УДК 519.685

Т. Н. Романова, А. В. Анисимов

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ ОПРЕДЕЛЕНИЯ НАБОРА КОМПОНЕНТОВ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Приведена математическая постановка задачи определения оптимального набора компонентов информационной системы. Рассмотрены два варианта задачи, относящиеся к разным классам сложности, и методы их решения. Приведен сравнительный анализ эффективности методов применительно к рассматриваемым задачам.

Ключевые слова: *информационная система, оптимальный набор компонентов.*

Крупная информационная система со сложной структурой является нетипичным объектом рассмотрения с точки зрения разработки программного обеспечения. Однако доля таких систем в сфере информационных технологий весьма велика и продолжает стремительно расти. Задача определения оптимального набора компонентов такой системы относится к классу задач структурного синтеза, которые привлекают к себе внимание специалистов различного профиля. Схожие задачи возникают в сфере схемотехники, цифровой и вычислительной техники, информационных систем, лингвистики, распознавания образов, теории автоматического управления, кибернетики и многих других областях науки и техники.

Рассмотрим информационную систему, которая состоит из большого числа исходных кодов (файлов). Специфика такой системы заключается в том, что производитель поставляет базовую конфигурацию в компанию, которая занимается распространением и поддержкой данной системы. В этой компании информационная система дорабатывается под нужды каждого заказчика индивидуально. Это схематично изображено на рис. 1.

Под доработкой понимается требование заказчика добавить какую-либо функцию в информационную систему, что выражается в разработке дополнительного числа исходных кодов, которые добавляются к конфигурации заказчика.

Однако может случиться так, что требуемая функция уже разрабатывалась для другого заказчика. В таком случае целесообразно осуществить ее перенос. Рассмотрим более сложный случай, когда требуется перенести целый набор функциональных возможностей, и все они были найдены среди нескольких конфигураций других заказчиков. В этом случае необходимо выбрать конфигурации, из которых будет осуществлен перенос функций. Особенностью задачи является то, что

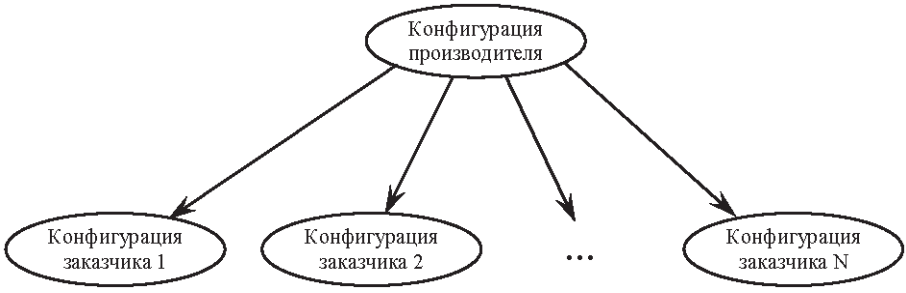

Рис. 1. Дерево конфигураций

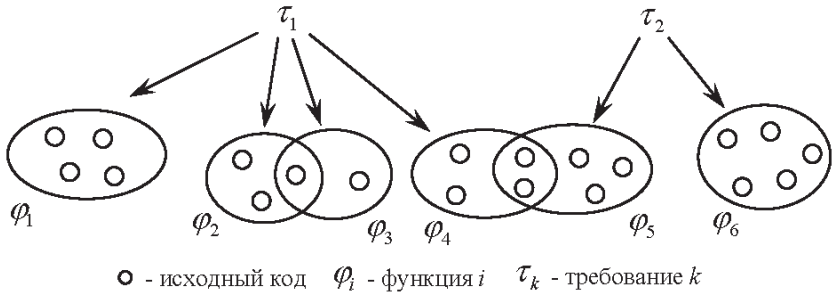

Рис. 2. Функции, удовлетворяющие требованиям

разные функции при реализации могут иметь общие исходные коды (рис. 2).

Из рисунка видно, что одному требованию может соответствовать несколько функций. Каждой функции сопоставлено подмножество исходных кодов, которые ее реализуют. Реализации разных функций могут иметь общие исходные коды, т.е. подмножества исходных кодов могут пересекаться. Пересекаться могут реализации функций, соответствующие как одному требованию, так и разным.

На рис. 3 схематично изображены подмножества исходных кодов, реализующие различные функции. Для удобства восприятия все функции снабжены дополнительным индексом, указывающим на требование, к которому они относятся. Некоторые из представленных на рис. 3 подмножеств исходных кодов, реализующих ту или иную функцию, пересекаются, поэтому мощность объединения подмножеств исходных кодов функций будет меньше суммы мощностей подмножеств исходных кодов функций.

Допустим, что все функции, соответствующие одному требованию, равноправны. На примере требования τ_2 можно сказать, что возможности системы в целом не зависят от того, какая функция и сопоставленное ей подмножество исходных кодов будут выбраны: φ_5^2, φ_8^2 или φ_{12}^2. Однако при необходимости реализации сразу нескольких требований есть смысл уменьшить число производимых операций, например уменьшив число исходных кодов, которые следует обработать. Этого можно добиться, комбинируя различные функции. Поскольку по

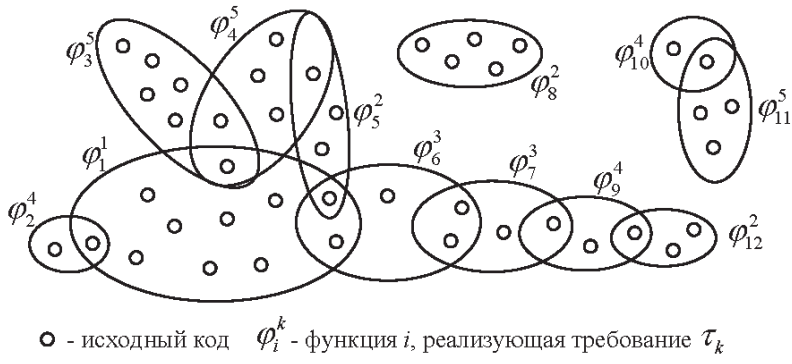

Рис. 3. Пересекающиеся подмножества исходных кодов

условию задачи для каждого требования может быть более одной реализующей его функции, то их следует выбирать так, чтобы число общих исходных кодов для них было максимальным, но при этом сопоставленное каждой функции число исходных кодов было минимальным. Для такой комбинаторно-оптимизационной задачи следует использовать автоматизированный поиск решения.

Постановка задачи. Итак, задача ставится следующим образом: среди сопоставленных функциям подмножеств исходных кодов, соответствующих всем возможным вариантам реализации выставленных требований, выбрать подмножества, объединение которых будет содержать минимальное число исходных кодов.

На рис. 4 приведены четыре примера выбора оптимального подмножества функций. В каждом из четырех примеров представлены два требования и по две функции к каждому требованию. Требуется выбрать по одной функции для каждого требования так, чтобы мощность объединения выбранных функций была наименьшей из всех возможных. Жирным выделены выбранные функции. Объединение их исходных кодов является минимальным в каждом примере, реализующим все выставленные требования. Для нахождения подмножества функций с минимальным числом исходных кодов следует осуществить поиск среди групп из n функций, где n равно числу выставленных требований, т.е. каждому требованию соответствует одна функция. Данную задачу целесообразно решать как прикладную задачу структурного синтеза [1] с оптимизацией на графах.

Построим модель объекта проектирования. На рис. 3 представлена предметная область в виде гиперграфа, но для данного метода такое представление должно быть видоизменено. Для приведенного гиперграфа построим соответствующий ему обыкновенный неориентированный граф, руководствуясь следующими правилами. Каждому ребру гиперграфа поставим во взаимно-однозначное соответствие вершину нового графа. В новом графе вершины будут обозначать функции, ребра — наличие общих исходных кодов у двух функций. Вершины и

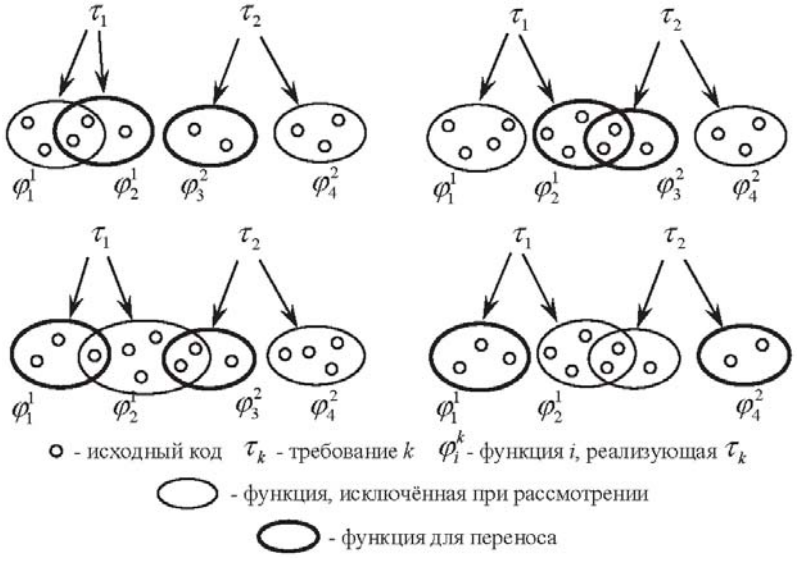

Рис. 4. Примеры выбора оптимального подмножества функций

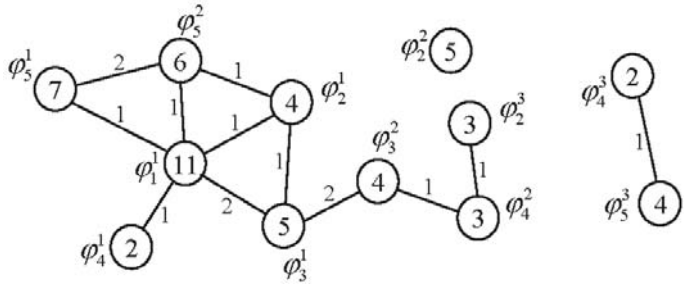

Рис. 5. Пересечения функций в виде неориентированного графа

ребра нового графа будут иметь вес. Вес вершины нового графа будет обозначать число исходных кодов (файлов с исходными кодами), которые реализуют представляемую вершиной функцию. Вес ребра будет обозначать число общих исходных кодов у двух функций, сопоставленных вершинам, которые соединяет ребро. Отсутствие ребра между вершинами означает отсутствие общих исходных кодов у двух функций и равнозначно ребру с нулевым весом. На рис. 5 приведен обыкновенный неориентированный граф, соответствующий гиперграфу, представленному на рис. 3.

Модель результата проектирования будет представлена также обыкновенным неориентированным графом, не имеющим циклов, построенным на вершинах графа модели объекта проектирования. Модель будет иметь такой вид, так как в ходе решения задачи необходимо определить, какие из функций следует отобрать для переноса. За основу будет взят пустой граф с теми же вершинами, что и граф объекта проектирования. По ходу решения выбранные для переноса

функции будут соединяться ребрами, и в итоге граф будет содержать единственную компоненту связности, соответствующую найденному решению.

Математическая постановка оптимизационной задачи. Для выставленных требований $\tau_k \in T_n$ найти $\Phi_{n\text{-opt}} \in \{\Phi_n\} : M(\cup\varphi_i^k) \to \min$, $\varphi_i^k \in \Phi_{n\text{-opt}}$ при условии $\Phi_{n\text{-opt}} \mapsto T_n$, или для выставленных требований $\tau_k \in T_n$ выполнить преобразование D: $G \xrightarrow{D} \Phi_{n\text{-opt}} : M(\cup\varphi_i^k) \to \min$, $\varphi_i^k \in \Phi_{n\text{-opt}}$ при условии $\Phi_{n\text{-opt}} \mapsto T_n$, где τ_k — k-е требование; n — число требований (τ_k, $k = \overline{1,n}$); k — индекс требования; φ_i^k — функция номер i, отвечающая требованию τ_k; G — множество всех функций; Φ_n — результирующее множество отобранных функций; $\Phi_{n\text{-opt}}$ — результирующее множество функций, принятое за оптимальное при данном решении (при неточном методе — локально-оптимальное); $M(\cup\varphi_i^k)$ — мощность объединения функций, входящих в результирующее множество $\Phi_{n\text{-opt}}$; T_n — подмножество множества всех требований мощности n, соответствующее выставленным требованиям; $\mapsto$ — отношение порядка "реализовывать". Под этим будем понимать отношение на множествах функций и исходных кодов. Запись $\Phi_n \mapsto T_n$ означает, что мощности множеств Φ_n и T_n совпадают и для $\forall\tau_k \in T_n \exists\varphi_i^k \in \Phi_n : \varphi_i^k \mapsto \tau_k$.

Учет общих исходных кодов функций при решении задачи. Ранее уже говорилось о том, что подмножества исходных кодов, сопоставленные функциям, могут пересекаться. Учет этого фактора в значительной степени влияет на сложность задачи. Рассмотрим вариант задачи, который не учитывает пересечение подмножеств.

