IT-ТЕХНОЛОГИИ:
РАЗВИТИЕ И ПРИЛОЖЕНИЯ

Материалы семинара

Владикавказ 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ГОРНО-МЕТАЛЛУРГИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ)»

Кафедра автоматизированной обработки информации

IT-ТЕХНОЛОГИИ:

РАЗВИТИЕ И ПРИЛОЖЕНИЯ

Материалы семинара

Владикавказ 2016

УДК 004
ББК 73
И74

И74 **IT-технологии: развитие и приложения:** Материалы семинара / Коллектив авторов; Северо-Кавказский горно-металлургический институт (государственный технологический университет). – Владикавказ: Северо-Кавказский горно-металлургический институт (государственный технологический университет). Изд-во «Терек», 2016. – 94 с.

ISBN 978-5-901585-90-0

В сборник вошли лучшие доклады, представленные на семинаре «IT-технологии: развитие и приложения» в декабре 2015 года. В них подведены итоги и отражены результаты по основным направлениям исследований, осуществлявшихся преподавателями и студентами кафедры автоматизированной обработки информации СКГМИ (ГТУ) в 2015 году. В рамках этих направлений были предложены новые технологии принятия решений, усовершенствованы методы определения эффективных стратегий антивирусной защиты компьютеров, разработаны и экспериментально проверены альтернативные Стандартной Модели подходы к моделированию сил гравитационного взаимодействия, получили развитие современные методы экстремального программирования, защиты информации и обработки изображений.

УДК 004
ББК 73

Редактор: *Хадарцева Ф. С.*

Компьютерная верстка: *Кравчук Т. А.*

ISBN 978-5-901585-90-0

Подписано в печать 15.04.2016. Формат бумаги 60x84 $^{1}/_{16}$. Бумага офсетная. Гарнитура «Таймс». Печать на ризографе. Усл. п.л. 5,46. Уч.-изд.л. 3,85. Тираж 40 экз. Заказ № ____.
Северо-Кавказский горно-металлургический институт (государственный технологический университет). Изд-во «Терек».
Отпечатано в отделе оперативной полиграфии СКГМИ (ГТУ).
362021. Владикавказ, ул. Николаева, 44.

НОВЫЕ ТЕХНОЛОГИИ ПРИНЯТИЯ РЕШЕНИЙ

УДК 519.834

Будаева А. А.

ИСПОЛЬЗОВАНИЕ МЕТОДА ЭТАЛОНОВ ДЛЯ ПОВЫШЕНИЯ КАЧЕСТВА ГРУППИРОВОК В ЗАДАЧАХ ТАКСОНОМИИ

В работе формулируется задача определения оптимальной многокритериальной таксономии, приводятся примеры различных критериев и формальные постановки задач. Для получения Парето-оптимальных решений предлагается использовать метод эталонов.

Ключевые слова: *таксономия, оптимизация, многокритериальность, эталон, дискретная оптимизация, булевы переменные.*

1. Введение

Применение эталонов для решения многокритериальных задач не является чем-то новым [1–5], однако области их применения до настоящего времени сравнительно ограничены. Ниже демонстрируется применение метода эталонов в таксономии.

Таксономия (или кластерный анализ) – это задача разбиения множества объектов на группы, называемые таксонами (кластерами). Внутри каждой группы должны оказаться «похожие» объекты, а объекты разных группы должны быть как можно более отличны.

Методы таксономии находят широкое применение в различных отраслях науки (социология, палеонтология, археология, биология и т. п.) при автоматическом формировании перечня образов по обучающей выборке. Как известно, одной, "самой естественной", "абсолютно объективной" таксономии не существует, так как все реальные объекты имеют бесконечное число свойств, полностью учесть которые невозможно.

Решением задачи таксономии являются разбиения, удовлетворяющие некоторому критерию оптимальности. Этот критерий часто представляет собой некоторый функционал, выражающий уровни же-

лательности различных разбиений и группировок, который называют целевой функцией.

Выбор критерия оптимальности зависит от решаемой задачи.

Применение таксономии в общем виде сводится к следующим этапам:

1. Отбор выборки объектов для таксономии.
2. Определение множества признаков, по которым будут оцениваться объекты в выборке. При необходимости – нормализация значений признаков.
3. Вычисление значений меры сходства между объектами.
4. Применение метода таксономии для создания групп сходных объектов (таксонов).
5. Представление результатов анализа.

Таким образом, очевидно, что перечень групп четко не задан и определяется в процессе работы алгоритма.

После получения и анализа результатов возможна корректировка выбранной метрики и метода кластеризации до получения оптимального результата.

2. Задачи оптимальной таксономии

Введем следующие **обозначения**:

N – число таксонов,

$z(i, j)$ – булева переменная, равная единице, если i-й объект принадлежит j-му таксону, и равная нулю в противном случае,

$r(i, j)$ – расстояние между i-м и j-м объектами в λ-пространстве,

$R(i)$ – радиус i-го таксона,

$r(c_i, p)$ – расстояние между центром i-го таксона и p-м объектом.

Обозначения, используемые локально, вводятся далее по ходу изложения.

Как уже было сказано выше, выбор критерия оптимальности зависит от решаемой задачи. Приведем *несколько* ***примеров*** задач оптимальной таксономии [6]:

Пусть *m* объектов следует распределить между *n* таксонами таким образом, чтобы суммарное расстояние между объектами, принадлежащими одному таксону, было минимально. Примером прикладной задачи, сводимой к сформулированной выше, является поиск оптимальной стратегии прокладки оптоволоконного кабеля между пользователями сети Internet при условии, что:

- суммарные затраты на прокладку кабеля минимальны;
- выход в *Internet* осуществляется через *N* спутниковых терминалов, причем каждый терминал принадлежит одному таксону.

Формальная постановка задачи имеет вид (1):

$$\begin{cases} \sum_{i=1}^{n}\sum_{p=1}^{m}\sum_{q\neq p} r(p,q)\cdot z(p,i)\cdot z(q,i) \to \min \\ \forall p : \sum_{i=1}^{n} z(p,i) = 1 \\ \forall p, \forall i : z(p,i) = 1,0. \end{cases} \quad (1)$$

Близкая задача возникает в системах эфирного вещания и сотовой связи: *N* передатчиков следует разместить на местности таким образом, чтобы условия их использования потребителями были наиболее комфортными. Иными словами, следует распределить все объекты (пользователей) по *N* таксонам таким образом, чтобы максимальное расстояние объекта до центра «своего» таксона было минимальным. Формальная постановка такой задачи отличается от (1) только целевой функцией:

$$\begin{cases} \max_{i}\max_{p}\ r(c_i, p)\cdot z(p,i) \to \min \\ \forall p : \sum_{i=1}^{n} z(p,i) = 1 \\ \forall p, \forall i : z(p,i) = 1,0. \end{cases} \quad (2)$$

Возможна модификация задачи (2). Пусть требуется распределить максимальное число объектов по *N* таксонам таким образом, чтобы расстояние объекта до центра «своего» таксона не превышало величи-

ны R_i. Примером прикладной задачи, сводимой к сформулированной выше, является поиск оптимального распределения максимального количества пользователей между передатчиками в системах эфирного вещания и сотовой связи, каждый из которых характеризуется радиусом действия R_i (система (3))

$$\begin{cases} \sum_{i=1}^{n}\sum_{p=1}^{m} Z(p,i) \to \max \\ \forall i, p : r(c_i, p)\cdot z(p,i) \le R_i \\ \forall p : \sum_{i=1}^{n} z(p,i) \le 1 \\ \forall p, \forall i : z(p,i) = 1,0. \end{cases} \quad (3)$$

Все приведенные математические модели являются однокритериальными и опираются на один из критериев оптимальности: расстояние между объектами внутри таксонов, удаленность от центра таксона и др.

Целевыми критериями также могут служить:

- минимум суммарного расстояния объектов до центров своих таксонов $\sum_i \sum_p r(c_i, p)\cdot z(p,i) \to \min$;
- максимум минимального количества объектов в одном таксоне $\min_i \sum_p z(p,i) \to \max$;
- минимум максимального расстояния между объектами в таксоне $\max_i \max_p r(q,p)\cdot Z(q,i)\cdot Z(p,i) \to \min$;
- максимум минимального расстояния между объектами разных таксонов $\min_{i,j\neq i} \min_{p,q\neq p} r(p,q)\cdot z(p,i)\cdot z(q,j) \to \max$;
- и др.

Очевидно, что в зависимости от принятого критерия оптимальности получаемые группировки могут отличаться. Тогда, возникает вопрос, насколько «хороша» та или иная таксономия объектов.

В идеале разбиение будет оптимальным, если оно оптимально с точки зрения всех критериев оптимальности. Таким образом, вместо однокритериальной задачи мы получаем многокритериальную задачу, целевыми функциями в которой будут отдельные критерии оптималь-

ности группировок. В общем виде для перечисленных выше критериев такая задача примет вид (4):

$$\begin{cases} F_1 = \sum_{i=1}^{n}\sum_{p=1}^{m}\sum_{q\neq p} r(p,q)\cdot z(p,i)\cdot z(q,i) \to \min \\ F_2 = \sum_{i}\sum_{p} r(c_i,p)\cdot z(p,i) \to \min \\ F_3 = \min_{i}\sum_{p} z(p,i) \to \max \\ F_4 = \max_{i}\max_{p} r(q,p)\cdot Z(q,i)\cdot Z(p,i) \to \min \\ F_5 = \min_{i,j\neq i}\min_{p,q\neq p} r(p,q)\cdot z(p,i)\cdot z(q,j) \to \max \\ F_6 = \max_{i}\max_{p}\ r(c_i,p)\cdot z(p,i) \to \min \\ \forall p: \sum_{i=1}^{n} z(p,i) = 1 \\ \forall p, \forall i: z(p,i) = 1,0. \end{cases} \quad (4)$$

Данная задача относится к задачам дискретной многокритериальной оптимизации. Для ее решения воспользуемся методом эталонов.

