1. Гмурман В.Е. Теория вероятностей и математическая статистика. – М.: Высш. школа, 1977.
2. Бобнев М.П. Генерирование случайных сигналов. Изд. 2-е перераб. и доп. – М.: Энергия, 1971.
3. Гундарь К.Ю., Гундарь А.Ю., Янишевский Д.А., Защита информации в компьютерных системах. – К.: Корнейчук, 2000. – 152с., ил.
4. Кнут Д. Искусство программирования для ЭВМ. Пер. с англ. В 3-х т. – М.: Мир, 1977. – Т. 2. Получисленные алгоритмы. – 724 с.
5. Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях / Под ред. В.Ф. Шаньгина: 2-е изд., перераб. и доп. – М.: Радио и связь, 2001. – 376 с.
6. Вероятностные распределения и математическая статистика // Сб. ст. АН УзССР, Ин-т математики им. В. И. Романовского; [Редкол.: С.Х. Сираджинов (отв. ред.) и др.]. – Ташкент, Фан, 1986. – 500 с.
7. Иванов М.А. Криптографические методы защиты информации в компьютерных системах и сетях. – М.: КУДИЦ - ОБРАЗ, 2001. – 368 с.
8. Гарасимчук О.І., Максимович В.М. Алгоритм формування пуассонівського імпульсного потоку // Вісник Нац. ун-ту "Львівська політехніка". – 2003. – №475. – С. 21–25.

Р.Мельник, О.Лучковський
Національний університет "Львівська політехніка"

УДК 621.38

ДОСЛІДЖЕННЯ АЛГОРИТМУ МІНІМІЗАЦІЇ ЛОГІЧНИХ ФУНКЦІЙ КЛАСТЕРИЗАЦІЄЮ

Розглянуто підхід до розв'язування задачі декомпозиції логічних функцій на основі нечіткого дерева згортання та нечіткої кластеризації. Описано особливості алгоритмів побудови дерева згортання вершин графів та аналітичних виразів логічних функцій для виділення незалежних змінних.

The approach for the logical functions decomposition on a base of fuzzy reduction tree or fuzzy clusterization is considered. The properties of the optimal reduction tree method for graphs and logical sentences are described.

Вступ

Для мінімізації логічних функцій відомі ряд методів, що базуються на картах Карно чи перетвореннях таблиць за методами Мак-Квейна чи Мак-Класкі. У цій роботі

запропоновано використати поняття нечіткої кластеризації для знаходження складових мінімальних логічних функцій як об'єктів декомпозиції виразу функції. Наявність характеристик суміжності елементів та неврахування нульових елементів зменшує обчислювальну складність алгоритму .

1. Мінімізація логічних функцій

Графічною формою подання логічних функцій є куби (тривимірний випадок) та гіперкуби для розмірності задач, більшої від трьох. Кожній вершині куба у тривимірному просторі відповідає значення змінних, наприклад, $x_1x_2x_3$ (101). Вони утворюють так званий кубічний комплекс: одиничні вершини – 0-куби, ребра з інцидентними одиничними вершинами – 1-куби. 2-куб – це сукупність двох 1-кубів без спільних вершин. Надалі p-куби будемо називати p-підграфами. У загальному випадку зазначені (p-підграфи) містять 2^p вершин графа.

Графи є зручними моделями для спрощення логічних функцій, наприклад, для подання їх у диз'юнктивній нормальній формі (ДНФ) з мінімальним числом кон'юнкцій. Для цього необхідно в графі всі вершини (а тут ми маємо на увазі тільки ненульові вершини) покрити p-підграфами. 0-підграф зображає кон'юнкцію всіх n змінних, 1-підграфи зображають кон'юнкцію n-1 змінних, 2-підграф – кон'юнкція n-2 змінних, вісім об'єднаних вершин (2 четвірки) – це 3-підграф і кон'юнкція з n-3 змінних і т.д. Якщо вибрано найбільші p-підграфи та використано найменшу їх кількість, то ДНФ буде найкоротшою.

Аналіз процесу мінімізації логічних функцій показує, що він зводиться до знаходження найбільших покриттів з 2^k сусідніх вершин графа. Необхідно, щоб кожна заповнена комірка входила в деяке покриття. Імпліканта, що відповідає покриттю вершин графа, містить символи тих змінних, значення яких збігаються у всіх об'єднаних комірках.

