в процессе проектирования СБИС.

Еще одно важное применение рекурсивного деления пополам заключается в том, чтобы найти порядок заполнения при факторизации разреженных матриц. Этот тип алгоритмов обычно называют алгоритмами упорядочения вершин путем вложенных диссекций. Вложенная диссекция рекурсивно разбивает граф на почти равные половины, пока не будет получено требуемое количество разделов. Достигаемое при этом упорядочение вершин обеспечивает ускорение умножения разреженной матрицы на вектор.

На практике для решения задачи k-блочного разбиения графа используются эвристические многоуровневые алгоритмы. Эти алгоритмы состоят из трех фаз. На фазе огрубления граф сжимается, чтобы получить иерархию графов меньших размеров. В фазе разбиения применяется алгоритм 2-блочного разбиения к наименьшему графу, построенному на первой фазе. На фазе восстановления это разбиение проецируется на граф меньшего уровня иерархии, с тем чтобы получить другое разбиение графа этого уровня. На каждом уровне используется метод поиска локального минимума для улучшения разбиения, полученного на предыдущем этапе. Формально алгоритм многоуровневой бисекции графа описывается следующим образом.

Для заданного графа $G_0 = (V_0, E_0)$ алгоритм многоуровневой бисекции состоит из следующих трех этапов. Этап огрубления преобразует граф G_0 в последовательность меньших графов $G_1, G_2, \ldots, G_m$, таких что $|V_0| > |V_1| > |V_2| > \ldots > |V_m|$. На этапе разбиения производится 2-блочное разбиение R_k графа $G_k = (V_k, E_k)$, которое разбивает V_k на две части, каждая из которых содержит половину вершин графа G_k. Фаза восстановления проецирует разбиение R_k графа G_k обратно в G_0, проходя промежуточные разбиения $R_{k-1}, R_{k-2}, \ldots, R_1, R_0$.

Лучшие программы многоуровневой бисекции позволяют получать k-блочные разбиения графов с сотнями тысяч вершин за приемлемое время на персональном компьютере, когда k не превышает сотни.

Следует отметить, что описанные алгоритмы k-блочного разбиения графа не позволяют получить решение (за приемлемое время) задачи поиска часто встречающихся подграфов для графов с большим числом вершин (миллион и более). Они могут служить только разумным упрощением операции поиска подграфов-кандидатов.

2. Обзор алгоритмов поиска часто встречающихся подграфов

Целью подобных алгоритмов является поиск полного множества часто встречающихся подграфов с использованием некоторой меры частоты встречаемости. Множество часто встречающихся подграфов составляют подграфы, у которых мера частоты встречаемости (support) больше заданной границы. Для решения задачи поиска полного множества часто встречающихся подграфов используются эвристические алгоритмы, эффективная работа которых зависит от специфических характеристик исходного графа, позволяющих, например, отличить эти графы от случайных графов произвольного вида. Обычно предполагается, что в исследуемом графе большинство вершин имеет небольшую степень и небольшой процент вершин имеет большое количество соединений. Это свойство выражается формально степенным законом распределения (power law distribution) числа вершин в зависимости от их степени.

Задача поиска часто встречающихся подграфов стала популярной областью исследований в последнее десятилетие, и к настоящему времени библиография этой задачи насчитывает сотни публикаций, однако литература на русском языке по данной проблеме практически отсутствует. Было предложено много алгоритмов поиска часто встречающихся подграфов. Последние из предложенных алгоритмов превосходят ранние разработки с точки зрения требований к памяти и быстродействию на нескольких порядков. Но ни один из алгоритмов не позволяет полностью решить проблему огромной сложности вычислений. Предлагаемые методы сокращения сложности вычислений за счет использования эвристик, не гарантирующих нахождения полного множества часто встречающихся подграфов, по-прежнему позволяют решать задачи большой размерности за экспоненциальное время.

Алгоритмы поиска часто встречающихся подграфов строятся по аналогии с подходами, используемыми в хорошо известной в задаче Data mining поиска часто встречающегося множества атрибутов (frequent item set mining). В этой задаче разыскиваются множества элементов, встречающиеся в достаточно большом количестве транзакций данной базы данных. Для задачи предложены алгоритмы, трудоемкость которых меньше полиномиальной, хотя множество всех решений составляет булеан множества элементов базы данных. Трудоемкость поиска часто встречающихся графов имеет более высокий порядок из-за необходимости решать задачу изоморфизма подграфа. К алгоритму перебора подграфов-кандидатов предъявляется требование по возможности избегать генераций кандидатов, которые нечасто встречаются или изоморфны ранее найденным, с тем чтобы как можно реже выполнять тест проверки изоморфизма подграфов.