Пусть есть перечень из n требований τ, которые должны быть удовлетворены полностью. По результатам обзора системы контроля версий, которая хранит в себе все исходные коды, для каждого τ_k, $k = \overline{1,n}$, найден некий набор функций, которые реализуют k-е требование и не пересекаются между собой (рис. 6). В этом случае требуется для каждого из n требований выбрать по одной функции, реализующей это требование. Таким образом, получается результирующее множество функций. В этом виде задача является линейной и решается с помощью жадного алгоритма Радо–Эдмондса [2]. Однако на практике чаще встречаются задачи, в которых необходимо учитывать пересечение подмножеств исходных кодов, сопоставленных функциям. На рис. 7 схематично изображено наличие общих исходных кодов. Штриховыми линиями соединены исходные коды, которые являются общими для различных функций. Фактически это одни и те же исходные коды, единые физические файлы. В этом случае выбор функции для i-го требования влияет на выбор функций для остальных требований.

Согласно положениям о классах сложности задач постановка задачи, не учитывающая пересечение подмножеств исходных кодов, реализующих функции, относится к классу сложности P. Постановка

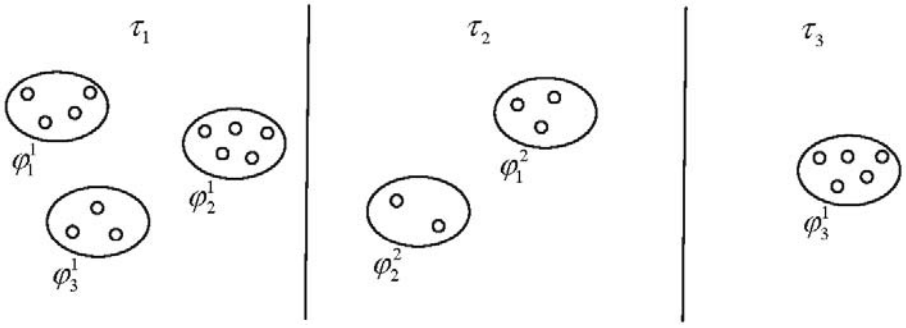

Рис. 6. Функции, реализующие требования

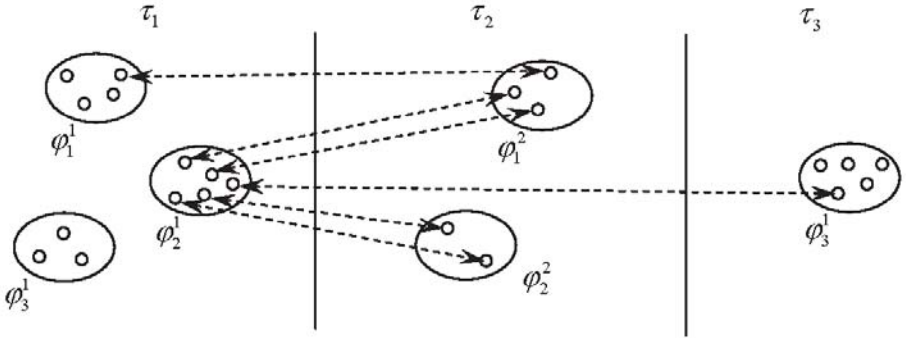

Рис. 7. Наличие общих исходных кодов

задачи, в которой учитывается пересечение подмножеств исходных кодов, относится к классу NP-трудных задач [3].

Методы решения NP-трудных задач. Для решения задач такого класса не найдено точных алгоритмов, работающих за полиномиальное время (хотя и не доказано, что их не существует). Это фактически означает что, решая NP-трудную задачу, не имеет смысла искать точный алгоритм, а следует воспользоваться одним из распространенных приближенных методов. К таким методам относятся метод случайного поиска (последовательность случайных выборов вариантов решения); псевдополиномиальные алгоритмы (подкласс динамического программирования); метод локальных улучшений (поиск лучшего решения в некоторой окрестности допустимого); генетические алгоритмы.

Применительно к поставленной задаче были рассмотрены все перечисленные методы, а также методы полного перебора и ветвей, и границ (идея отсечений заведомо неоптимальных решений целыми классами в соответствии с некоторой оценкой). Для всех них (за исключением генетических алгоритмов, которые нецелесообразно использовать для данной задачи) были построены оценки вычислительной сложности, они приведены далее. Жадный алгоритм Радо–Эдмондса, используемый для поиска начального решения, также рассматривается в качестве вспомогательного метода.

Результаты сравнения методов решения задачи. Прежде чем сравнивать методы, необходимо договориться о критериях, на основе которых такое сравнение проводится. Для данной предметной области можно выделить следующие основные критерии: время работы, трудозатраты (память, вычислительные ресурсы), число выполняемых операций, объем переносимого кода, число обрабатываемых файлов, отклонение по дате.

Для рассматриваемой в настоящей статье задачи из перечисленного будем учитывать только критерии времени работы и число выполняемых элементарных операций, как наиболее значимые. Поскольку данные задачи хранятся в базе данных, а основная работа при поиске решения связана с выбором данных из базы (выполнением реляционных операторов SELECT), то за эталонную операцию примем операцию извлечения одной записи из базы данных по ключевым атрибутам (индексированным полям).

В таком контексте критерии учета числа элементарных операций и времени работы суть одно и то же, так как общее время выбора записей из базы данных линейно зависит от числа выбираемых записей.

Для аналитического сравнения рассматриваемых методов воспользуемся асимптотическими оценками вычислительной сложности алгоритмов типа $O(n)$ [4]. Основанием для использования такой оценки является тот факт, что многие практические задачи имеют большую размерность.

Приведем оценки для описанных методов. Оценку построим как функцию одного переменного — объема входных данных. В данном случае в качестве переменного целесообразно выбрать число требований и ввести две константы: среднее число функций, приходящихся на одно требование, и среднее число исходных кодов, приходящихся на одну функцию. Число функций, приходящихся на требование, обозначим через p, а исходных кодов, приходящихся на функцию, — через q. Порядки p и q гораздо ниже числа исходных кодов входных данных задачи.

Оценка для жадного алгоритма Радо–Эдмондса. Этот метод, позволяющий легко решить задачу без учета пересечений, является вспомогательным для многих других методов, и его оценка также понадобится. В рамках данного метода совершается обход всех требований и для каждого требования отыскиваются все функции, среди которых выбирается функция с наименьшей мощностью подмножества реализующих ее исходных кодов. Если входные данные содержат n требований, для каждого требования определено в среднем p функций, то при поиске решения придется рассмотреть np функций. Поскольку каждой функции соответствует в среднем q исходных кодов, то окончательно оценка примет вид: $\varphi_{\text{Р–Э}}(n) = npq$.

Оценка для метода полного перебора. Данный метод начинается с построения множества всех решений данной задачи. Для этого необходимо извлечь из базы данных все функции, на что требуется *np* элементарных операций.

На каждом шаге алгоритма необходимо вычислить целевую функцию для полного решения. Для каждой комбинации функций (для каждого решения) необходимо отобрать *nq* исходных кодов. Для исключения дубликатов необходимо провести их поиск, последовательно сравнивая исходные коды. Это добавляет $\sum_{i=1}^{qn-1} i = \frac{(qn-1)qn}{2}$ элементарных операций.

Однако наибольший вклад в оценку дает число повторений алгоритма. Для перебора всех возможных решений понадобится p^n повторений. Получаем окончательную оценку для метода:

$$\varphi_{\text{полн}}(n) = pn + p^n \left(qn + \frac{qn(qn-1)}{2} \right).$$

Оценка для метода случайного поиска. Оценка для данного метода будет во многом зависеть от того, какое число шагов будет выбрано. При малом числе шагов оценка получится небольшой, однако большой точности можно достичь только при сравнительно большом числе повторений алгоритма.

Для построения случайных решений необходимо выбрать все функции. Для этого требуется *np* элементарных операций. Для каждого случайно выбранного решения требуется вычислить целевую функцию. Целевая функция вычисляется для полного решения, такие затраты также были уже вычислены (метод полного перебора), они равны $qn + \frac{qn(qn-1)}{2}$. В итоге оценка будет иметь следующий вид:

$$\varphi_{\text{случ}}(n) = k \left(pn + qn + \frac{qn(qn-1)}{2} \right),$$

где k — число повторений алгоритма.

Оценка для метода динамического программирования. Данный метод имеет вполне конкретное число проходов. Для каждого требования необходимо отыскать все функции и выбрать ту, для которой целевая функция, вычисленная по частичному решению, дает наилучший результат.

Число проходов алгоритма будет определяться произведением числа требований на среднее число функций, приходящихся на требование, т.е. *np*.

Затраты на вычисление целевой функции будут увеличиваться от шага к шагу. Их можно вычислить по сумме ряда

$$\sum_{i=1}^{n} pqi = pq\frac{n(n+1)}{2}.$$

При этом следует учесть расходы на исключение дубликатов, которое проводится сравнением исходных кодов при присоединении очередной функции:

$$\sum_{i=1}^{n-1} q^2 i = q^2 \frac{(n-1)n}{2}.$$

Оценка для всего метода примет вид

$$\varphi_{\text{динамич}}(n) = pn + \frac{pqn(n+1)}{2} + \frac{q^2(n-1)n}{2}.$$

Оценка для метода локальных улучшений. Этому методу требуется начальное решение, которое можно было бы улучшать. В качестве начального решения целесообразно взять значение, вычисленное жадным алгоритмом Радо–Эдмондса. Как было показано ранее, затраты на поиск решения без учета пересечений равны npq. В данном методе осуществляется столько же проходов, сколько и в методе динамического программирования, т.е. pn проходов. На каждом шаге вычисляется значение целевой функции для полного решения. Такое значений также уже вычислялось, оно равно

$$qn + \frac{qn(qn-1)}{2}.$$

В итоге получаем оценку сложности для метода локальных улучшений:

$$\varphi_{\text{локальн}}(n) = pqn + pn + pn\left(qn + \frac{qn(qn-1)}{2}\right).$$

Оценка для метода ветвей и границ. Метод ветвей и границ хуже других поддается оценке. Во-первых, число его проходов сильно зависит от входных параметров. Во-вторых, при реализации данного метода требуется хранить в памяти дерево решений и вычисленные оценки, что может потребовать чрезвычайно больших вычислительных ресурсов. Здесь нельзя пренебречь операциями записи, затраты на которые значительно больше, чем затраты на чтение.

Оценим затраты на чтение так же, как это делалось для других методов. Полное число вершин (если потребуется построить все дерево целиком) будет равно $1 + p^n - p^2$. Для каждой вершины потребуется однократно вычислить мощности частичных решений по всем ее дочерним вершинам, а также при каждом проходе провести анализ вычеркнутых дочерних вершин (сравнение с эталоном). Для сравнения потребуется p элементарных операций.

Оценим затраты на вычисление мощностей частичных решений. Это будет сумма ряда, учитывающая уровень дерева. Число вершин на уровне i будет равняться p^{i-1}, для каждой вершины частичное решение будет состоять из $(i-1)q$ исходных кодов. К этому частичному

Порядки сложности оценок методов

Алгоритм	Порядок
Радо–Эдмондса	n
Динамическое программирование	n^2
Локальных улучшений	n^3
Полного перебора	$p^n n^2$
Ветвей и границ	$p^{n+1} + p^n n$
Случайного поиска	$k(n)n^2$

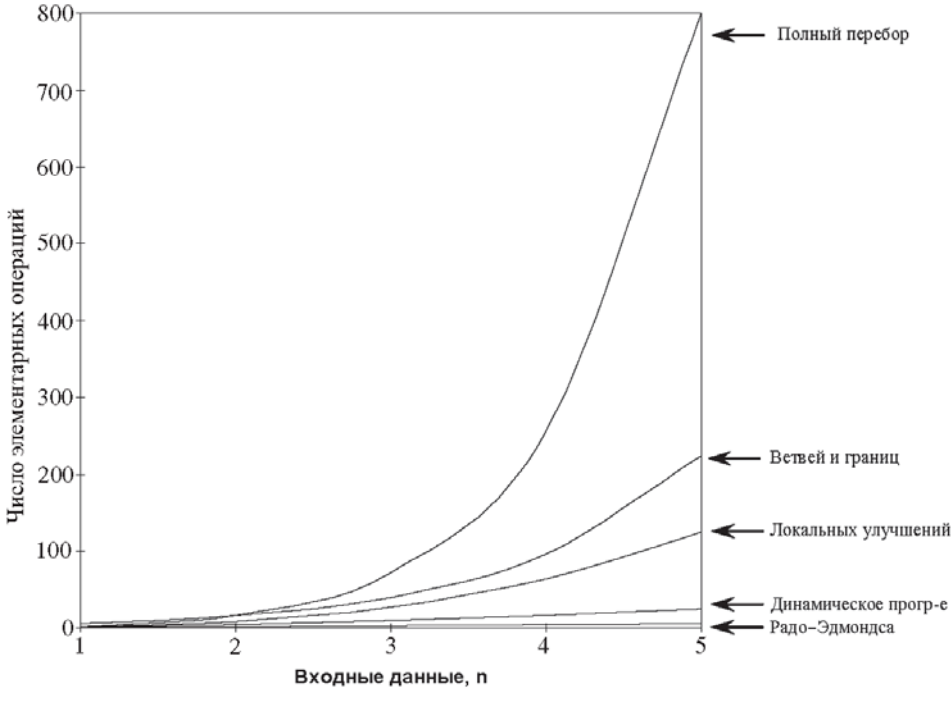

Рис. 8. Графическое сравнение методов

решению надо присоединить еще q исходных кодов p раз. В итоге для вычисления целевых функций потребуется $\sum\limits_{i=1}^{n} p^i iq$ элементарных операций чтения. Получаем оценку сложности для метода ветвей и границ:

$$\varphi_{\text{в–г}}(n) = (1 + p^n - p^2)p(p+1) + \sum_{i=1}^{n} p^i iq.$$

Конечно, это оценка для случая, когда потребуется обход всего дерева, что маловероятно. Тем не менее, по порядку величины она сопоставима с оценкой для метода полного перебора.

Ниже приведены все вычисленные порядки сложности для различных методов, а на рис. 8 — графическое сравнение методов при $p = 2$. Графики построены только для небольших значений входных данных, так как рост функций сильно отличается и их сложно отобразить на одном рисунке.

Жадный алгоритм Радо–Эдмондса, не учитывающий пересечений, находит решение за линейное время. Методы динамического программирования и локальных улучшений работают за полиномиальное время, однако не гарантируют нахождения точного решения. Методы полного перебора ветвей и границ обеспечат точное решение, но время работы оценивается как степенная зависимость. На приведенном графике метод ветвей и границ показывает достаточно хорошие результаты (близкие к методу локальных улучшений), однако с ростом n значение функции резко возрастает.

Выводы. Рассмотрена комбинаторно-оптимизационная задача выбора оптимального набора исходных кодов для переноса функциональных возможностей с учетом общих исходных кодов и без. Получены две постановки задачи, относящиеся к классам сложности NP и P соответственно.

Задачу класса сложности NP предложено решать как задачу структурного синтеза. Рассмотрена возможность применения пяти различных методов нахождения результирующего множества функций, проведены оценки сложности вычислений. Полученные результаты говорят о целесообразности использования методов динамического программирования и локальных улучшений.

СПИСОК ЛИТЕРАТУРЫ

1. Божко А. Н., Толпаров А. Ч. Структурный синтез на элементах с ограниченной сочетаемостью // Наука в образовании: электронное научное издание. – № 5. – 2004.
2. Томас Х. Кормен. Алгоритмы: построение и анализ. – М.: Вильямс, 2006.
3. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982.
4. Овчинников В. А. Алгоритмизация комбинаторно-оптимизационных задач при проектировании ЭВМ и систем. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.

Статья поступила в редакцию 16.09.2008

Татьяна Николаевна Романова окончила МГУ им. М.В. Ломоносова. в 1980 г. Канд. физ.-мат. наук, доцент кафедры “Программное обеспечение ЭВМ и информационные технологии” МГТУ им. Н.Э. Баумана. Автор 8 научных работ в области программирования и аэрогидродинамики.

T.N. Romanova graduated from the Lomonosov Moscow State University in 1980. Ph. D. (Phys.-Math.), assoc. professor of “Computer Software and Information Technologies” department of the Bauman Moscow State Technical University. Author of 8 publications in the field of programming and aero-hydrodynamics.

Алексей Викторович Анисимов родился в 1982 г., окончил МИФИ в 2005 г. Аспирант кафедры “Программное обеспечение ЭВМ и информационные технологии” МГТУ им. Н.Э. Баумана. Автор 5 научных работ в области конфигурационного управления и управления изменениями в исходных кодах.

A.V. Anisimov (b. 1982) graduated from the Moscow Engineering and Physics Institute in 2005. Post-graduate of “Computer Software and Information Technologies” department of the Bauman Moscow State Technical University. Author of 5 publications in the field of configuration management and control of variation in source codes.