УДК 004.8.032.26
Кочурко В. А.
Научный руководитель: к.т.н., доцент Шуть В.Н.

НЕКОТОРЫЕ АЛГОРИТМЫ ГЕНЕРАЦИИ РЕГУЛЯРНЫХ ГРАФОВ

Введение. Задача создания полного списка регулярных графов, вместе с задачей распознавания изоморфных графов, является одной из старейших задач конструктивной комбинаторики [1]. Помимо простого научного интереса, генерация таких графов имеет также и практическое значение.

Регулярные графы применяются в разнообразных областях, например, в peer-to-peer сетях [2], некоторые из которых базируются на связных регулярных неориентированных графах, при идентификации молекул веществ, при разработке баз знаний интеллектуальных систем. В связи с повсеместным распространением информационных систем и их объединением в локальные и глобальные компьютерные сети наиболее перспективным является использование таких графов при создании оптимальной топологии таких сетей [3], а именно – при необходимости наилучшим образом связать большое количество однородных элементов.

В данной работе анализируются четыре алгоритма, основанные на идеях построения дерева, прямого произведения множеств, циклического перебора и избирательной вставки. Анализ базируется на программной реализации данных алгоритмов, выполненной студентами группы ИИ-4 Брестского государственного технического университета под руководством доцента кафедры ИИТ Шутя В. Н.

Постановка задачи. Регулярный граф K_N^M – граф, у которого степени всех вершин равны между собой и где ***N*** – количество этих вершин, а ***M*** – степени вершин. Изоморфные графы – графы, для которых выполняются следующие условия: каждой вершине одного графа можно однозначно поставить в соответствие вершину другого графа так, что если любые две вершины первого графа (не) смежны, то соответствующие вершины второго графа также (не) смежны.

Задача состоит в генерации всех неизоморфных между собой регулярных графов с заданными ***M*** и ***N***.

Алгоритм дерева. Алгоритм генерации графов методом дерева относится с категории точных алгоритмов.

Первоначально строится так называемое кольцо – граф K_n^2.

На базе графа K_n^2 строится граф K_n^3, путем установки в кольцо ребер. Для того, чтобы иметь все возможные комбинации, воспользуемся древовидным отображением возможных ребер.

Изначально мы выбираем на кольце одну вершину. Пусть это будет вершина x1. Она будет вершиной дерева отображения комбинаций ребер. Эту вершину мы можем связать ребром со следующими вершинами: x3, x4, x5... xn-1.

На первом шаге строится ребро {x1, xi}, где $i = 3...\frac{n}{2}$. Для сокращения числа ветвлений ребра из вершины x1 устанавливаются ребра лишь на половину вершин для избежания симметрии в замкнутом контуре.

На последующих шагах определяются свободные вершины (вершины, степени которых меньше M), и строятся новые ребра вплоть до установки последнего. Тогда число шагов для генерации одного графа:

$$g_c = \frac{MN}{2} - N, \qquad (1)$$

где это также число ребер для установки [3].

Трудоемкость данного алгоритма:

$$(N-2)! \Rightarrow O(N!) \quad (2)$$

Количество используемой памяти:

$$1+(n-3)+(n-3)(n+2)+(n-3)(n+2)(n-3)+\ldots \quad (3)$$

Алгоритм прямого произведения множеств. В данном алгоритме используется свойство многих регулярных графов – наличие гамильтонова контура и представление такого графа в виде «обод-спица». Первоначально строится граф K_n^2. Затем на его базе строится специальная матрица внутренних ребер графа K_n^m. Очевидно, что существует множество комбинаций установки ребер графа K_n^m в обод. Поэтому в первом столбце матрицы внутренних ребер графа K_n^m строятся все возможные ребра из вершины x_1. Далее во втором столбце находятся все возможные варианты построения внутренних ребер графа K_n^m выходящие из вершины x_2, и т.д. Каждый столбец-множество обозначается буквой $A_1, A_2, \ldots A_{n-2}$.

Далее, на базе матрицы внутренних ребер строим прямое произведение множеств $A_1, A_2, \ldots A_{n-2}$. При построении прямого произведения множеств определяем, возможно ли одновременное существование ребер графа, задаваемых элементами множеств. Если это возможно, то результат произведения равен единице, иначе прямое произведение будет равно нулю. При этом образуются пары множеств $(A_1 A_2), (A_1 A_3), \ldots (A_{n-1} A_{n-2})$. Каждое такое прямое произведение множеств позволяет построить два внутренних ребра графа K_N^M.

На последующих шагах алгоритма производим прямое произведение полученных множеств и исходных множеств в случае нечетного количества внутренних ребер графа, либо прямое произведение полученных множеств между собой в случае четного количества внутренних ребер графа. Количество ребер можно определить по формуле (1).