До розв'язування задачі мінімізації застосуємо алгоритм побудови так званого нечіткого дерева згортання [1], в якому допускається присутність однієї вершини дерева у різних вершинах піддерев. В практичній реалізації прискорення алгоритму полягає в обчисленні критеріальної функції F тільки для суміжних пар, що зменшить обчислювальні затрати.

Покриття утворюється вершинами верхнього рівня дерева згортання, тобто такими, які вже не можуть брати участі у згортанні. Вони можуть містити вершини різних рівнів дерева згортання включно з базовими (вершинами графа). Для оцінки якості покриття (кількості символів у функції) приймається сума рівнів вершин, які утворюють покриття. Чим більше рівнів об'єднання вершин – тим менше символів у виразі логічної функції. Кількість фрагментів розбиття відповідає кількості імплікант у виразі логічної функції.

Пошук покриттів – це перебір комбінацій утворених фрагментів розбиття. Вхідними даними для знаходження мінімального виразу логічної функції є фрагменти верхнього рівня дерева згортання типу $M_{l,s}$ – підмножини з s вершин k-го рівня. Кількість цих підмножин залежить від вхідної логічної функції та характеру дерева згортання. На основі посортованих підмножин згідно з потужностями (кількістю базових вершин, рівнем розташування) виконується процедура покриття.

До раніше вибраних підмножин M_i, ..., M_j додається підмножина M_k, якщо виконується умова:

$$| M_i \cup ... \cup M_j \cup M_k | \geq 1 ,$$

тобто при умові внесення в покриття хоча б однієї базової вершини, якої немає в об'єднанні попередніх підмножин.

2. Програмна реалізація

Подавати *p*-підграфи зручно за допомогою двох наборів бітів: один для зберігання значень змінних, інший – для входження змінних. Наприклад, щоб подати 1-підграф [101–], потрібно використати такі набори бітів: [1010] і [1110] (перший – для значень, другий – для входжень); 2-підграф [0–1–1] може бути розділений на такі набори бітів: [00101] і [10101] відповідно. Щоб використовувати таке подання у програмі, необхідно оголосити два класи: один для зберігання і виконання операцій над наборами бітів (*BitSet*), а інший – для подання *p*-підграфів і маніпуляцій над ними (*Node*). Призначення методів класу *BitSet* (рис.1) є таким:

– *size*() – повертає кількість бітів у наборі;
– *get*() – призначений для повернення значення біта в певній позиції;
– *set*() і *reset*() – відповідно встановлюють і скидають заданий біт або весь набір бітів;
– *count*() – повертає кількість встановлених бітів у наборі;
– *print*() – виводить набір бітів.

Клас *BitSet* повинен забезпечувати виконання побітових операцій над об'єктами класу ("не", "і", "або", ін.). Важливою умовою вдалої реалізації цього класу є висока швидкість виконання методів класу, оскільки вони викликатимуться найчастіше.

У класі *Node* оголошено такі властивості та методи:

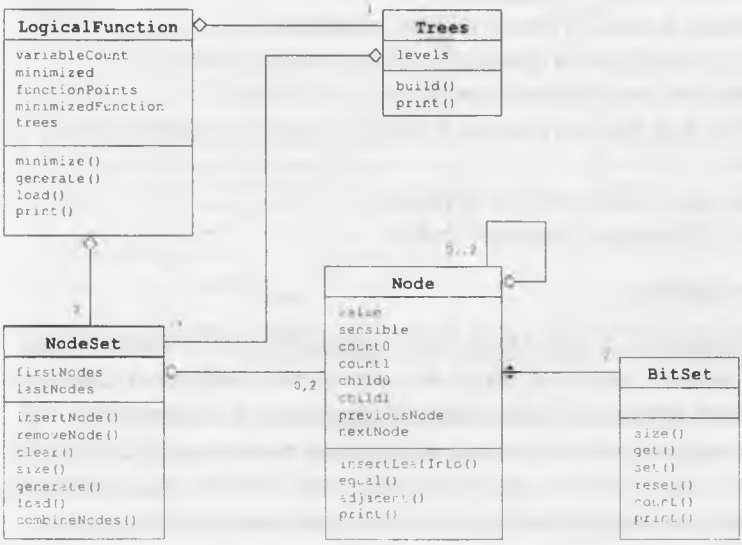

Рис. 1. Статична модель програми

– *value* – набір бітів, зберігає значення змінних у *p*-графі;
– *sensible* – набір бітів, зберігає входження змінних у *p*-граф;
– *count*0, *count*1 – містять відповідно кількості нулів і одиничок у *p*-графі;
– *child*0, *child*1 – вказують на *p*-графи (відповідно на граф з нулем і на граф з одиничкою), з яких утворений даний *p*-граф;

– *insertLeafInto*() – призначений для вставляння усіх *p*-графів, з яких утворений *p*-граф, у певний набір точок;
– *equal*() – використовується для перевірки на рівність з іншим об'єктом цього ж класу;
– *adjacent*() – перевіряє, чи даний граф суміжний з іншим графом;
– *print*() – виводить *p*-граф.