2. Метод эталонов

Применение метода эталонов с одной стороны позволяет получать Парето-оптимальные решения, с другой стороны – не требует экспертной оценки исходной информации [2–5]. Суть метода состоит в выделении эталонного решения: наилучшего или наихудшего. Наилучший эталон будет иметь в качестве своих характеристик наилучшие значения целевых критериев задачи (4), а наихудший, соответственно, – наихудшие значения.

В этом случае, оптимальной группировкой будет та, которая наименьшим образом удалена по целевым критериям от наилучшего эталона (или наиболее удалена от наихудшего эталона).

Для определения характеристик наилучшего эталона необходимо решить задачу (4), применительно к каждому критерию, в результате чего получим вектор наилучших значений критериев – $\vec{K} = \{K_1, K_2, \ldots, K_n\}$.

Тогда, задача (4) преобразуется в однокритериальную задачу вида (5).

В качестве меры близости объектов воспользуемся квадратом евклидова расстояния:

$$\begin{cases} R_K\left(\vec{Z}\right) \to \min \\ \forall p : \sum_{i=1}^{n} z\left(p,i\right) = 1 \\ \forall p, \forall i : z\left(p,i\right) = 1,0, \end{cases} \tag{5}$$

где $R_K\left(\vec{Z}\right) = \sum_{i=1}^{n}\left[K_i - F_i\left(\vec{Z}\right)\right]^2$ – квадрат евклидова расстояния до лучшего эталона

Целевая функция в задаче (5) нелинейна. Однако, принимая во внимание дискретность значений вектора $\vec{Z}$, для решения задачи (5) можно воспользоваться методом ветвей и границ.

Таким образом, решение задачи (4) состоит из двух основных этапов: оптимизация каждой отдельной целевой функции с учетом заданных ограничений и минимизация критерия $R_K\left(\vec{Z}\right)$.

Основным преимуществом предлагаемого подхода является возможность кластеризации с учетом разнородных критериев оптимальности. К недостаткам можно отнести большую размерность задач таксономии, и как следствие, – значительные затраты памяти и машинного времени на обработку данных. Для минимизации затрат алгоритм решения нужно распараллелить, распределив расчет отдельных критериев на отдельные рабочие станции.

Заключение

Предложенный выше подход позволяет повысить эффективность решения задач таксономии, выполняя кластеризацию одновременно по нескольким критериям оптимальности. Использование при этом метода эталонов дает возможность, с одной стороны, получать Парето-оптимальные решения, с другой стороны, «естественно» переходить от поиска оптимального решения многокритериальных задач к оптимальному решению задач с одной целевой функцией без привлечения дополнительных условий [3, 4, 5].

Литература

1. *Будаева А. А., Гроппен В. О.* Выбор оптимальной технологии ранжирования // Устойчивое развитие горных территорий. 2014. № 3. С. 3–7.
2. *Гроппен В. О.* Принципы принятия решений с помощью эталонов // АиТ. 2006. № 4. С. 167–184.
3. *Groppen V. O.* New Solution Principle for Multi-criteria Problems Based on Comparison Standards: Models, Algorithms, Applications // Applications to Industrial and Societal Problems. CIMNE, Barcelona, Spain. 2008. P. 201–209.
4. *Гроппен В. О., Вагин В. С., Позднякова Т. А., Будаева А. А.* Многокритериальное ранжирование объектов методом эталонов как инструмент оптимального управления // Устойчивое развитие горных территорий. 2010. № 1. С. 47–56.
5. *Будаева А. А.* Использование метода эталонов для решения задач дискретной многокритериальной оптимизации // Известия Саратовского университета. Новая серия. Серия: Математика. Механика. Информатика. 2015. Т. 15. № 1. С. 22–27.
6. *Будаева А. А.* Математические модели и средства поддержки поиска оптимальных группировок в задачах таксономии: Дис. … канд. техн. наук. Владикавказ, 2004.

Будаева А. А.,
канд. техн. наук, доцент кафедры автоматизированной обработки информации СКГМИ (ГТУ)
e-mail: budalina@yandex.ru

THE USE OF STANDARDS TO IMPROVE THE QUALITY GROUPINGS IN THE PROBLEMS OF TAXONOMY

Associate Professor **A. A. Budaeva**

The paper formulates the problem of determining of the optimal multicriteria taxonomy, are presented examples of different criteria and formal statements of problems. For Pareto-optimal solutions is proposed to use the method of measurement standards.

***Keywords**: taxonomy, optimization.*

УДК 004.942

Даурова А. А.,
Фролов О. И.

РЕШЕНИЕ МНОГОКРИТЕРИАЛЬНОЙ ЗАДАЧИ ПОИСКА ОПТИМАЛЬНОГО РЕЖИМА ФУНКЦИОНИРОВАНИЯ АНТИВИРУСНЫХ ПРОГРАММ С ПОМОЩЬЮ ЭТАЛОНОВ

В работе показана неэффективность существующих стратегий оптимизации использования антивирусных сканеров и предложен подход, позволяющий повысить эффективность их использования. Приводятся результаты машинных экспериментов, подтверждающие эффективность предложенного подхода.

***Ключевые слова**: антивирусные сканеры, частота запуска, блокировка вредоносных программ, восстановление запорченных файлов, оптимизация, выбор модели, экспериментальная проверка.*

1. Введение

Актуальность задачи защиты компьютеров от вирусных атак в наше время не вызывает сомнений, однако реализация различных стратегий антивирусной защиты часто приводит к избыточным затратам процессорного времени на «самообслуживание», отнимая это время у задач пользователя. Целью настоящей работы является поиск математической модели, позволяющей, с одной стороны, минимизировать временные затраты такого рода, а с другой – не требующей значительных вычислительных ресурсов на поиск оптимальной стратегии защиты от вирусов.

2. Обозначения и определения

Постановка и решение задачи защиты данных с помощью антивирусного сканирования предполагают, что время, затрачиваемое на поиск объектов, повергнутых заражению, прямо пропорционально частоте запуска сканера, а время, затрачиваемое на восстановление данных обратно пропорционально ему [1].

Введем следующие обозначения:

ν – частота запуска антивирусного сканера, мин$^{-1}$;

t – время обеспечения безопасной работы компьютера, с;

t_1 – время поиска и блокады вредоносных объектов антивирусным сканером, с. В первом приближении время t_1 прямо пропорционально частоте ν запуска сканера, что подтверждается полученными в ходе эксперимента данными (рис. 1);

t_2 – время восстановления «запорченных» данных, с. Экспериментально также подтверждается, что время t_2 обратно пропорционально частоте ν запуска сканера (рис. 2) [2].

3. Экспериментальные данные

Экспериментальные значения времен t_1 и t_2 сведены в таблицу 1. Очевидно, $t_1 + t_2 = t$.

Таблица 1

Частота ν, мин$^{-1}$	0,0083	0,0100	0,0133	0,0200	0,0333	0,0500	0,1000	0,2000
Время t_1, с	68	81	65	84	70	98	140	196
Время t_2, с	270,6	238,0	214,6	175,4	149,0	120,0	80,0	58,0

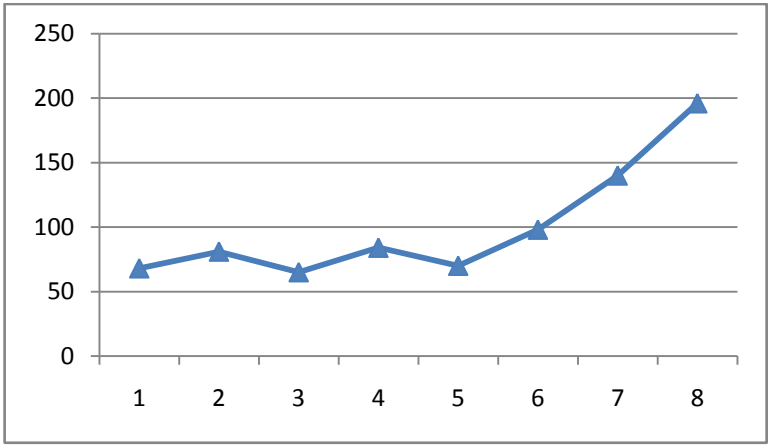


Рис. 1. Зависимость времени t_1 поиска и блокады вирусов от частоты ν запуска антивирусного сканера CureIt производителя ПО Dr.Web. Уравнение линейной регрессии: $t_1 = 675{,}66 \cdot \nu + 63{,}57$

Было показано, что наилучшим образом зависимость времени t_2 от частоты ν аппроксимируется степенной регрессией вида

$t_2 = bv^{-\frac{1}{2}}$ [4]. Однако этот вывод был сделан на основе использования только одного критерия. Представляется целесообразным рассмотреть два критерия и выбрать оптимальную частоту запуска сканера, применяя для решения задачи метод идеальной точки.

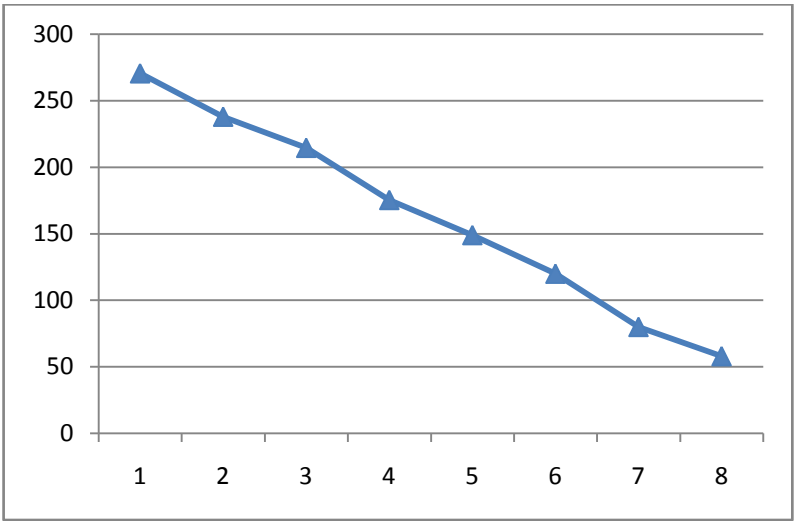


Рис. 2. Зависимость времени t_2 восстановления запорченных данных от частоты v запуска антивирусного сканера CureIt производителя ПО Dr.Web. Экспериментальные данные

4. Выбор модели

Предположения о видах связи:

$$t^{(1)} = av + \frac{b}{v} + c, \quad (1)$$

$$t^{(2)} = av + b\ln v + c, \quad (2)$$

$$t^{(3)} = av + \frac{b}{\sqrt{v}} + c \quad (3)$$

приводят к уравнениям регрессии вида (соответственно):

$$t_{F_1}^{(1)} = 369{,}80 \cdot v + \frac{1{,}40}{v} + 171{,}57,$$

$$t_{F_1}^{(2)} = 1149{,}27 \cdot v - 91{,}22 \cdot \ln v - 117{,}58,$$

$$t_{F_1}^{(3)} = 617{,}12 \cdot v + \frac{22{,}77}{\sqrt{v}} + 81{,}68,$$

если критерием F_1 выбора коэффициентов a, b и c, был минимум суммы квадратов невязок, а также к уравнениям регрессии вида:

$$t_{F_2}^{(1)} = 370{,}65 \cdot v + \frac{2{,}24}{v} + 172{,}39,$$

$$t_{F_2}^{(2)} = 1058{,}06 \cdot v - 86{,}74 \cdot \ln v - 26{,}36,$$

$$t_{F_2}^{(3)} = 616{,}71 \cdot v + \frac{22{,}52}{\sqrt{v}} + 81{,}44,$$

если критерием F_2 выбора коэффициентов a, b и c был минимум наибольшей невязки суммы квадратов отклонений экспериментальных данных от значений, полученных интерполированием. Здесь $t_{F_j}^{(i)}$ соответствует уравнению регрессии в предположении о характере связи $t^{(i)}$ ($i = \overline{1,3}$) при выборе критерия F_j ($j = \overline{1,2}$).

Выбор коэффициентов a, b и c при определении второй группы регрессий производился следующим образом. Диапазон варьирования каждого из коэффициентов определяется наибольшим и наименьшим значениями коэффициентов, полученными методом наименьших квадратов. Шаг варьирования – 0,01. Для каждого набора значений коэффициентов a, b и c находим наибольшие невязки, из которых затем выбираем наименьшую. Соответствующие этому значению коэффициенты определяют искомую регрессию.

Значения полученных коэффициентов приведены в таблице 2.

Таблица 2

	Функция	МНК, F_1	ММНН, F_2
1	$t = av + \frac{b}{v} + c$	a = 369,80 b = 1,40 c = 171,57	a = 370,65 b = 2,24 c = 172,39
2	$t = av + b\ln v + c$	a = 1149,27 b = –91,22 c = –117,58	a = 1058,06 b = –86,74 c = –26,36
3	$t = av + \frac{b}{\sqrt{v}}$+c	a = 617,12 b = 22,77 c = 81,68	a = 616,71 b = 22,52 c = 81,44

Найдем "расстояние" от каждой из полученных регрессий до идеальной точки, которая в нашем случае есть точка с координатами (0; 0):

$$R_1 = \sqrt{t_{F_1}^{(1)^2} + t_{F_2}^{(1)^2}} = \sqrt{206{,}176^2 + 106{,}746^2} = 232{,}17,$$

$$R_2 = \sqrt{t_{F_1}^{(2)^2} + t_{F_2}^{(2)^2}} = \sqrt{626{,}187^2 + 59{,}436^2} = 629{,}00,$$

$$R_3 = \sqrt{t_{F_1}^{(3)^2} + t_{F_2}^{(3)^2}} = \sqrt{196{,}800^2 + 6{,}385^2} = 196{,}90.$$

Таким образом, $\min\{R_i\} = 196{,}90$, что соответствует регрессии вида:

$$y = 616{,}92 \cdot v + \frac{22{,}65}{\sqrt{v}} + 81{,}56.$$

Минимум этой функции достигается в точке $v_{\text{опт}} = 0{,}069$ мин$^{-1}$; значение $t_{\text{опт}} = t(v_{\text{опт}}) = 210{,}96$ с = 3,52 мин, что соответствует периоду запуска антивируса 14,49 минуты.

5. Заключение

Предложенный подход, как показывают результаты экспериментов, позволяет существенно повысить эффективность работы антивирусных сканеров. Одним из направлений его дальнейшего развития является расширение диапазона анализируемых функций, другим – учет времени поиска оптимальной стратегии с помощью каждой модели наряду с временем, затраченным на блокаду вирусов и восстановление запорченных файлов.

Литература

1. *Сорокин С. В.* Система оптимальной защиты информации (СОЗИ) // Труды Северо-Кавказского государственного технологического университета. 2002. Вып. 9. С. 192–198.

2. *Петров А. Ю.* Многофакторная оптимизация задачи антивирусной защиты // Вестник Воронежского государственного технического университета. Воронеж: Изд-во Воронежского ГТУ, 2011. № 7. С. 137–144.

3. *Гроппен В. О., Петров А. Ю.* Многокритериальная задача оптимизации режимов функционирования антивирусных сканеров // Сборник научных трудов Северо-Осетинского отделения Академии наук высшей школы РФ. Владикавказ: СКГМИ (ГТУ). 2011. № 9. С. 56–58.

4. *Даурова А. А.* К задаче поиска оптимального режима функционирования антивирусных программ // Материалы XIV Международной научно-технической конференции. ИТ-Технологии: развитие и приложения. Владикавказ: Фламинго, 2013.

Даурова А. А.,
канд. техн. наук, доцент кафедры
автоматизированной обработки информации
СКГМИ (ГТУ)
e-mail: Albina_Daurova@mail.ru

Фролов О. И.,
студент СКГМИ (ГТУ)
e-mail: frolov93oleg@mail.ru

SOLUTION OF A MULTI-CRITERIA PROBLEM FOR FINDING OF THE OPTIMAL MODE OF OPERATION OF ANTI-VIRUS PROGRAMS BY STANDARDS USAGE

Associate Professor **A. A. Daurova**, student **O. I. Frolov**

We prove the low effectiveness of used today strategies of anti-virus scanners application, and suggested an approach permitting to improve the efficiency of their use. The results of computer experiments confirming the effectiveness of the proposed approach are also presented.

Keywords: *anti-virus scanners, frequency, blocking malware, a corrupt file recovery, optimization, model selection, experimental verification.*

УДК 004.942

Даурова А. А.,
Пономарева А. Н.

ИССЛЕДОВАНИЕ ФУНКЦИЙ, АРГУМЕНТАМИ КОТОРЫХ ЯВЛЯЮТСЯ ПЕРЕСТАНОВКИ

Работа посвящена описанию алгоритма поиска решения экстремальных задач, аргументами которых являются перестановки. Используется терминология теории графов. Работа алгоритма иллюстрируется примером.

Ключевые слова: *экстремум, перестановка, граф, координаты вершин, перебор.*

1. Введение

Пусть целевая функция некоторой задачи имеет вид $f = < c_i, x_i > \rightarrow \text{extr}$ и является функцией на комбинаторной конфигурации $X = \{x\}$. Заданы также ограничения $A_{ij}x_j \leq b_i$. Тогда задача формулируется следующим образом: найти элемент конфигурации, для которого функция достигает экстремального (максимального, минимального) значения и при этом выполняются условия задачи (ограничения) [3].

Далее рассмотрен алгоритм поиска точек конфигурации, удовлетворяющих ограничениям задачи, и пример, иллюстрирующий его работу.

2. Алгоритм модифицированного координатного метода

Шаг 1. Определяем количество вершин графа комбинаторной конфигурации [1, 2].

Шаг 2. Определяем количество крайних точек в общем графе каждого из подграфов.

Шаг 3. Находим максимальное и минимальное значения заданной целевой функции в крайних точках подграфов общего графа:

$$f(x)\max = c_1x_{k-n+1} + c_2x_{k-n+2} + \ldots + c_{n-1}x_{k-1} + c_nx_k ,$$

$$f(x)\min = c_nx_1 + c_{n-1}x_2 + \ldots + c_2x_{n-k-1} + c_1x_{n-k.}$$

Строим структурный граф, который отображает только крайние вершины подграфа: p_1^i, $q_1^{i'}$.

Шаг 4. Определяем значение целевой функции в начальной $p_1^{\,i}$ и конечной $q_1^{\,i'}$ вершинах подграфов. Для поиска значений функции в следующих крайних вершинах воспользуемся технологией координатного метода:

a) для получения координат начальных вершин подграфа с фиксированной координатой m для j-го уровня необходимо выполнить последовательно перестановки элемента $(m-1)$ с $(m-(j+1))$;

b) для получения координат конечных вершин подграфа с фиксированной координатой m для j-го уровня необходимо последовательно выполнить перестановки элемента $(m-1)$ и $(j-1)$;

c) согласно координатному методу значения функции находится по формуле:

$$f(p_1^{\,i}) = f(p_k^{\,i}) - \Delta\,,$$

где $\Delta = (j_k - j_l)(c_k - c_{ц(l)})$;

j_k, j_l – координаты вершины;

$c_k, c_{ц(l)}$ – коэффициенты функции.

Шаг 5. Для каждого структурного подграфа проверяем следующие условия: если для начальной вершины структурного графа выполняется условие $f(p^i_{i1}) \le b_i$, то весь подграф удовлетворяет ограничению j и включается в множество $Dj \subset X$. Переходим к рассмотрению следующего подграфа или к шагу 10, если все подграфы рассмотрены, иначе переходим к шагу 7.

Шаг 6. Если $f(p^i_{i1}) \ge b_i$ и для конечной вершины выполняется условие $f(q^i_{i1}) \ge b_i$, то рассматриваемый подграф не удовлетворяет условию и не включается в множество $Dj \subset X$. Переходим к рассмотрению следующего подграфа или к шагу 10, если все подграфы рассмотрены; иначе – к шагу 8.

Шаг 7. Если $f(p^i_{i1}) \ge b_i$, а $f(q^i_{i1}) \le b$, то часть элементов подграфа удовлетворяет условию, и их поиск осуществляется с помощью углубления в структуру графа. Переходим к шагу 9.

Шаг 8. В подграфе, определенном на шаге 7, фиксируем последнюю из незафиксированных координат. Переходим к шагу 4.

Шаг 9. Формируем множество точек-элементов конфигурации, для которых $X \subset Dj$.

Выполнив этот алгоритм для всех ограничений, получим k множеств $Di \subset X$, где $i \in Nk$.

Шаг 10. Находим пересечение $D^* = D1 \cap D2 \cap ... \cap Dk$. Теперь для поиска максимального (минимального) значения достаточно найти значение в максимально (минимально) возможных точках каждой из гиперплоскостей.

3. Пример

Рассмотрим пример применения данного алгоритма.

Пусть задана целевая функция:

$$f(x) = x_1 + 13x_2 + 2x_3 + 4x_4 \to \max.$$

В качестве ограничительной задана функция:

$$g(x) = 4x_1 + 7x_2 + 3x_3 + 6x_4 \leq 49$$

при условии, что все переменные неотрицательны.

Тогда $\pi = \begin{matrix} 1 & 2 & 3 & 4 \\ 4 & 7 & 3 & 6 \end{matrix}$. Упорядочим коэффициенты по возрастанию: $\pi^{-1} = \begin{matrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 6 & 7 \end{matrix}$.

Получим неравенство $3x_1 + 4x_2 + 6x_3 + 7x_4 \leq 49$. Далее работа продолжается с функцией

$$g_1(x) = 3x_1 + 4x_2 + 6x_3 + 7x_4.$$

Найдем минимум и максимум этой функции:

$$g_1(x)_{\min} = g(7, 6, 4, 3) = 43$$
$$g_1(x)_{\max} = g(3, 4, 6, 7) = 57.$$

Построим структурный граф.

(1234) ——————————— (3214)
(1243) ——————————— (4213)
(1342) ——————————— (4312)
(2341) ——————————— (4321)

Вычислим значения Δ для левых вершин:

$$\Delta_1 = (4 - 3) \cdot (7 - 6) = 1$$
$$\Delta_2 = (3 - 2) \cdot (7 - 4) = 3$$
$$\Delta_3 = (2 - 1) \cdot (7 - 3) = 4$$

Значения Δ для правых вершин:

$$\Delta_1 = (4 - 3) \cdot (7 - 3) = 4$$
$$\Delta_2 = (3 - 2) \cdot (7 - 4) = 3$$
$$\Delta_3 = (2 - 1) \cdot (7 - 6) = 1$$

Используя полученные значения, заполним таблицу, согласно вышеприведенному алгоритму (табл. 1):

Таблица 1

G_1^i	P_1^i	$g_1(p_1^i)$	q_1^i	$g_1(q_1^i)$
G_1^4	(1234)	57	(3214)	51
G_1^3	(1243)	56	(4213)	47
G_1^2	(1342)	53	(4312)	44
G_1^1	(2341)	49	(4321)	43

Учитывая ограничения, заключаем, что точки подграфа G_1^4 не удовлетворяют условию, т. к. значения функции во всех точках этого подграфа превышают заданное. Искомые точки находятся в подграфах G_1^3, G_1^2 и G_1^1. Их мы и рассмотрим далее.

Построим структурный граф G_1^3: – подграф с одной зафиксированной координатой – последней, равной 3:

(1243) ——————————— (2143)
(1423) ——————————— (4123)
(2413) ——————————— (4213)

Аналогично, вычислим значения Δ для левых и правых вершин того подграфа, занесем в таблицу 2 следующие значения:

Таблица 2

G_1^{3i}	P_1^i	$g_1(p_1^i)$	q_1^i	$g_1(q_1^i)$
G_1^{34}	(1243)	56	(2143)	55
G_1^{32}	(1423)	52	(4123)	49
G_1^{31}	(2413)	49	(4213)	47

Получаем два удовлетворяющих условию интервала: [49; 52], [47; 49]. Следовательно, далее рассматриваем подграфы с двумя зафиксированными координатами: G_1^{32} и G_1^{31}, которым отвечают таблицы 3 и 4.

Структурный граф G_1^{32}:

(1423) ——————————— (1423)
(4123) ——————————— (4123).

Таблица 3

G_1^{32i}	P_1^i	$g_1(p_1^i)$	q_1^i	$g_1(q_1^i)$
G_1^{324}	(1423)	52	(1423)	52
G_1^{321}	(4123)	49	(4123)	49

Т. е. в этом подграфе лишь одна перестановка удовлетворяет условию – (4123).

Переходим к структурному графу G_1^{31}:

(2413) ——————————— (2413)
(4213) ——————————— (4213).

Таблица 4

G_1^{31i}	P_1^i	$g_1(p_1^i)$	q_1^i	$g_1(q_1^i)$
G_1^{314}	(2413)	49	(2413)	49
G_1^{312}	(4213)	47	(4213)	47

Как видно из таблицы 4, обе эти точки, координаты которых (2413) и (4213), удовлетворяют ограничению.

Теперь, когда подграфы G_1^3 рассмотрены и все подходящие под ограничение перестановки на нем найдены, можно перейти к G_1^2. Построим его (см. табл. 5).

(1342) ——————————— (3142)
(1432) ——————————— (4132)
(3412) ——————————— (4312

Таблица 5

G_1^{2i}	P_1^i	$g_1(p_1^i)$	q_1^i	$g_1(q_1^i)$
G_1^{24}	(1342)	53	(3142)	51
G_1^{23}	(1432)	51	(4132)	48
G_1^{21}	(3412)	45	(4312)	44

Искомые точки можно найти только на подграфах G_1^{23} и G_1^{21}.

G_1^{23}: (1432) ——————————— (1432)
(1432) ——————————— (4132).

Таблица 6

G_1^{23i}	P_1^i	$g_1(p_1^i)$	q_1^i	$g_1(q_1^i)$
G_1^{234}	(1432)	51	(1432)	51
G_1^{231}	(4132)	48	(4132)	48

Видим, что лишь одна точка с координатами (4132) подходит под ограничение.

G_1^{21}: (3412) ——————— (3412)
(4312) ——————— (4312)

Таблица 7

G_1^{21i}	P_1^i	$g_1(p_1^i)$	q_1^i	$g_1(q_1^i)$
G_1^{214}	(3412)	45	(3412)	45
G_1^{213}	(4312)	44	(4312)	44

Оба подграфа, G_1^{214} и G_1^{213}, содержат подходящие точки.

Из таблицы 1 видно, что все точки подграфов графа G_1^1 также удовлетворяют условию и будут включены в множество найденных перестановок. Поиск завершается.

В результате выполнения алгоритма будем иметь таблицу (табл. 8), составленную из найденных перестановок – координат точек – и значений ограничительной функции в этих точках.

Таблица 8

Координаты вершины p	Значение функции
(4123)	49
(2413)	47
(4213)	49
(4132)	48
(3412)	45
(4312)	44
(2341)	49
(3241)	48
(2431)	47
(4231)	45
(3421)	44
(4321)	43

Таким образом, мы получили 12 перестановок из возможных 4! = 24. Далее рассматривать целесообразно только их. Поэтому переходим к целевой функции. Согласно условию задачи, необходимо максимизировать функцию:

$$f(x) = x_1 + 13x_2 + 2x_3 + 4x_4.$$

Для этого найдем ее значения на полученных перестановках и выберем максимальное из них.

Таблица 9

Координаты вершины *p*	Значение целевой функции
(4123)	35
(2413)	68
(4213)	44
(4132)	31
(3412)	65
(4312)	53
(2341)	53
(3241)	41
(2431)	64
(4231)	40
(3421)	63
(4321)	51

Получаем, что максимального значения функция достигает в точке с координатами (2, 4, 1, 3). Ответ: $x_1 = 2$, $x_2 = 4$, $x_3 = 1$, $x_4 = 3$, $f = 68$.

Заключение

К достоинствам предложенного выше подхода можно отнести гарантию получения глобально оптимального решения и хорошие шансы на сокращение объема перебора.

Литература

1. *Донец Г. П., Колечкина Л. М.* Экстремальные задачи на комбинаторных конфигурациях: Монография. Полтава: ПУЕТ, 2011. 362 с.

2. *Донец Г. А., Колечкина Л. Н.* Об одной задаче оптимизации дробно-линейной функции цели на перестановках // Проблемы управления и информатики. 2010. № 2. С. 31–41.

3. *Колечкина Л. Н., Дверная Е. А.* Модифицированный подход к решению векторных задач оптимизации на комбинаторных конфигурациях // Радиоэлектроника и информатика. 2011. № 3. С. 41–44.

Даурова А. А.,
канд. техн. наук, доцент кафедры
автоматизированной обработки информации
СКГМИ (ГТУ).
e-mail albina_daurova@mail.ru

Пономарева А. Н.,
студентка СКГМИ (ГТУ).
e-mail mischiefcj@mail.ru

STUDY OF FUNCTIONS WHOSE ARGUMENTS ARE PERMUTATIONS

Associate Professor **A. A. Daurova,** student **A. N. Ponomareva**

This work is devoted to description of search algorithm for solving extreme problems, which arguments are permutations. It uses the terminology of graph theory. The algorithm is illustrated by an example.

***Keywords**: extremum, permutation, graph, coordinates of the vertices, exhaustive search.*

ТЕХНОЛОГИИ ЭКСТРЕМАЛЬНОГО ПРОГРАММИРОВАНИЯ

УДК 681.343.001

Томаев М. Х.,
Горькавая Е. Ю.,
Литвинов Д. И.

ВЫБОР ОПТИМАЛЬНОЙ СТРАТЕГИИ МАКРОЗАМЕН В ПРОГРАММАХ, НАПИСАННЫХ НА ЯЗЫКЕ «C++»

Предлагается подход для оптимизации вызова функций методом макрозамен с помощью inline-подстановок. В работе формулируются критерии оптимальности различных стратегий макрозамен применительно к зацикленным и конечным алгоритмам. Описываются методы решения, а также особенности программной реализации. Полученный в результате проведенных исследований программный продукт позволяет повысить эффективность программных систем, а реализованные в работе модели и подходы могут быть использованы в составе комплексных средств оптимизации в качестве одного из инструментов.

Ключевые слова: *программа, оптимизация, макрозамена, память, модель, производительность.*

1. Обзор метода макрозамен. Анализ эффективности

Быстрое развитие аппаратной составляющей вычислительных комплексов привело к смещению приоритетов в отрасли разработки программного назначения. Подход к созданию продуктов стал более экстенсивным: если раньше большое значение уделялось эффективности использования вычислительных ресурсов пользовательским программным кодом, то теперь этот критерий уходит на второй план, а главным критерием потребительской привлекательности становятся количество предоставляемых услуг, т. е. степень автоматизации прикладной отрасли деятельности, а также удобство использования. Од-

новременно с этим происходит эволюция оптимизационных подходов. В новых условиях все большую актуальность приобретают модели на основе методов так называемого «экстремального программирования», общей характеристикой которых является то, что в каждом из них улучшение одного из критериев качества достигается за счет использования дополнительных ресурсов вычислительной системы (как правило, оперативной памяти). Один из таких подходов – это использование макрозамены [3], в результате которой код функции копируется в место ее вызова. Полученный выигрыш в производительности складывается из времени на передачу управления в функцию и возврат из нее, а также времени на выделение необходимого для размещения всех локальных переменных и массивов объема стековой памяти и может быть сформулирован в виде выражения (1):

$$f(v) = t^{call} + t^{stack}(v), \quad (1)$$

где t^{call} – время на передачу управления в функцию и возврат из нее без учета времени, необходимого на выделение локального стека функции. Эта величина является постоянной и соответствует времени выполнения машинной инструкции CALL (передающей управление процедуре на языке Ассемблера);

$t^{stack}(v)$ – время, затрачиваемое на выделение локальной (стековой) памяти, предназначенной для размещения переменных, объявленных внутри блока функции. Это время зависит от суммарного размера локальных данных, ниже данное утверждение будет подтверждено экспериментально.

Чтобы оценить порядок величин t^{call}, $t^{stack}(v)$ и оценить их значимость в $f(v)$ были проведены ряд экспериментов. Экспериментальные замеры проводились с помощью тестового кода, написанного на языке C++ в среде Visual Studio. Для получения максимально объективных оценок в настройках проекта параметру «Optimization» было присвоено значение «Disabled (/Od)», что предотвращает влияние встроенных в компилятор Microsoft методов оптимизации. Кроме того, параметр «Inline function expansion» был установлен в «Only __inline (/Ob1)» – данный флаг запрещает компилятору игнорировать инструкцию inline при построении исполняемого кода (рис. 1). Замеры проводились в режиме «Release» для исключения диагностического кода.

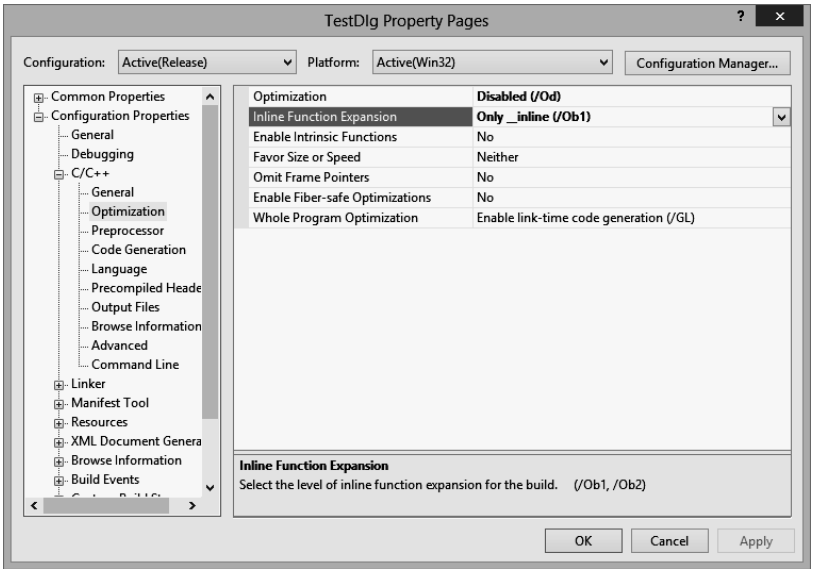


Рис. 1. Настройки тестового проекта

Для оценки величины времени вызова и возврата из функции t^{call} был проведен тест, в котором замерялось время выполнения кода, выполняющего в цикле функцию с пустым телом и без параметров (чтобы исключить влияние операторов выделения стека). Количество итераций цикла последовательно изменялось от 1 000 000 до 20 000 000 с шагом 1 000 000. Исходный код теста приведен в Листинге 1:

```
CString cstr="";
for (int N=1000000; N<=20000000; N+=1000000){
    DWORD dwStart = GetTickCount();
    for (int i=0; i<N; i++){
        f0;
    }
    DWORD dwTime = GetTickCount() – dwStart;
    CString cs; cs.Format("%u\n",dwTime);
    cstr += cs;
}
AfxMessageBox(cstr);
```

Листинг 1. Исходный код тестовой программы.

Были проведены 2 замера: в первом функция была объявлена обычным образом (Листинг 2):

```
void f()
{

}
```

Листинг 2. Объявление функции *f()*.

Во втором случае с использованием инструкции *_forceinline* (Листинг 3):

```
__forceinline void f()
{

}
```

Листинг 3. Объявление функции f() в виде макроса.

Результаты замеров показали, что отличия во времени работы лежали в пределах статистической погрешности. Максимальное время выполнения (для 20 000 000 итераций) составило около 47 микросекунд.

Так как существовала возможность того, что компилятор игнорирует флаг запрета встроенной оптимизации и все-таки исключает функции с пустым телом из объектного кода, то были проведены 2 дополнительных теста: в первом вместо функции *f()* в цикле вызывалось выражение log(10.5) (Листинг 4):

```
CString cstr="";
    for (int N=1000000; N<=20000000; N+=1000000){
        DWORD dwStart = GetTickCount();
        for (int i=0; i<N; i++){
            log(10.5);
        }
        DWORD dwTime = GetTickCount() – dwStart;
        CString cs; cs.Format("%u\n",dwTime);
        cstr += cs;
    }
```

```
AfxMessageBox(cstr);
```

Листинг 4. Функция *f()* заменена на выражение log(10.5).

а во втором тесте выражение *log(10.5)* было помещено в тело функции *f()*(Листинг 5):

```
void f()
{
    log(10.5);
}
```

Листинг 5. Объявление функции *f()*.

Сравнение результатов этих двух тестов также показало очень незначительные расхождения рис. 2. Это говорит о том, что затраты на вызов функции и возврат из нее незначительны.

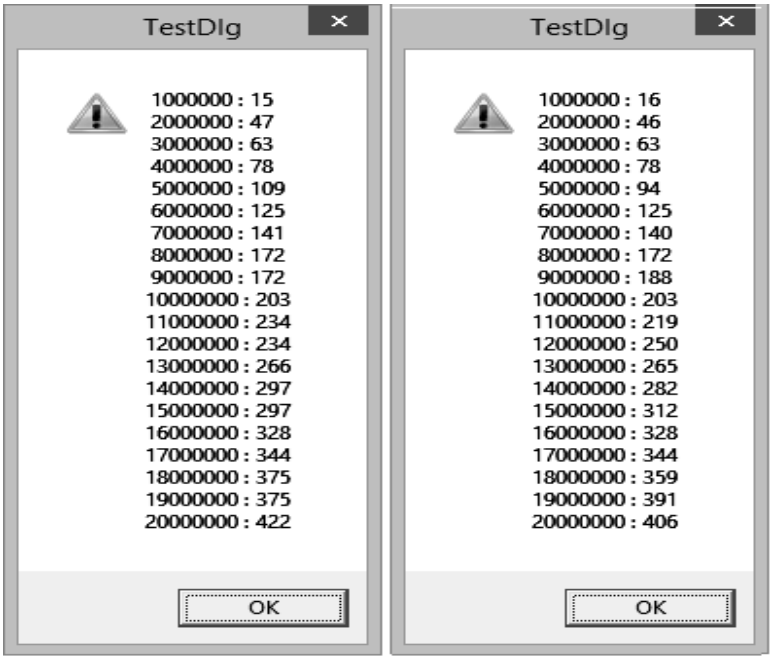


Рис. 2. Результаты 2-го теста оценки t^{call}

Для оценки зависимости времени выделения стековой памяти $t^{stack}(v)$ от размера локальных данных был проведен следующий эксперимент: число итераций было зафиксировано значением 20 000 000, Далее были проведены 10 замеров для каждого из которых менялся размер массива «y», объявленного в теле функции f() (Листинг 6) от 100 000 до 1 000 000 байт включительно:

```
void f()
{
    char y[1000000];
    y[0] = 'a';
}
```

Листинг 6. Исходный код функции на последнем 10-м измерении.

Затем был проведен аналогичный эксперимент для варианта функции *f()* с использованием модификатора *__forceinline*. Результаты обоих экспериментов приведены в таблице ниже.

Результаты замеров времени выделения локальной памяти

Размер данных	Время работы обычной функции	Время работы *__forceinline* функции
100000	579	47
200000	1312	47
300000	2187	47
400000	3468	47
500000	4343	47
600000	5187	47
700000	6079	47
800000	6922	47
900000	7781	47
1000000	8641	47

Полученные результаты говорят о высокой эффективности использования модификатора *_forceinline*: очевидно, что объявление данных компилятор поместил перед циклом. На графике, изображенном на рисунке 3 хорошо виден линейный характер зависимости вре-

мени выделения локального стека функции от его размера: поверх кривой экспериментальных данных, взятых из первой и второй колонок таблицы 1, проведена прямая (пунктир), коэффициенты, которой получены аппроксимацией данных методом МНК.

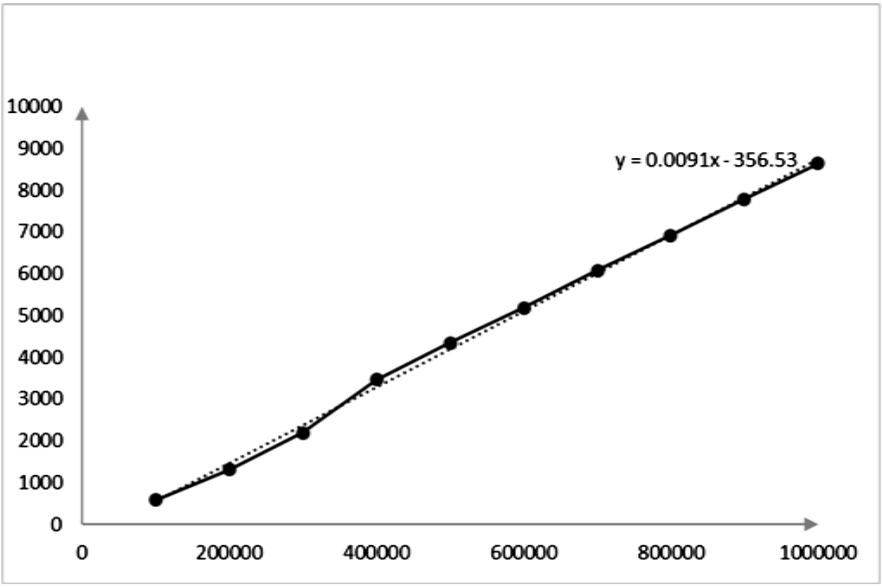


Рис. 3. Кривая экспериментальных данных и прямая, описывающая линейную зависимость времени выделения стековой памяти от ее размера

На графике, изображенном на рис. 3, ось абсцисс соответствует размеру стек (в байтах), а ось ординат – времени выделения стека (в миллисекундах).

Проведенные эксперименты показали, что выражение (1) можно упростить, убрав из него, представив выигрыш от макрозамены t^{call}, ввиду его фактической несущественности (2).

$$f(v) = t^{stack}(v). \tag{2}$$

Кроме того, подтвержденный экспериментально линейный характер зависимости времени выделения стека от его размера позволяет с высокой степенью точности использовать для прогнозирования времени выделения стека коэффициенты пропорциональности либо более

наглядный показатель скорости выделения стека, с учетом этого выражение (2) можно преобразовать в следующее (3):

$$f(v) = \frac{v}{s}, \tag{3}$$

где v – суммарный размер локальных переменных (стека функции);
s – скорость выделения стека.

В методе макрозамен, как и во всех задачах, относящихся к категории моделей «экстремального программирования», улучшение качества программы достигается за счет использования дополнительных вычислительных ресурсов. Макрозамены приводят к увеличению размера кода программы пропорционально количеству вызовов функций, для которых такая замена будет осуществляться. Естественно, в сложных системах с большим числом функций, неограниченное использование макрозамен невозможно – прежде всего из-за того, что оперативная память, используемая для размещения исполняемого кода, является ресурсом достаточно дорогим. Таким образом, имеет место оптимизационная проблема выбора стратегии макрозамен, улучшающая производительность кода, в условиях ограниченного объема ресурсов оперативной памяти, необходимого для достижения данной цели.

При формулировке критерия качества пользовательского кода часто бывает необходимо учитывать особенности структуры алгоритма. В работе [1] предлагается способ классификации алгоритмов на основе признаков, формулируемых с использованием математического аппарата теории графов. Далее будут предложены подходы к оптимизации двух из них – зацикленных и конечных.

2. Оптимальная стратегия макрозамен зацикленных программных алгоритмов

Зацикленные программные алгоритмы характеризуются отсутствием вариантов завершения. Если вероятности переходов в таком алгоритме заранее неизвестны, то программа может зациклиться в любом из контуров. Так как время работы программы на участке любого из контуров в общем случае равно бесконечности, а время выполнения линейных участков кода, не входящих в состав ни одного их контуров, является конечным, то при формулировке оптимизационной

модели линейные участки можно исключать из рассмотрения. Высокая вычислительная сложность дискретных оптимизационных подходов делает необходимым выделение наиболее важных участков кода, оптимизация которых является наиболее актуальной. В работах [1], [2] предлагаются две противоположные стратегии оптимизации, в соответствии с которыми для рассмотрения выбираются контуры, соответствующие либо верхней, либо нижней границе однократного зацикливания. Адаптируя данный подход к методу макрозамен, можно предложить модель (4), описывающую задачу поиска оптимальной стратегии макрозамен, которая минимизирует верхнюю границу однократного зацикливания:

$$\begin{cases} F = \max\limits_{i, a_i \in A(G)} \sum\limits_{k \in K(a_i)} ((1 - z_k) t_k N_k) \to \min; \\ \forall i, \forall k \in K(a_i): \sum\limits_{k \in K(a_i)} z_k v_k N_k \le V \\ z_k = 1,0 \end{cases}, \qquad (4)$$

где $K(a_i)$ – множество функций, лежащих на контуре a_i;

t_k – время на вызов *k*-ой функции;

N_k – количество вызовов *k*-ой функции на контуре a_i;

z_k – булева переменная, равная 1, если для *k*-ой функции выполняется макрозамена и 0 – в противном случае;

v_k – размер *k*-ой функции;

V – верхняя граница, выделенного на оптимизацию дополнительного объема памяти.

В общем случае t_k, с учетом подтвержденной экспериментально зависимости (3), можно сформулировать следующим образом:

$$t_k = \frac{v_k^{param}}{s}, \qquad (5)$$

где v_k^{param} – объем данных, передаваемых в функцию в качестве параметров.

Для того чтобы перейти от игрового подхода к задаче дискретной оптимизации, заменим исходную формулировку двумя следующими (6), (7) задачами, которые решаются последовательно.

Вначале решаем задачу поиска максимального контура (длина контура определяется суммарным временем на вызов функций, лежащих на контуре), для которого выполняется равенство:

$$\sum_{k \in a_q} t_q = \max_i \left(\sum_{k \in a_i} t_k \right), \tag{6}$$

где q – индекс контура максимальной длины.

Далее формулируем задачу поиска оптимальной стратегии макрозамен на множестве функций найденного контура:

$$\begin{cases} F_3 = \sum_{k \in K(a_q)} ((1 - z_k) t_k N_k) \to \min \\ \forall k \in K(a_q): \sum_{k \in K(a_q)} z_k v_k N_k \le V \\ z_k = 1{,}0 \end{cases}, \tag{7}$$

где q – номер контура, соответствующего решению (6).

Для решения данной задачи можно применить различного рода комбинаторные процедуры. В частности, в ходе реализации практической части была разработана программа, осуществляющая поиск решения (7) полным перебором.

3. Программная реализация метода макрозамен циклящихся алгоритмов

3.1. Алгоритм работы программы

Входными данными являются: исходный код программы на C++, верхняя граница памяти и скорость выделения стека.

Выходные данные: функции, рекомендуемые к замене inline.

Алгоритм программы включает следующие шаги:

1) лексический и синтаксический анализ исходного кода;

2) нахождение весов для всех функций;

3) построение матрицы смежности переходов, соответствующей графу состояний, описывающему пользовательский алгоритм;

4) нахождение контуров;

5) выбор контура максимальной длины в соответствии с (6);

6) определение оптимальной стратегии макрозамен на множестве функций, составляющих максимальный контур.

В основу алгоритма поиска контуров был положен метод поиска в глубину. Поиск в глубину (depth-first search, DFS) – это рекурсивный алгоритм обхода вершин графа. Стратегия поиска в глубину состоит в том, чтобы идти «вглубь» графа, насколько это возможно. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из рассматриваемой вершины ребра. Если ребро ведет в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать ребра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось ребер, которые ведут в нерассмотренную вершину. Если после завершения алгоритма не все вершины были рассмотрены, то необходимо запустить алгоритм от одной из нерассмотренных вершин.

Алгоритм поиска в глубину:

Шаг 1. Всем вершинам графа присваивается значение не посещенная (Colors[u] = 0). Выбирается первая вершина и помечается как посещенная (Colors[u] = 1).

Шаг 2. Для последней, помеченной как посещенная, вершины выбирается смежная вершина, являющаяся первой помеченной как не посещенная, и ей присваивается значение посещенная. Если таких вершин нет, то берется предыдущая помеченная вершина.

Шаг 3. Повторить шаг 2 до тех пор, пока все вершины не будут помечены как посещенные.

Заметим, что в каждый момент помеченные вершины образуют путь в графе, в точности соответствующий стеку вызовов функции DFS. Таким образом, если для обрабатываемой вершины алгоритм

находит смежную с ней вершину, которая уже помечена, то это будет означать, что цикл найден.

Исходный код алгоритма поиска в глубину приведен в Листинге 7.

```
private void DFS(int u) //поиск в глубину для вершины u
    {
        Colors[u] = 1; //пометим вершину u
myStack.Push(u); //забросим ее в стек, в стеке находятся вершины,
        //они образуют путь в графе, в точности
        //соответствующий стеку вызовов функции DFS

        //выбираем первую непомеченную вершину, смежную с u
        //вызываем для нее поиск в глубину
        for (int i = 0; i < vertexCount; i++)
        {
            if ((adjacencyMatrix[u, i] == 1) && (Colors[i] == 0)) DFS(i);
        }
    //если нашли смежную с u вершину, которая уже помечена,
    // то цикл найден
        //из стека считываем последовательность вершин, образующих цикл
        for (int i = 0; i < vertexCount; i++)
        {
            if ((adjacencyMatrix[u, i] == 1) && (Colors[i] == 1))
            {
                bool flag = false;
                String s = Convert.ToString(i) + "_";
                foreach (int x in myStack)
                    if (flag == false)
                    {
                        if (x != i) s = Convert.ToString(x) + "_" + s;
                        if (x == i) { flag = true; }
                    }
                //запоминаем найденный контур в список
            myList.Add(s);
            }
        }
        Colors[u] = 0;// снова делаем вершину непомеченной,
  //чтобы она могла участвовать в других циклах
        myStack.Pop();//исключаем ее из стека
```

```
}
```

Листинг 7. Нахождение контуров методом поиска в глубину.

В следующем фрагменте кода (Листинг 8) представлена программная реализация описанного выше алгоритма на языке C#. Исходный код включает комментарии наиболее важных участков.

```
// int k  – кол-во функций на контуре
// int[] Nk_ = new int[k]; количество вызовов k-ой функции на контуре
// float[] Weight_ = new float[k]; вес функций на контуре
// float[]WeightParams_ = new float[k]; вес k-ой функции с учетом объема данных, передаваемых в функцию в качестве параметров
    int []X=new int [k]; //генерируем двоичные комбинации длины K
   for (int i = 0; i < k; i++)
        X[i]=0;
        for (int m=1;m<=Math.Pow(2,k);m++){
        int j=k-1;
        summa = 0;
           float celfunc = 0;
           for (int i = 0; i < k; i++)
            summa += Weight_[i] * X[i];
           if (summa < Params.v)
           {
              celfunc = 0;
              for (int i = 0; i < X.Length; i++)
                 if (X[i] == 0)
                 {
                    celfunc  +=  (float)WeightParams_[i]  /  Params.skor  *
Nk_[i];
                 }

              sum.Add(celfunc);
           }
           //генерация новой комбинации
           while (j >= 0 && X[j] == 1) { X[j] = 0; j--; }
           if (j >= 0) X[j]++;
        }
    int j1 = 0;
    float min = sum[0]; int posmin=0;
    for (j1 = 1; j1 < sum.Count; j1++)
```

```
    {
        if (sum[j1] < sum[posmin]) posmin = j1;
    }
    // значение целевой функции находится в sum[posmin]
}
```

Листинг 8. Исходный код процедуры решения задачи (7) методом полного перебора.

3.2. Интерфейс программы

Программа разработана на языке C# на основе виртуальной машины MS Framework 4.5. Выбор данного языка обосновывался необходимостью обеспечить многоплатформенность реализуемых оптимизационных решений (VS поддерживает разработку под Android, iOS и Windows).

Структурно программа выполнена в виде диалогового окна с элементами управления для загрузки исходного файла пользователя и запуска процесса оптимизации. При запуске программы появляется следующее окно (рис. 4).

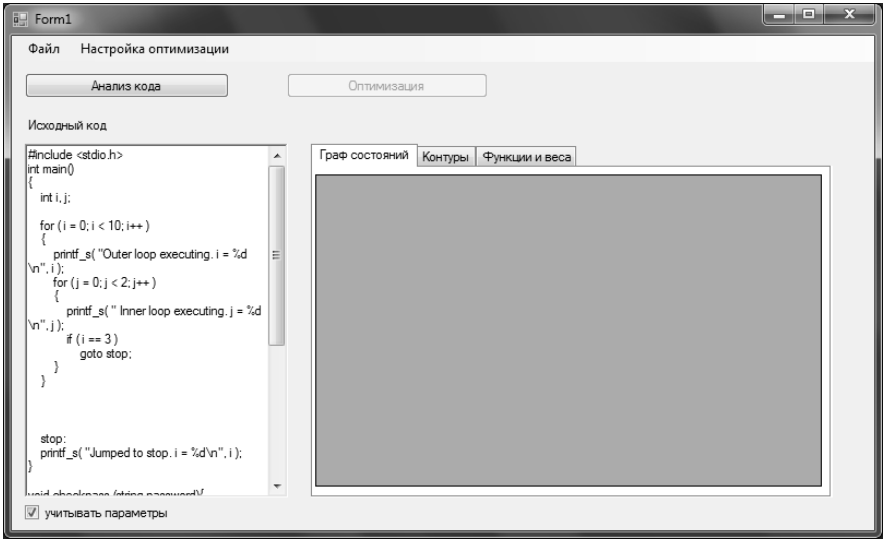


Рис. 4. Окно программы с загруженным исходным кодом пользователя

Окно программы с загруженным исходным кодом пользователя. В левой части окна можно ввести исходный код программы на языке C++ вручную или открыть существующий файл (*.cpp), содержащий готовый код, для чего следует выбрать пункт «Файл» > «Открыть» (рис. 5).

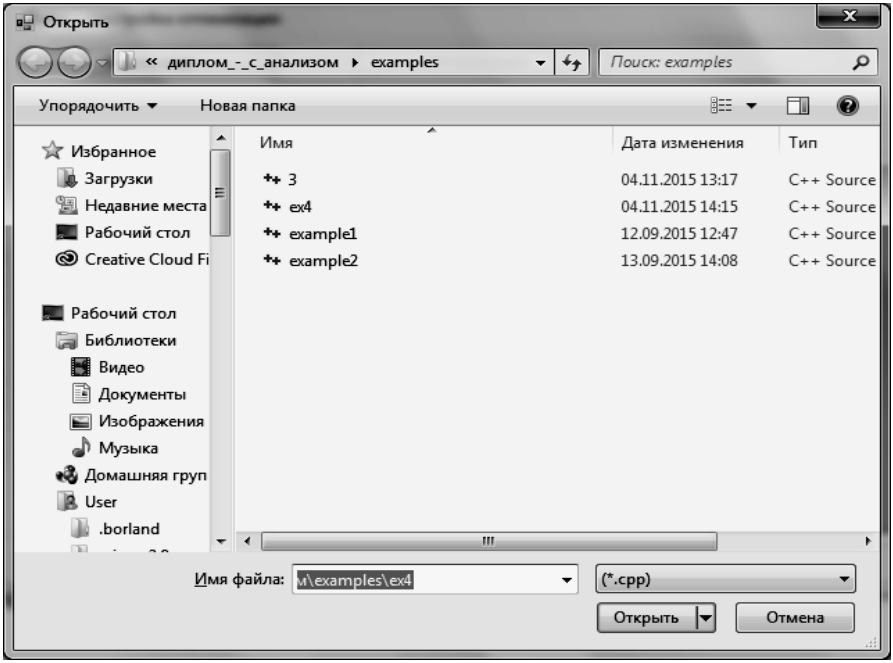


Рис. 5. Пример работы программы. Открытие файла

Кнопка «Анализ кода» предназначена для предварительного анализа кода исходной программы, включающего в себя следующие этапы:

1) лексический и синтаксический анализ кода,
2) нахождение весов для всех функций,
3) построение графа состояний,
4) нахождение контуров.

Результаты анализа отображаются на вкладках «Граф состояний», «Контуры», «Функции и веса», как показано на рис. 6.

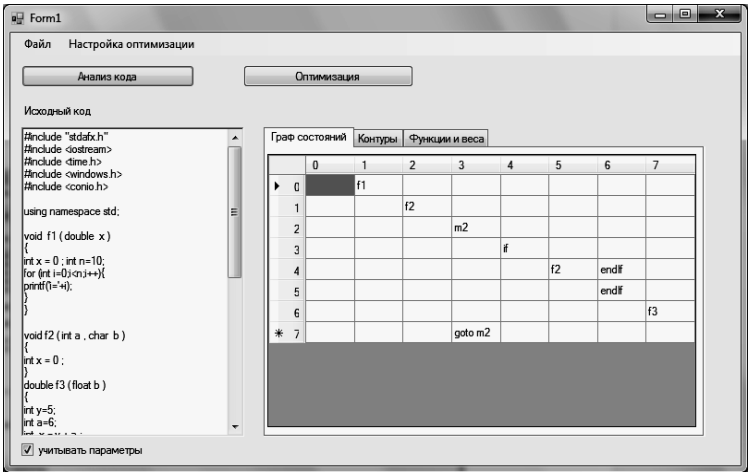


Рис. 6. Матрица смежности, изображенная по результатам анализа исходного кода пользовательского алгоритма

На вкладке «Граф состояний» в матричном виде представлен граф переходов и вызовов. Матрица смежности, изображенная по результатам анализа исходного кода пользовательского алгоритма.