Если вершины и ребра графа имеют уникальные метки, то граф можно задать множеством меток ребер, а затем использовать существующие алгоритмы поиска часто встречающегося множества атрибутов, чтобы найти все часто встречающиеся подграфы. Так как отображение вершин и ребер на метки не однозначно, то это сведение не всегда возможно, однако переход к маркированным (раскрашенным) графам позволяет сделать тест изоморфизма подграфа более эффективным. Кроме того, для генерации часто встречающегося множества атрибутов можно использовать подход frequent item set, основанный на «apriori принципе», который для задачи поиска часто встречающихся подграфов формулируется следующим образом [9]: все графы, содержащие нечасто встречающийся подграф, тоже не относятся к часто встречающимся.

Маркированный (раскрашенный) граф $G = (V, E, L, l)$, представляется множеством V вершин, множеством дуг $E \subseteq V \times V$ и функцией l: $V \cup E \to L$ отображения вершин и дуг в множество меток L. Граф $S = (V_S, E_S, L_S, l)$ является подграфом графа $G = (V, E, L, l)$, если $V_S \subseteq V$, $E_S \subseteq E$ и $L_S(u) = L(u)$ для всех $u \in V_S \cup E_S$.

Отношение изоморфизма подграфов между графами g_1 и G является отношением изоморфизма между g_1 и подграфом S графа G. Граф g_1 называется шаблоном, а S – вхождением шаблона g_1 в G.

Для подграфа G_s и графа G два вхождения G_s в G называются одинаковыми, если они используют в G один и тот же набор ребер. Два вхождения G_s в G называются непересекающимися по вершинам, если они не имеют общих вершин в G. Два вхождения G_s в G называются непересекающимися по ребрам, если они не имеют общих ребер в G.

В подходе на основе «apriori принципа» перед генерацией подграфов-кандидатов размера $k + 1$ необходимо иметь все кандидаты размера k, где размер графа определяется числом его вершин [8]. Чтобы получить следующий подграф-кандидат, два подграфа размера k объединяются вместе, формируя граф размера $k + 1$. Чтобы выбрать два кандидата для объединения, нужно, чтобы они имели общий подграф (тест на изоморфизм подграфов). Сгенерированный граф может быть изоморфным кандидату, сгенерированному ранее (тест на изоморфизм графов), в этом случае он отбрасывается. Сгенерированный граф может быть нечасто встречающимся (массовый тест на изоморфизм подграфов [9]), в этом случае он также отбрасывается. При подсчете числа вхождений кандидата нужно учитывать, что вхождения могут перекрываться. Если это произойдет, необходимо указать, какие вхождения должны быть приняты во внимание, чтобы «apriori принцип» выполнялся.

В методе grow-and-store [10], позволяющем уменьшить число проверок на изоморфизм, выполняются следующие шаги.

1. Находятся вершины, у которых число вхождений больше заданной границы, и сохраняются все найденные вхождения в списке вхождений.

2. Каждое вхождение из списка расширяется на одну вершину, образуя подграф-кандидат, оценивается частота его вхождения и, если он часто встречающийся, сохраняются все его вхождения. Это действие выполняется до тех пор, пока находятся новые часто встречающиеся подграфы.

Оценка частоты встречаемости подграфа здесь не требует теста на изоморфизм. Основным «узким» местом этого подхода является создание списка для хранения всех вхождений каждого из часто встречающихся подграфов. Этот список оказывается неприемлемо большим для вычисления и хранения.

Генерирование кандидатов, исходя из часто встречающихся подграфов меньшего размера, позволяет радикально уменьшить пространство поиска часто встречающихся подграфов заданного графа. Однако проблемой остается тест на включение кандидата в множество часто встречающихся подграфов. Тривиальным решением этой проблемы является выполнение теста кандидата на изоморфизм с каждым графом множества часто встречающихся подграфов. Эта проверка выполняется более экономным способом, если каждый граф из множества часто встречающихся подграфов канонизирован [10] и хранится в таблице по хеш-адресу, вычисленному по канонической маркировке подграфа. Когда в этой таблице по хеш-адресу, вычисленному по канонической маркировке подграфа-кандидата, хранится подграф, то он изоморфен тестируемому подграфу. Наибольший выигрыш получается, если кандидаты генерируются в канонической маркировке. Канонические маркировки графов, включенных в решение, лексикографически упорядочиваются, и построение подграфа-кандидата заключается в нахождении маркировки, которая больше всех уже построенных маркировок.

3. Цветовое кодирование подграфов

Другой класс алгоритмов подсчета частоты встречаемости подграфов основан на мощной методике их цветового кодирования (color coding) [11]. Исходными данными для метода служит шаблон и раскрашенный граф $G = (V, E, L, l)$, представляемый множествами вершин V, дуг E и функций l отображения вершин и дуг в множество красок L. Идея цветового кодирования заключается в случайной раскраске графа, соответствующего шаблону, с использованием k цветов (через k обозначается размер шаблона) из множества L. Далее в исходном раскрашенном графе $G = (V, E, L, l)$ подсчитывается количество «красочных» вхождений шаблона, т.е. подграфов, в которых все вершины имеют разные цвета. Затем это количество масштабируется по вероятности того, что вхождение шаблона является красочным. Среднее из масштабированных оценок частоты вхождений шаблона после нескольких итераций с различными случайными раскрасками исходного графа дает частоту вхождений шаблона. Метод цветового кодирования позволяет приблизительно подсчитать частоту вхождений некоторого заданного шаблона в большой граф. Главным недостатком алгоритмов цветового кодирования с вычислительной точки зрения является рост их потребности в памяти, необходимой для представления результатов, пропорционально росту размера шаблона.

Теоретически самый быстрый алгоритм подсчета вхождений k-вершинного подграфа в n-вершинный граф выполняется за время $n^{\omega k/3}$, где $O(n^{\omega})$ – временная сложность матричного умножения (в настоящее время $\omega \approx 2{,}38$). Это, конечно, лучше временной сложности $O(2^{n^2} n^k)$ тривиального алгоритма, но все равно чрезмерно велико даже для шаблонов с размером k порядка нескольких десятков.

4. Поиск часто встречающихся подсхем в КМОП схеме из транзисторов

Топология двудольных графов, которые моделируют схемы из транзисторов, имеет некоторые специфические свойства. Например, схемы транзисторного уровня, выполненные по КМОП технологии, имеют всего два типа элементов: n-МОП и p-МОП транзисторы. Каждый из них имеет по четыре вывода, соответствующие стоку, затвору, истоку и подложке (см. рис. 1). Выводы транзисторов и порты схемы (выводы всей электрической схемы – входные и выходные полюсы) соответствуют вершинам первой доли двудольного графа. Соответственно, каждая из вершин первой доли может иметь один из восьми возможных цветов, если она соответствует выводу транзистора n-МОП и p-МОП типа. Вершины, соответствующие разным портам схемы, раскрашены в разные цвета (но не из числа упомянутых восьми цветов).

Вторая особенность схемы из транзисторов заключается в том, что каждая вершина, принадлежащая второй доле и соответствующая цепи транзисторной схемы, определяет свою компоненту связности графа. Такая вершина может характеризоваться значением ее степени. Большинство

вершин второй доли имеет небольшие степени, но выделяется несколько вершин с большими степенями; такие вершины соответствуют, например, цепям питания, земли, тактирования. Для того чтобы метод Graph Mining работал эффективно, необходимо, чтобы распределение числа вершин в зависимости от их степеней в графе выражалось степенным законом (power law distribution). Можно считать, что это условие выполняется для двудольных графов, которые представляют схемы из транзисторов.

При распознавании подсхемы внутри схемы уровня транзисторов выделяются вхождения в последнюю некоторого шаблона (графа, задающего основу подсхемы). Подсхема строится по шаблону включением в нее вершин, соответствующих выводам, упоминаемым в шаблоне, и ребер, соединяющих эти вершины с вершинами, соответствующими цепям шаблона. Для формирования подсхемы цепи шаблона должны быть разбиты на два непересекающихся подмножества: внутренние (локальные) цепи и внешние цепи. В графе подсхемы внешние цепи шаблона соединены с выводами подсхемы. При построении шаблонов разбиением графа на подграфы внешними цепями являются цепи, которые входят в разрез экземпляра шаблона. Для часто встречающихся подграфов внутренние цепи локальны во всех экземплярах подграфа, остальные цепи являются внешними цепями шаблона подсхемы.