На рисунку не показано властивості об'єктів класу *Node*, які використовуються для зв'язування *p*-графів у списки (*previousNode* і *nextNode*), оскільки вони не важливі для розуміння даного класу.

Клас *NodeSet*, який використовується для оперування множинами *p*-графів, є двонапрямленим списком і тому містить властивості *firstNodes* і *lastNodes* для зберігання перших і останніх елементів з однаковою кількістю одиничок, а також методи:
– *insertNode*() – для вставляння нових елементів;
– *removeNode*() – для видалення елементів;
– *clear*() – для очищення списку;
– *size*() – для отримання кількості елементів у списку;
– *generate*() – для заповнення списку випадковими елементами;
– *load*() – для заповнення списку елементами з файла;
– *combineNodes*() – для створення нового списку об'єднанням суміжних елементів списку.

Для зберігання і побудови дерева згортання призначений клас *Trees*. Його головна (і єдина) властивість – це масив наборів вершин. Метод *build*() використовується при побудові дерева згортання для утворення вищих рівнів із нижчих.

Щоб подавати логічні функції, виконувати операції над ними та зберігати результати цих операцій, було розроблено клас *LogicalFunction*. Властивості та методи класу *LogicalFunction* є такими:
– *variableCount* – кількість змінних (вимірів) функції;
– *minimized* – ознака мінімізованості функції;
– *functionPoints* – точки, в яких функція набуває значення 1;
– *minimizedFunction* – мінімізована функція у вигляді набору точок;
– *trees* – дерево згортання, що утворюється під час мінімізації;
– *minimize*() – метод для мінімізування функції, заданої *functionPoints*, у *minimizedFunction*;
– *generate*() – генерує випадкову логічну функцію;
– *load*() – завантажує функцію із заданого файла.

3. Дослідження алгоритму

Експериментально доведено, що час мінімізації для цього методу значно збільшується при наближенні кількості значень функції до максимально можливої кількості для заданого виміру логічної функції. Наприклад, для функції з 9-ма змінними час мінімізації значно зростатиме, якщо кількість точок наближатиметься до 512 (29). Це відбувається внаслідок того, що кількість пар *p*-підграфів, які об'єднуються, стрімко збільшується, а це призводить до зростання кількості *p*-підграфів на вищих рівнях дерева згортання (табл.і рис.2).

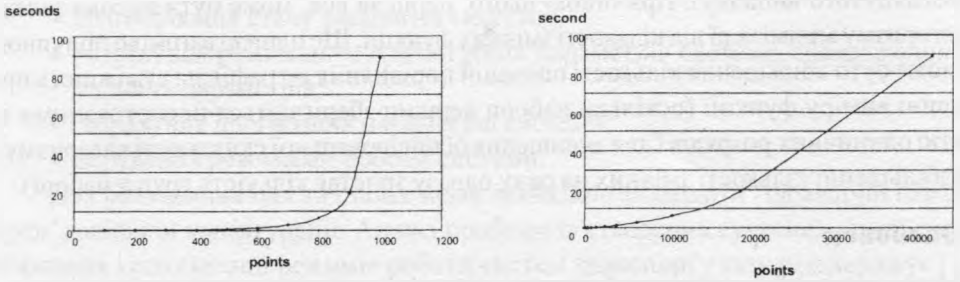

Рис. 2. Залежності часу пошуку мінімільного виразу від кількості значень функції (ліворуч) і від загальної кількості *p*-графів (праворуч) для функції 10-го виміру

Як бачимо, час на графіку праворуч зростає значно повільніше, ніж на тому, що зліва. Отже, основна проблема, яку необхідно вирішити, щоб прискорити виконання програми, – збільшення кількості вершин на вищих рівнях дерева згортання.