Далее, при построении прямого произведения множеств, определяем, возможно ли одновременное существование всех ребер графа, полученных в каждом из множеств. Если это возможно, то результат произведения равен 1, иначе – 0.Алгоритм выполняется до момента достижения необходимого количества ребер в графе [3].

Трудоемкость данного алгоритма:

$$\frac{MN*(N-2)!}{M!*(N-2-M)!} \Rightarrow O(N^{M+1}) \quad (4)$$

Количество используемой памяти:

$$\frac{N^{M-1}*(N-2)!}{M!*(N-2-M)!} \quad (5)$$

Алгоритм циклического перебора.

Этот алгоритм также основан на принципе «обод-спица».

Изначально строится обод (граф K_n^2).

Алгоритм работает с двумя вершинами, относительно которых вычисляет возможность либо невозможность постановки на них ребра. На первом шаге эти вершины совпадают. Ребро поставить нельзя.

Тогда вторая вершина смещается на единицу. Ребро {x1, x2} уже существует, поэтому на третьем шаге вершина смещается еще на единицу. Постановка ребра {x1, x3} допустима. После того, как ребро устанавливается, рекурсивно вызывается та же процедура, но с увеличенной на единицу первой вершиной и уменьшенным на единицу количеством свободных ребер. При этом вторая вершина возвращается на начало и опять последовательно проходит все вершины, пока не найдет место, где возможна установка ребра. На шаге 7 видно, что сложившаяся расстановка неразрешима, т.к. остается еще

одно свободное ребро, которое поставить некуда. В этом случае алгоритм выходит на один шаг из рекурсии, что ведет к отмене последнего проставленного ребра (см. шаг 14) и нахождению другого места для постановки ребра. На шаге 49 был построен один из графов. Этот граф запоминается, первая точка переходит на следующую позицию, вторая ставится на начальную вершину и алгоритм повторяется, пока все вершины контура не пройдут алгоритм в роли входных вершин.

Трудоемкость данного алгоритма:

$$\frac{(N-3)!}{(M-1)!*(N-3-M)!} \Rightarrow O(N^M) \qquad (6)$$

Количество используемой памяти:

$$\frac{N^3*(N-3)!}{M!*(N-3-M)!} \qquad (7)$$

Алгоритм двух вершин. Ещё одним из алгоритмов генерации регулярных графов является алгоритм двух вершин. Его качественное отличие от предыдущих алгоритмов заключается в отказе от прочной связи алгоритма с графами типа «обод-спица».

Допустим, что существует исходный граф K_n^m, из которого необходимо получить граф K_{n+2}^m. Тогда возьмем ребро {xi; xj} этого графа, и установим на нем две вершины xn+1 и xn+2. Эти вершины будут смежны между собой и смежны вершинам xi и xj.

Вторым шагом является удаление ребер. Поочередно перебираются все ребра, неинцидентные вершинам xi и xj, и в каждом случае такое ребро {xi1; xj1} удаляется. В таком случае появляется возможность установки новых ребер, причем двумя способами: {xi; xj1} и {xi1; x1}, или же {xi1; xi} и {xj; xj1}. Полный перебор всех ребер в качестве входных завершает данный алгоритм.

Трудоемкость данного алгоритма:

$$\frac{(M-2)N(N-1)}{2} \Rightarrow O(MN^2) \qquad (8)$$

Количество используемой памяти:

$$2^{M-3}*M*N*\prod_{K=1}^{M-2}\left(\frac{M*N}{2}+(1-2M)*K\right) \qquad (9)$$

Заключение. Алгоритмы дерева и прямого произведения множеств просты в понимании и реализации, но очень ресурсоемки и трудоемки, а также не генерируют полный список графов, что делает данные методы непригодными для использования в неучебных целях.

Алгоритм циклического перебора некоторым образом похож на алгоритм прямого произведения множеств, однако менее трудоемок. Отсутствие негамильтоновых графов среди сгенерированных данным алгоритмом также ставит под сомнение выгодность этого алгоритма.

И кардинально отличающийся от вышеописанных алгоритм двух вершин имеет наименьшую трудоемкость и возможность генерировать все типы графов, но требует больших ресурсов памяти и нуждается в исходном графе, затраты на генерацию которого не вошли в расчеты трудоемкости и ресурсоемкости. Однако этот алгоритм наиболее привлекателен среди данных четырех.

ЛИТЕРАТУРА

1. Gunnar Brinkmann: Fast Generation of Cubic graphs. Journal of Graph Theory Vol. 23, No. 2 (1996), 139 – 149.

2. Ali Shokoufandeh: Peer-to-peer networks based on random transformations of connected regular undirected graphs. Journal of Algebraic Combinatorics: An International Journal. Volume 19, Issue 3 (May 2004), 257 – 272.

3. Шуть В. Н., Свирский В. М., Муравьев Г. Л., Анфилец С. В. Генерация регулярных связных графов // Вестник БрГТУ. – 2006. – № 5. – С. 44-47.