Набор шаблонов для заданной КМОП схемы из транзисторов может строиться разбиением графа схемы на подграфы на основе классов изоморфности этих подграфов. Но так как критерием разбиения служит минимизация некоторой оценки разреза (числа межсоединений подграфов), то этот метод получает подграфы с числом локальных межсоединений, бóльшим числа внешних межсоединений. В КМОП схемах из транзисторов такие подграфы получаются только тогда, когда число транзисторов в подграфе составляет сотни и тысячи [12], в то время как схемы типовых логических элементов содержат десятки транзисторов и число локальных соединений в них меньше числа внешних соединений. Например, схема статического КМОП инвертора локальных соединений вообще не содержит (см. рис. 1). Таким образом, разбиением графа на подграфы (graph partition) шаблоны типовых логических элементов получить практически невозможно.

Для нахождения набора шаблонов подсхем путем поиска часто встречающихся подграфов (frequent subgraph mining – FSM) могут быть использованы известные алгоритмы решения задачи FSM. Но в силу чрезвычайно большой трудоемкости выполнения операции оценки частоты встречаемости подграфа применение известных практических алгоритмов FSM ограничивается поиском подграфов небольшого размера (меньше 10 вершин) – графлетов [13]. Для графов с миллионами вершин удается подсчитать оценку частоты встречаемости графлетов еще меньшего размера [14]. Графлетом является двудольный граф, задающий схему простейшего логического элемента – статического КМОП инвертора, который содержит 8 вершин в доле выводов элементов (см. рис. 1, 2) и четыре вершины в доле цепей. Для нахождения более сложных логических элементов быстродействия известных методов FSM уже недостаточно.

Заключение

Типовыми шаблонами, обнаружение которых является задачей Graph Mining, служат часто встречающихся подграфы и кластеры – подграфы, в которых число внутренних ребер больше числа внешних ребер. Известные практические алгоритмы FSM позволяют определить частоту встречаемости графлетов, сложность которых меньше сложности простейшего логического КМОП вентиля – инвертора. Методы кластеризации графов (graph partitioning methods) выделяют шаблоны со свойством, которым КМОП вентили не обладают. Подграфы, представляющие подсхемы КМОП вентилей, имеют больше внешних ребер, чем внутренних.

Проведенное исследование известных методов Graph Mining показывает, что, несмотря на привлекательность подхода (независимость от технологии построения элементов КМОП схем), известные методы Graph Mining не позволяют построить декомпилятор КМОП схем из транзисторов.