Дослідження функції 10-и змінних

	100	200	300	400	500	600	700	800	900	1000
дерево	0,003	0,020	0,057	0,123	0,280	0,631	1,439	3,582	13,259	77,221
покриття	0,007	0,010	0,043	0,113	0,273	0,587	1,251	2,731	5,921	11,853
загалом	0,010	0,030	0,100	0,236	0,553	1,218	2,690	6,313	19,180	89,074
Кількості *p* -графів у дереві згортання										
1-граф	46	192	435	781	1229	1754	2389	3126	3957	4882
2-граф		9	80	269	674	1341	2501	4286	6879	10479
3-граф			1	9	59	198	711	2123	5462	12676
4-граф						2	28	253	1668	9107
5-граф								2	116	3660
6-граф									1	662
7-граф										32
загалом	146	401	816	1459	2462	3895	6329	10590	18983	42498

Один з можливих шляхів розв'язання цієї проблеми полягає у наведенні початкової функції у вигляді КНФ, якщо кількість вершин функції є більшою за половину максимальної кількості вешин для даного виміру логічної функції. Це дасть змогу уникнути різкого зростання кількості суміжних *p*-графів і, як наслідок, прискорити виконання програми для функцій з великою кількістю точок.

Розглянемо результати експериментів, проведених над логічними функціями 15 та 20 змінних на проміжку від 400 до 4000 вершин (рис. 3).

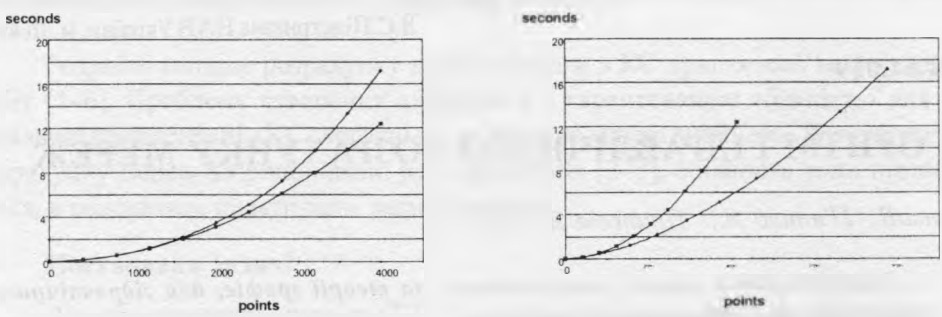

Рис. 3. Порівняльні графіки для функцій 15-го і 20-го вимірів: час пошуку від кількості значень функцій (зліва) і від загальної кількості *p*-графів (справа)

Як бачимо з першого графіка, час обрахунку мінімального виразу відрізняється незначно. Це зумовлено тим, що загальні кількості *p*-графів є незначними і майже не відрізняються.

Другий графік показує, що при однаковій кількості *p*-графів пошук мінімального виразу для логічної функції з більшою вимірністю триває приблизно у 2 рази довше

(для розглянутого випадку). Причиною цього, перш за все, може бути висока залежність алгоритму мінімізації від кількості змінних функції. Ще однією вагомою причиною цього може бути збільшення кількості операцій порівняння *p*-графів на суміжність при збільшенні виміру функції (оскільки набори вершин зберігаються посортованими за кількістю одиничних розрядів (для зменшення обчислювальної складності алгоритму), то при збільшенні кількості змінних на одну одразу зростає кількість груп у наборі).

Висновок

Метод нечіткої кластеризації показує хороші часові результати порівняно з методом Квейна – Мак-Класкі, і він підходить для функцій з невеликою кількістю змінних і точок. А для функцій з великою кількістю змінних і значень алгоритм потребує вдосконалення.

1. Мельник Р.А. Алгоритми ієрархічного моделювання просторової та площинної топології НВІС. – Львів: Держ. ун-т "Львівська політехніка", – 1999. – 180 с.
2. Object Management Group. Unified Modeling Language Specification. – 1999. – 808 с.

В. Павленко*, Я. П'янило, М. Притула
*Національна академія державної податкової служби України, м. Ірпінь
Центр математичного моделювання Інституту прикладних проблем механіки і математики ім. Я.С.Підстригача НАН України, м. Львів

УДК 622.64.029

АЛГОРИТМ ГІДРАВЛІЧНОГО РОЗРАХУНКУ МЕРЕЖ

Запропоновано метод, що базується на теорії графів, для гідравлічного розрахунку газотранспортних систем.

The method for hydraulic calculation of gas - transporter systems based on theory of graphs is proposed in this work.

Постановка проблеми

При побудові програмно - технічних комплексів управління процесами в газотранспортних системах виділяють такі групи взаємозв'язаних задач: