УДК 519.658, 519.677

Алгоритм декомпозиции электронных схем с ячейками произвольной площади

А.С. Русаков[1], М.В. Шеблаев

[1]Институт проблем проектирования в микроэлектронике РАН, rusakov@inm.ras.ru

Аннотация — В статье предложены эвристики и комбинации эвристик для решения задачи сбалансированного разбиения гиперграфа, моделирующего электронную схему, на подграфы.

Ключевые слова — размещение элементов СБИС, разбиение гиперграфа, декомпозиция электронных схем.

I. Введение

Задача декомпозиции электронной схемы с учетом ограничений часто возникает при логическом или физическом синтезе. В частности, для размещения элементов СБИС может быть использован метод последовательных размещений (mincut), в котором по электронной схеме строится ее гиперграф, который последовательно разбивается на приблизительно равные части так, чтобы оптимизировать число гиперребер, принадлежащих нескольким частям разбиения. Таким образом можно разместить интегральную схему без нарушения плотности и минимизируя длину проводов.

Как известно, задача сбалансированного разбиения графа NP-полная, поэтому для ее решения используются эвристические алгоритмы. Одним из первых был предложен алгоритм Кернигана-Лина [1]. Этот алгоритм итеративно улучшает начальное разбиение графа за конечное число проходов. Базовый принцип попарного обмена элементами подмножеств был в дальнейшем применён в эвристической процедуре разбиения двудольного графа [9]. Сложность алгоритма Кернигана-Лина - $O(|V|^3)$, где $|V|$ - число узлов гиперграфа. В дальнейшем он был вытеснен другими подходами, в частности, алгоритмом пошагового итеративного улучшения - методом Фидуччи-Мэтьюза (FM-алгоритм) [2]. FM-алгоритм вычисляет приближенные решения задачи, близкие к оптимальным, за линейное время.

Как показано в работе [3] для качественного решения задачи поиска минимального сечения необходима кластеризация вершин, делающая алгоритм многоуровневым. На каждом уровне для поиска минимального сечения или его уточнения может использоваться метод FM. В настоящее время известны пакеты программ, использующие идею многоуровневого разбиения графа на подграфы, такие как (h)METIS, Scotch, Zoltan, MLPART и др.

Мы предлагаем новые эвристики FM-метода, которые позволяют улучшить качество, т.е. уменьшить размер полученного сечения гиперграфа. Новые эвристики исследованы и применены в рамках многоуровневого алгоритма разбиения гиперграфа для задачи размещения СБИС [3].

Классический FM-алгоритм применяется для графов с одинаковыми весами узлов. В задаче размещения эти веса соответствуют площади занимаемой ячейкой. В реальных задачах элементы схемы занимают разную площадь или, как в случае размещения ПЛИС или структурированных схем, могут занимать даже разные классы ресурсов. Кроме того, даже если входной граф многоуровневого алгоритма состоит из узлов с одинаковыми весами, кластеризованный граф теряет это свойство. Для таких графов известные алгоритмы либо теряют в качестве разбиения, либо требуют $O(|V|^2)$ операций. В отличие от других работ, посвященных этой проблеме (например [5]), мы предлагаем модифицировать структуру данных поиска оптимальных движений. Надо отметить, что один проход метода FM - принципиально последовательный алгоритм. Для ускорения за счет параллельных вычислений в методе FM одновременно и параллельно решаются задачи оптимизации сечения с разными начальными разбиениями. Поэтому задача ускорения одной итерации FM остается актуальной. Подходы к распараллеливанию многоуровневого алгоритма разбиения известны и не являются предметом данной статьи.

В четвертой части работы предложено использовать комбинацию эвристик выбора из элементов корзины движений с одинаковой стоимостью: LIFO и LIFO*[7], что также позволяет усовершенствовать поведение алгоритма на тестовых схемах.

II. FM-алгоритм

Пусть $H(V,E)$ - гиперграф с множеством вершин V и множеством гиперребер E. Пусть для каждого ребра $e \in E$ определена весовая функция с целыми значениями $w(e)>0$, а для каждой вершины гиперграфа $v \in V$ определена весовая функция $w(v)$. Задача минимизации разбиения гиперграфа, удовлетворяющего условию баланса, состоит в построении таких непересекающихся подмножеств

$A \in V, B \in V, A \bigcup B = V, A \bigcap B = 0$, что для ребер e, принадлежащих разрезу $\Psi = ((u,v) \in E : u \in A, v \in b)$, минимизируется функция стоимости:

$$CutCost = \sum_{e_i \in \Psi} w(e_i) \qquad (1)$$

при условии выполнения балансовых ограничений:

$$1 - \varepsilon \le \frac{\sum_{u \in A} w(u)}{\sum_{v \in B} w(v)} \le 1 + \varepsilon \ . \qquad (2)$$

В другой постановке, часто использующейся, например, при решении практических задач размещения электронных схем, условие баланса может быть записано следующим образом:

$$\sum_{u \in A} w(u) \le S_A \quad , \qquad (3)$$

$$\sum_{v \in B} w(v) \le S_B \quad , \qquad (4)$$

где S_A, S_B – допустимые суммы весов вершин в подобластях. Также может быть полезным введение терминальных вершин гиперграфа t, фиксированных в подобласти и имеющих нулевой вес. Гиперребра могут содержать не больше одной терминальной вершины.

Алгоритм 1 FM-алгоритм

```
function FM Pass (graph G = (V, E), max balance maxbal)
    P_saved ← V_1, V_2                    ▷ Начальное разбиение V = V_1 ∪ V_2
    CutCost_saved ← CutCost(P)
    Вычислить Δ(v_i) для каждой v_i ∈ V
    repeat
        ▷ Выбрать незафиксированную вершину v ∈ V с максимальной Δ(v),
        v ← GetBestMove()
        if v ∈ V_1 then
            V_1 ← V_1 − v; V_2 ← V_2 + v
        else
            V_2 ← V_2 − v; V_1 ← V_1 + v
        end if
        for v', связанных ребрами с v do
            if v' не зафиксирована then
                Вычислить Δ(v')
                                  ▷ Обновить стоимость ячейки в корзине
                UpdateGainCell(Δ(v'))
            end if
        end for
        Зафиксировать v
        P ← V_1, V_2
        if CutCost_saved > CutCost(V_1, V_2) then
            CutCost_saved ← CutCost(V_1, V_2)
            P_saved ← V_1, V_2
        end if
    until все вершины незафиксированы
    Снять фиксирование со всех вершин
    return P_saved
end function
```

Рис. 1. Псевдокод одной итерации FM-алгоритма

Формулировки (1)-(4) позволяют решать широкий круг задач, возникающих при логическом или физическом синтезе схем. Если решается задача размещения, то данные постановки позволяют решать не только задачу оптимизации длины проводов, но и задачи минимизации временных задержек и обеспечения трассируемости схемы на кристалле (см., например, [4, 10]). При использовании алгоритма сбалансированного разбиения гиперграфа в методе последовательных разбиений веса вершин гиперграфа моделируют площадь одной ячейки схемы или, в случае иерархических подходов, площадь кластера из нескольких ячеек. Гиперребра моделируют связи между ячейками, а их вес определяется вкладом ребра в длину межсоединений и критичностью путей, проходящих через гиперребро. Существуют разные варианты методов размещения, в которых используется подход последовательного разбиения графа схемы, но решение NP-полной задачи (1)-(4) определяет качество полученного размещения по выбранной метрике и быстродействие программной реализации.

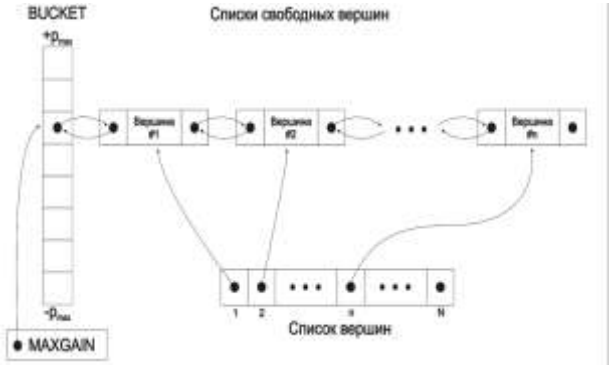

Рис. 2. Структура корзины движений. MAXGAIN указывает на лист из узлов графа с максимальной стоимостью

Классический FM-алгоритм на входе имеет первоначальное разбиение, например случайное, и состоит из последовательных итераций. Каждая итерация начинается с присвоения всем вершинам гиперграфа стоимости - значения, на которое уменьшится размер сечения при передвижении этой вершины в другую подобласть. Далее на каждом шаге узел графа с максимальной стоимостью перемещается из одной подобласти в другую, перемещенный узел фиксируется на данной итерации, и стоимости всех соседних узлов пересчитываются. В конце итерации выбирается последовательность шагов, максимизирующая суммарную стоимость или, что тоже самое, минимизирующая размер разреза. Итерации повторяются заданное число раз или до тех пор, пока не наступает насыщение алгоритма. Псевдокод FM-алгоритма представлен на рис. 1.

Ключевым элементом этого алгоритма является "корзина стоимостей" (gain bucket) - структура данных для хранения и быстрого пересчета стоимости локальных перемещений вершин между A и B, позволяющая найти локальное перемещение с наибольшей выгодой со сложностью O(1) [2] (см. рис. 2).

Результат применения эвристики существенно зависит от входного разбиения, поэтому из множества решений, полученных для некоторого набора первоначальных разбиений, в результате выбирается лучшее. В наших экспериментах такой набор состоял из 35 вариантов для каждого из используемых методов кластеризации. Такое большое число вариантов выбрано на основе анализа поведения алгоритма на наиболее важных схемах.

Мы используем модификации известных алгоритмов кластеризации HEC, MHEC [3], HEM, FC [5]. При сравнении алгоритмов кластеризации мы обнаружили, что алгоритмы HEC и MHEC достигают насыщения после нескольких итераций и размерность кластеризованного гиперграфа оказывается слишком большой. Поиск минимального сечения такого графа оказывается неэффективным. Чтобы улучшить вычислительную эффективность алгоритма, к гиперграфу, полученному из HEC или MHEC, мы применяем метод кластеризации FC, который успешно сжимает гиперграф еще в несколько раз. В нашем многоуровневом алгоритме мы используем несколько алгоритмов кластеризации и выбираем лучшее решение [4].

III. Оптимизация корзины движений FM-алгоритма

Корзина движения позволяет найти "оптимальную" ячейку за время O(1). Если в графе много вершин с большим весом, эти вершины оказываются сверху корзины, но их движение нарушит условие баланса и поиск сбалансированной оптимальной ячейки может потребовать количество операций пропорциональное количеству “тяжелых” вершин. Существуют разные стратегии обработки таких ячеек. Самая простая - удалять несбалансированные ячейки из корзины как только они встречаются, такая ячейка двигаться на данной итерации FM-алгоритма не будет. Другая стратегия - не удалять узлы графа, которые на данном шаге не сбалансированы, а оставлять их в корзине и передвигать в другую подобласть, как только условие баланса будет достигнуто и узел будет узлом максимальной стоимости. Псевдокод этих стратегий приведен на рис. 3. Эта модификация ухудшает быстродействие алгоритма, но в наших экспериментах мы обнаружили, что выбрасывание из корзины временно несбалансированных движений ячеек сильно увеличивает размер разреза. В таблицах 1-2 мы приводим время и качество решения двух вариантов FM-алгоритма на тестах [8]. Видно, что игнорирование временно несбалансированных ячеек приводит к росту размера полученного сечения гиперграфа, причем даже разумное увеличение числа проходов или числа начальных решений не позволяет приблизиться к решению, полученному при стратегии, в которой из корзины не выбрасываются несбалансированные ячейки.

Алгоритм 2 Классический быстрый поиск в корзине движений

```
function GetBestMove
    cell ← maxGainCell      > поиск следующей сбалансированной ячейки начинается с узла с наибольшей ценой в корзине движений
    while cell != NULL do
        if IsBalancedMove(cell) then
            RemoveCellFromBucket(cell)
            return cell
        end if
                > ячейка удаляется из корзины как только она рассмотрена
        RemoveCellFromBucket(cell)
        cell ← cell->next
    end while
    return NULL
end function
```

Алгоритм 3 Медленный поиск в корзине движений

```
function GetBestMove
    cell ← maxGainCell
    while cell != NULL do
        if IsBalancedMove(cell) then
            cell ← cell->next
                > ячейка удаляется из корзины только после ее движения
            RemoveCellFromBucket(cell)
            return cell
        end if
    end while
    return NULL
end function
```

Рис. 3. Псевдокоды поиска “оптимальной” сбалансированной ячейки в корзине движений

Алгоритм 4 Новый поиск в корзине движений

```
function GetBestMove
    cell ← maxGainCell

    if firstBalanced then
        > начинаем поиск с предположительно сбалансированной ячейки
        cell ← firstBalanced

    end if
    while cell != NULL do
        if IsBalancedMove(cell) then
            RemoveCellFromBucket(cell)
                    > сохраняем указатель на ячейку следующую за сбалансированной
            firstBalanced ← cell->next
            return cell
        end if
    end while
    firstBalanced ← NULL
    return NULL
end function
```

Рис. 4. Псевдокоды поиска “оптимальной” сбалансированной ячейки в предложенной корзине движений. Как только узел найден, указатель *firstBalanced* показывает на следующий узел. Следующий поиск начинается с *firstBalanced*

Для того, чтобы приблизиться к качеству медленной корзины движений, мы предложили модификацию с рестартами. В структуру данных добавлен указатель, который указывает либо на ноль, либо на следующую вершину. Изначально он инициализируется нулем. Как только мы находим узел графа с максимальной стоимостью и выбираем ее для движения, указатель показывает на следующий элемент в корзине движений. Если указатель равен нулю, то поиск движения начнется с вершины максимальной стоимости, как в случае “медленной” корзины, иначе поиск сбалансированной ячейки начнется с ячейки маркированной указателем. Если после нескольких ходов алгоритма и пересчета стоимостей узлов графа какая-либо ячейка из корзины

приобретает стоимость большую, чем ячейка, на которую указывает наш указатель на сбалансированный элемент, мы его обнуляем, т.е. делаем «рестарт». Таким образом, мы уменьшаем среднюю стоимость поиска лучшей ячейки для движения в другую подобласть, так как не нужно пробегать через длинный список ячеек с большим весом (или площадью), оказавшихся в верху корзины. В худшем случае такой алгоритм поиска все равно потребует O(|*V*|) операций, а реализация FM-алгоритма $O(|V|^2)$, но можно ожидать, что следующая ячейка для движения будет найдена за O(1). Важно отметить, что такая модификация корзины за счет своей простоты практически не ухудшает сложность поиска лучшей вершины и в том случае, когда все движения сбалансированы.

Экспериментальное сравнение новой структуры корзины движений для плоского и многоуровневого алгоритма приведено в таблицах 1-2. Для анализа поведения нескольких алгоритмов на множестве тестов мы ввели коэффициент качества. Для каждого из *N* тестов мы находим лучшее значение сечения *BestCut = min(Cut)*. Тогда качество для каждого из алгоритмов на заданном тесте равно: $q = BestCut / Cut$. Качество алгоритма на множестве тестов по сравнению с другими: $Q = \sum q$.

Результаты показывают, что применение корзины с рестартом позволяет существенно ускорить FM по сравнению с "медленной", но точной корзиной, при этом размер сечения увеличивается незначительно. В многоуровневой реализации результат заметен меньше, так как FM алгоритм применяется к кластеризованному графу меньшей размерности.

IV. Эвристики выбора из вершин с одинаковой стоимостью

В FM-алгоритме [2] для каждого движения между подобластями выбирается вершина с наибольшей стоимостью. В том случае, когда таких вершин для очередного хода несколько, необходимо выбрать одну из них. Для этого выбора не существует доказанной оптимальной стратегии, но экспериментально показано, что разные стратегии существенно влияют на результат работы алгоритма. В работе [6] рассматривались следующие стратегии локального выбора:

- LIFO, при которой элемент, помещаемый в соответствующую корзину движений последним, выбирался первым;
- FIFO, при которой элемент, пришедший первым, первым же и выбирался;
- Random, при которой реализовывался случайный выбор элемента среди прочих кандидатов.

Таблица 1

Размер сечения и время работы FM-алгоритма, $\varepsilon = 0.02$

	Slow Bucket			Classic bucket			New bucket		
Test Name	Cut	Time	Quality	Cut	time	Quality	Cut	time	Quality
ibm01	303	1	1	328	1	0,92	393	2	0,77
ibm02	446	4	0,84	8	2	0,49	373	3	1
ibm03	1734	4	0,85	2513	3	0,59	1558	3	0,95
ibm04	1042	7	0,94	1423	4	0,69	1093	5	0,9
ibm05	2010	7	0,93	3201	4	0,59	1878	5	1
ibm06	1076	9	0,96	2752	4	0,38	1173	7	0,88
ibm07	1505	16	0,89	1574	9	0,85	1332	12	1
ibm08	1937	35	1	5242	10	0,37	2491	17	0,78
ibm09	1246	20	1	3806	12	0,33	1457	15	0,86
ibm10	2638	49	0,85	4204	17	0,53	2230	24	1
ibm11	2612	32	0,82	5757	18	0,37	2136	22	1
ibm12	3722	73	1	8350	18	0,45	4741	30	0,79
ibm13	2188	39	0,76	3042	22	0,54	2036	28	0,81
ibm14	3855	69	0,82	7528	43	0,42	4561	54	0,7
ibm15	4948	168	0,87	9542	51	0,45	5246	75	0,82
ibm16	3683	179	1	10828	57	0,34	4356	82	0,85
ibm17	3285	117	1	13236	60	0,25	5434	79	0,6
ibm18	3345	191	1	9522	63	0,35	3758	102	0,89
Score		1020	16,52		398	8,89		565	15,58

Таблица 2

Размер сечения и время работы многоуровневого алгоритма разбиения гиперграфа с разными корзинами FM-алгоритма $\varepsilon = 0.02$

	Slow bucket			New bucket			Classic bucket		
Test Name	Cut	Time	Quality	Cut	Time	Quality	Cut	Time	Quality
ibm01	219	4	1	245	4	0,89	336	3	0,65
ibm02	269	5	0,99	269	5	0,99	266	5	1
ibm03	751	10	1	791	8	0,95	911	7	0,82
ibm04	535	9	0,96	515	8	1	513	7	1
ibm05	1733	11	1	1732	12	1	1725	10	1
ibm06	585	11	0,89	535	11	0,98	788	11	0,66
ibm07	802	22	1	833	17	0,96	820	15	0,98
ibm08	1190	32	1	1190	17	1	1187	16	1
ibm09	533	18	1	535	17	1	535	16	1
ibm10	1119	39	0,97	1084	17	1	1216	17	0,89
ibm11	874	34	0,97	868	26	0,97	844	27	1
ibm12	2195	64	0,95	2185	33	0,95	2084	26	1
ibm13	1073	62	0,95	1111	43	0,92	1024	43	1
ibm14	1938	100	1	1955	77	0,99	1960	74	0,99
ibm15	2567	200	1	2653	92	0,97	2709	93	0,95
ibm16	1788	151	1	1805	92	0,99	1868	104	0,96
ibm17	2296	150	1	2362	158	0,97	2648	92	0,87
ibm18	1943	49	1	2232	40	0,87	2057	23	0,94
Score		971	17,67		677	17,4		589	16,71

Таблица 3

Сравнение стратегий локального выбора. Приведен размер разреза полученного многоуровневым алгоритмом разбиения гиперграфа, $\varepsilon = 0.02$

Test Name	**LIFO**	**LIFO***		**LIFO + LIFO***	
ibm01	262	252	3,97%	245	6,49%
ibm02	269	346	-22,25%	269	0,00%
ibm03	826	850	-2,82%	822	0,48%
ibm04	537	539	-0,37%	537	0,00%
ibm05	1731	1726	0,29%	1731	0,00%
ibm06	596	718	-16,99%	596	0,00%
ibm07	820	820	0,00%	820	0,00%
ibm08	1190	1190	0,00%	1190	0,00%
ibm09	535	535	0,00%	535	0,00%
ibm10	1088	1011	7,62%	1088	0,00%
ibm11	877	820	6,95%	877	0,00%
ibm12	2185	2193	-0,36%	2185	0,00%
ibm13	1170	1175	-0,43%	1166	0,34%
ibm14	1955	1969	-0,71%	1955	0,00%
ibm15	2488	2639	-5,72%	2488	0,00%
ibm16	1805	1824	-1,04%	1789	0,89%
ibm17	2378	2403	-1,04%	2378	0,00%
ibm18	1959	1931	1,45%	1959	0,00%
			-31,47%		8,20%

В работах [5-6] экспериментально показано, что для этих случаев стратегия LIFO обеспечивает лучшее качество итогового разбиения. Объяснение, предложенное авторами, состояло в том, что в этой стратегии вершины с сильной связностью (ядра кластеров) будут перемещаться последовательно и с большим приоритетом.

В работе [7] авторы сделали наблюдение о том, что большее влияние может иметь перемещение вершин, связанных с недавно перемещенными вершинами. В таком случае предлагается использовать стратегию, разделяющую изменения стоимостей при перемещении на три типа:

- тип A, при котором стоимость увеличивается,
- тип B, при котором стоимость уменьшается,
- тип C, при котором стоимость остается неизменной.

Перемещения типа A в духе наблюдения [6] для алгоритма FM должны поощряться и обрабатываться в первую очередь, перемещения типа B, напротив, должны влиять на последующие перемещения как можно меньше, перемещения типа C никак не изменяют общей картины.

В соответствии с этим наблюдением вершины при перемещении типа A должны помещаться в "активную зону" корзины выгоды, из которой они будут извлечены вскоре, т.е. в начало списка; вершины типа B - в конец списка, чтобы не "отравлять" последующие перемещения. Вершины типа C остаются на своем месте. Такая стратегия получила название LIFO*.

Мы реализовали и сравнили разные стратегии, включая LIFO* на примерах гиперграфов, полученных из электронных схем. Мы получили в среднем результаты хуже по сравнению со стандартной стратегией LIFO. В тоже время на некоторых гиперграфах новая эвристика работает лучше. Для того, чтобы улучшить результаты нашего алгоритма, мы реализовали комбинацию эвристик LIFO* и LIFO. На каждой третьей итерации FM мы используем LIFO*. Мы предполагаем, что комбинация эвристик эффективней позволяет выйти из локального минимума FM. В таблице 3 показано, что наше предложение позволяет добиться роста качества разбиения на 3 задачах и идентичных результатов на остальных. Время работы алгоритма не изменилось и в таблице не приведено.

V. Заключение

В работе предложены модификации эвристик алгоритма FM для задачи разбиения гиперграфа. Использование комбинация известных эвристик LIFO* и LIFO для выбора ячейки графа позволило улучшить качество алгоритма.

Мы показали, что известные алгоритмы поиска "наилучшей" вершины в гиперграфе с произвольными весами вершин дают решение, далекое от оптимального. Предложенный алгоритм поиска в корзине движений позволяет получать сбалансированное разбиение гиперграфа с меньшим размером разреза без заметной потери быстродействия по сравнению с известными пакетами.

Литература

[1] Kernighan B.W., Lin S. An efficient heuristic procedure for partitioning graphs // Bell Syst. Tech. J. 1970.

[2] Fiduccia C.M., Mattheyses R.M. A linear time heuristic for improving network partitions // Proc. ACM/IEEE Design Automation Conf. 1982. P. 175-181.

[3] Karypis G., Aggarwal R., Kumar V., Shekhar S. Multilevel hypergraph partitioning: application in VLSI domain // Proceeding of Design Automation Conference. ACM, New York. 1997. P. 526-529.

[4] Андреев А.Е., Пависич И., Русаков А.С. Алгоритм глобального размещения структурированных заказных схем // V Всероссийская научно-техническая конференция «Проблемы разработки перспективных микро- и наноэлектронных систем - 2012». Сб. трудов / под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН. С. 231-236.

[5] Papa D.A. and Markov I. Hypergraph Partitioning and Clustering // in Approximation Algorithms and Metaheuristics, CRC Press. 2007. P. 61-1- 61-19.

[6] Hagen, On implementation choices for iterative improvement partitioning algorithms // IEEE Trans. 1997. P. 1–12.

[7] Yourim Yoon, Yong-Hyuk YH Kim. New Bucket Managements in Iterative Improvement Partitioning Algorithms // Appl. Math. 2013. Vol. 10. № 1. P. 37-57.

[8] Charles J. Alpert, The ISPD-98 Circuit Benchmark Suite 1998 Proc. ACM/IEEE Int'l Symp. Physical Design (ISPD 99), ACM.

[9] Русаков С.Г., Святский А.Б. Метод автоматического разбиения на подсхемы для машинного анализа больших интегральных схем // Управляющие системы и машины. Киев «Наукова думка», 1975. С. 116-119.

[10] Caldwell A., Kanhg A., Markov I. Can recursive bisection produce routable placements? // 37th IEEE/ACM Design Automation Conference. ACM, New York. 2000. P. 477-482.