ЛИТЕРАТУРА

1. Logic Gate Recognition in Guardian LVS // Silvaco. URL: https://www.silvaco.com/content/appNotes/iccad/2-003_LogicGates.pdf (accessed: 04.01.2018).
2. Hunt V.D. Reengineering: Leveraging the Power of Integrated Product Development. Wiley, 1993. 283 p.
3. Baker R.J. CMOS Circuit Design, Layout, and Simulation. 3rd ed. Wiley-IEEE Press, 2010. 1214 p.
4. Krasilnikova L.V., Pottosin Yu.V. Partition of a transistor circuit into library modules from a given library // Proc. of the Second Intern. Conf. on Computer-Aided Design of Discrete Devices (CAD DD'97), Minsk, Republic of Belarus, November 12–14, 1997. Minsk : Natl. Acad. Sci. Belarus Inst. Eng. Cybern., 1997. V. 1. P. 94–97.
5. Karypis G., Kumar V. Multilevel Graph Partitioning Schemes // Proc. 24th Intern. Conf. Par. Proc., III. CRC Press, 1995. P. 113–122.
7. Bauer M.A., Sarrafzadeh M., Somezi F. Fundamental CAD algorithms // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2000. V. 19 (12). P. 1449–1475.
6. Schlag S., Henne V., Heuer T., Meyerhenke H., Sanders P., Schulz C. k-way Hypergraph Partitioning via n-Level Recursive Bisection // Proc. 18th Workshop on Algorithm Engineering and Experiments (ALENEX), 2016. P. 53–67.
8. Inokuchi A.; Washio T.; Motoda H. An apriori-based algorithm for mining frequent substructures from graph data // European Symposium Principle of Data Mining and Knowledge Discovery (PKDD'00), 2000. P. 13–23.
9. Sun Z., Wang H., Shao B., Li J. Efficient subgraph matching on billion node graphs // Proc. VLDB Endow. 2012. V. 5, No. 9. P. 788–799.
10. Kuramochi M.; Karypis G. Frequent subgraph discovery // Proc. Int. Conference on Data Mining (ICDM'01). San Jose, CA, 2001. P. 313–320.
11. Slota G.M., Madduri K., Fast approximate subgraph counting and enumeration // Proc. 42nd Int. Conference on Parallel Processing (ICPP). 2013. P. 210–219.
12. Trukhachev A., Ivanova N. Extracting of High-Level Structural Representation from VLSI Circuit Description Using Tangled Logic Structures // Biologically Inspired Cognitive Architectures (BICA) for Young Scientists / A.V. Samsonovich, V.V. Klimov (ed.). Springer Int. Publishing, 2018. P. 318–323.
13. Bressan M., Chierichetti F., Kumar R., Leucci S., Panconesi A. Counting Graphlets: Space vs Time // Proc. of the 10th ACM Int. Conference on Web Search and Data Mining (WSDM 2017). 2017. P. 557–566.
14. Pinar A., Seshadhri C., Vishal V. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs // Proc. of the 26th Int. Conference on World Wide Web (WWW '17). Int. World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland. 2017. P. 1431–1440.

Поступила в редакцию 22 мая 2018 г.

Cheremisinov D.I., Cheremisinova L.D. (2019) SUBCIRCUITS DISCOVERY IN TRANSISTOR LEVEL CMOS CIRCUITS USING GRAPH MINING. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie vychislitelnaja tehnika i informatika* [Tomsk State University Journal of Control and Computer Science]. 48. pp. 74–82

DOI: 10.17223/19988605/48/9

In recent years, there has been an increased interest in the development of data mining algorithms that work on graphs. Such graphs occur naturally in a number of different areas, such as network attack detection, semantic web, behavioral modeling, re-designing of VLSI, analysis of social networks and classification of chemical compounds. The problem of Graph Mining is to discover typical patterns of graph data. Common feature of such patterns are frequent subgraphs. Graph Mining is one of the arms of data mining in which voluminous complex data are represented in the form of graphs and mining is done to infer knowledge from them. Frequent subgraph mining (FSM) is a sub section of graph mining domain which is extensively used for graph classification, building indices and graph clustering purposes. Identification of frequently occurring graphs / subgraphs in a database or in one large graph is a method that can be used for motif detection, social network monitoring, fraud detection, etc.

Electrical circuits consist of elements that are connected to each other by "wires", and the natural formal model of the description of the scheme is a colored undirected bipartite graph. One part consists of the elements' terminals and ports of the circuit, and the other – the connections (nets) between the terminals, i.e., networks (nets) are the "wires". The transformation of the hierarchical circuit of an electronic device into a circuit consisting solely of primitive elements is naturally called a compilation. The reverse process, in the result of which a hierarchical circuit is built up from a flat circuit, is decompilation. To build a hierarchical structural description, you need to select in the circuit as a separate component a set of interconnected transistors, i.e. find subcircuits in the original circuit. After replacing the subcircuits with elements, the description of the circuit becomes two-level. Modern digital circuits contain up to a billion primitive elements at the transistor level. Assuming that subcircuits are frequent subgraphs, you can try to build a decompiler using the Graph Mining methods.

The task FSM has become a popular area of research in the last decade, and so far, the bibliography of this task has hundreds of publications. The complexity of FSM is enormous because of the need to solve many times the problem of subgraph isomorphism. The naive search algorithm for FSM consists of two operations. The first operation searches for all candidate subgraphs for a given

graph *G*, and the second ones calculates the frequency of occurrence of each candidate subgraph. Due to the extremely large computational complexity of the second operation, the well-known practical FSM algorithms are limited to the search for small subgraphs (less than 10 vertices) – graflets.

Our study of the known Graph Mining methods shows that, despite the attractiveness of the approach (independence from the style of primitive elements of circuits), the known methods of Graph Mining do not allow to build decompiler of transistor circuits.

Keywords: clustering of a graph; subgraph counting; subgraph isomorphism; graph color coding; transistor level CMOS circuits.

CHEREMISINOV Dmitry Ivanovich (Candidate of Technical Sciences, Associate Professor, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, Belarus)
E-mail: cher@newman.bas-net.by

CHEREMISINOVA Liudmila Dmitrievna (Doctor of Technical Sciences, Professor, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, Belarus)
E-mail: cld@newman.bas-net.by

REFERENCES

1. Silvaco.com. (n.d.) *Logic Gate Recognition in Guardian LVS – Silvaco*. [Online] Available from: https://www.silvaco.com/content/appNotes/iccad/2-003_LogicGates.pdf (Accessed: 4th January 2018)
2. Hunt, V.D. (1993) *Reengineering: Leveraging the Power of Integrated Product Development*. Wiley.
3. Baker, R.J. (2010) *CMOS Circuit Design, Layout, and Simulation.* 3rd ed. Wiley-IEEE Press.
4. Krasilnikova, L.V. & Pottosin, Yu.V. (1997) Partition of a transistor circuit into library modules from a given library. *Proc. of the Second International Conference on Computer-Aided Design of Discrete Devices (CAD DD'97).* Minsk, Republic of Belarus. November 12–14, 1997. Vol. 1. pp. 94–97.
5. Karypis, G. & Kumar, V. (1995) Multilevel Graph Partitioning Schemes. *Proc. 24th Int. Conf. Par.* Proc., III – CRC Press. pp. 113–122.
6. Schlag, S., Henne, V., Heuer, T., Meyerhenke, H., Sanders, P. & Schulz, C. (2016) K-way hypergraph partitioning via n-level recursive bisection. *Proc. 18th Workshop on Algorithm Engineering and Experiments (ALENEX).* pp. 53–67. DOI: 10.1137/1.9781611974317.5
7. Bauer, M.A., Sarrafzadeh, M. & Somezi, F. (2000) Fundamental CAD algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 19(12). pp. 1449–1475.
8. Inokuchi, A., Washio, T. & Motoda, H. (2000) An apriori-based algorithm for mining frequent substructures from graph data. *European Symposium Principle of Data Mining and Knowledge Discovery (PKDD'00).* pp. 13–23. DOI: 10.1007/3-540-45372-5_2
9. Sun, Z., Wang, H., Shao, B. & Li, J. (2012) Efficient subgraph matching on billion node graphs. *Proc. VLDB Endow*. 5(9). pp. 788–799. DOI: 10.14778/2311906.2311907
10. Kuramochi, M. & Karypis, G. (2001) Frequent subgraph discovery. *Proc. Int. Conference on Data Mining (ICDM'01).* San Jose, CA. pp. 313–320. DOI: 10.1109/ICDM.2001.989534
11. Slota, G.M. & Madduri, K. (2013) Fast approximate subgraph counting and enumeration. *Proc. 42nd Int. Conference on Parallel Processing (ICPP)*. pp. 210–219. DOI: 10.1109/ICPP.2013.30
12. Trukhachev, A. & Ivanova, N. (2018) Extracting of High-Level Structural Representation from VLSI Circuit Description Using Tangled Logic Structures. In: Samsonovich, A.V. & Klimov, V.V. (eds) *Biologically Inspired Cognitive Architectures (BICA) for Young Scientists.* Springer Int. pp. 318–323.
13. Bressan, M., Chierichetti, F., Kumar, R., Leucci, S. & Panconesi, A. (2017) Counting Graphlets: Space vs Time. *Proc. of the 10th ACM Int. Conference on Web Search and Data Mining (WSDM 2017).* pp. 557–566. DOI: 10.1145/3018661.3018732
14. Pinar, A., Seshadhri, C. & Vishal, V. (2017) ESCAPE: Efficiently Counting All 5-Vertex Subgraphs. *Proc. of the 26th International Conference on World Wide Web (WWW '17). Int. World Wide Web Conferences Steering Committee.* Republic and Canton of Geneva, Switzerland. pp. 1431–1440. DOI: 10.1145/3038912.3052597