На вкладке «Контуры» (рис. 7) показаны все контуры графа, среди которых выделен максимальный. Для каждого контура указаны функции, лежащие на нем, а также расчетное время на вызова.

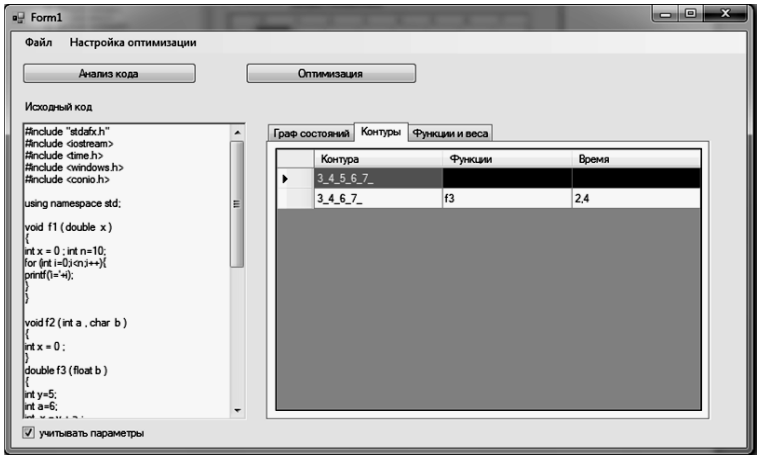


Рис. 7. Пример работы программы. Вкладка «Контуры»

На вкладке «Функции и веса» (рис. 8) можно посмотреть вес функций. В столбце вес рассчитывается полный вес функции с учетом всех параметров, переменных и действий, производимых в функции. В столбце «вес с параметром» отображается вес функций с учетом только параметров (если галочка в пункте «Учитывать параметры не стоит», то помимо параметров еще учитываются переменные, в теле функции).

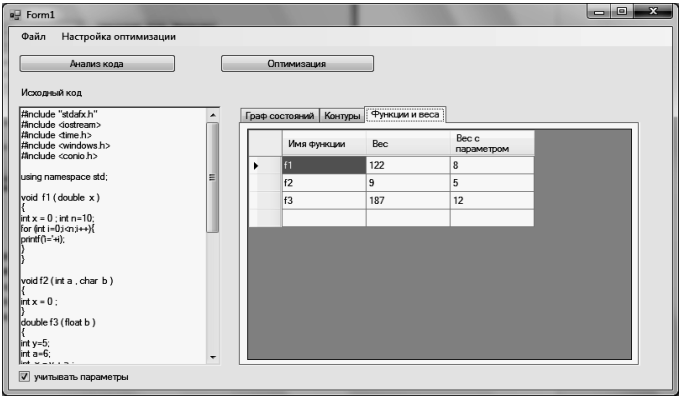


Рис. 8. Пример работы программы. Вкладка «Функции и веса»

В меню «Настройка оптимизации» задается верхняя граница, выделенная на оптимизацию дополнительного объема памяти, а также скорость выделения стека (рис. 9).

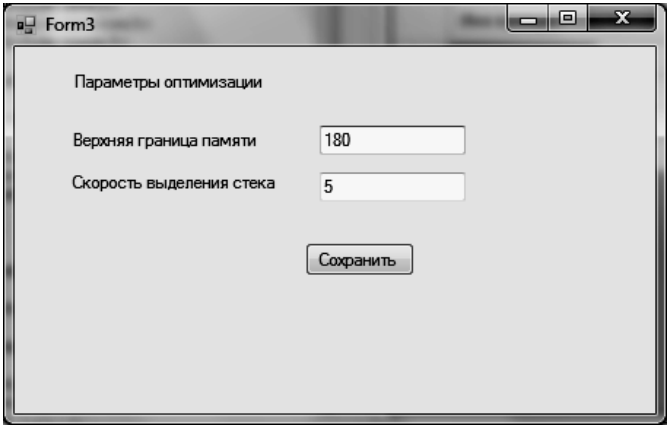


Рис. 9. Пример работы программы. Настройка оптимизации

По нажатию на кнопку «Оптимизация», появляется следующее окно (рис. 10), более темным цветом фона подсвечивается решение.

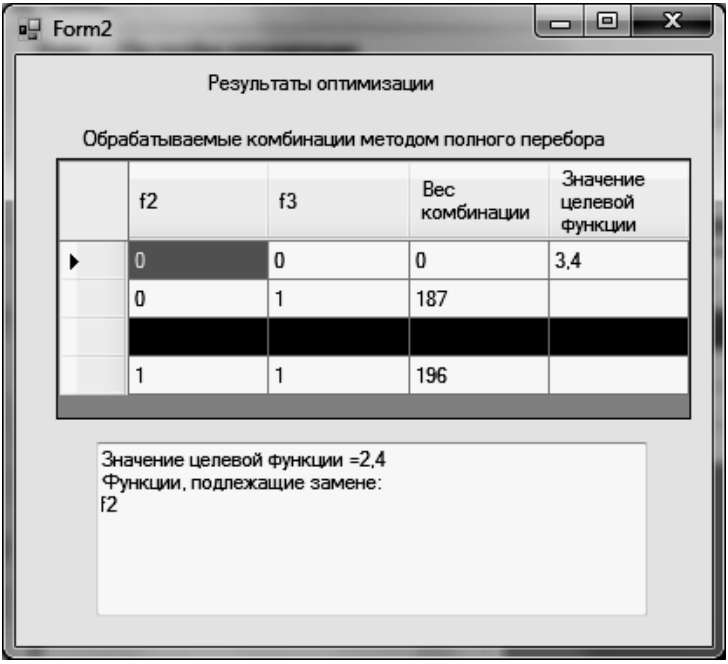


Рис. 10. Пример работы программы. Оптимизация

Полученный программный продукт можно использовать позволяет выполнять макрозамены в соответствии с оптимальной стратегией, сформулированной в задачах (6) и (7) критериями оптимальности в условиях наличия верхней границы объема оперативной памяти, доступной для оптимизации такого рода. Практические наработки могут быть использованы в качестве отдельного продукта, а также в включены в состав комплексного решения, осуществляющего оптимизацию пользовательского исходного кода.

4. Оптимальная стратегия макрозамен конечных программных алгоритмов

Конечные программные алгоритмы характеризуются отсутствием вариантов зацикливания, то есть граф, описывающий алгоритм не со-

держит контуров. Критерием производительности таких программ является время перехода программы из начального состояния в одно из конечных. Пользуясь подходом, аналогичным используемому при формулировке (4), выделим верхнюю границу времени поиска решения и соответствующее подмножество функций в качестве объекта оптимизации и сформулируем игровую модель, описывающую задачу поиска оптимальной стратегии макрозамен, минимизирующих верхнюю границу поиска решения (завершения программы) в виде задачи (8):

$$\begin{cases} F = \max\limits_{x_q \subset X^T} \max\limits_{j} \sum\limits_{k \in \{L_j(0,q)\}} ((1 - z_k) t_k N_k) \to \min \\ \forall (q, j): \sum\limits_{k \in \{L_j(0,q)\}} z_k v_k N_k \le V \\ z_k = 1{,}0 \end{cases}, \qquad (8)$$

где $L_j(0,q)$ – j-й вариант пути из начального состояния в одну из терминальных вершин x_q;

x_q – q-ая терминальная вершина;

X^T – множество терминальных вершин;

t_k – время на вызов k-ой функции;

N_k – количество вызовов k-ой функции на пути $L_j(0, x_i)$;

z_k – булева переменная, равная 1, если для k-ой функции выполняется макрозамена и 0 – в противном случае;

v_k – размер k-ой функции;

V – верхняя граница, выделенная на оптимизацию дополнительного объема памяти

Для перехода к дискретной линейной модели, заменим исходную формулировку двумя следующими (9), (10) задачами, которые решаются последовательно.

Вначале определим верхнюю границу времени однократного зацикливанию, соответствующую такому пути $L_{\max}(0, q^{\max})$ из начального состояния в одно из конечных, для которого выполняется равен-

ство (9) (длина пути определяется суммарным временем на вызов функций, лежащих на пути):

$$\sum_{k\in\{L\max(0,q^{\max})\}} t_k = \max_{x_q\subset X^T} \max_j \Big(\sum_{k\in\{L_j(0,q)\}} t_k \Big) \qquad (9)$$

Далее задачу поиска оптимальной стратегии макрозамен формулируем только для подмножества функций, формирующих найденный контур:

$$\begin{cases} F_2 = \sum_{k\in\{L\max(0,q^{\max})\}} ((1-z_k)t_k N_k) \to \min \\ \forall k: \sum_{k\in\{L\max(0,q^{\max})\}} z_k v_k N_k \le V \\ z_k = 1{,}0 \end{cases}, \qquad (10)$$

где $k \in \{L\max(0, q^{\max})\}$ – это максимальный путь из начального состояния в одно из конечных, соответствующий решению (9).

В случае, если конечный алгоритм пользователя является линейным, задачу можно упростить в виде системы (11):

$$\begin{cases} F_4 = \sum_{k=1}^{M} ((1-z_k)t_k N_k) \to \min \\ \sum_{k=1}^{M} z_k v_k N_k \le V \\ z_k = 1{,}0 \end{cases}, \qquad (11)$$

где M – это количество функций в программе.

Для решения задач (10), (11) можно применить различного рода комбинаторные процедуры. В настоящий момент выполнена программная реализация алгоритма решения данной задачи (11) методом полного перебора.

5. Программная реализация метода макрозамен конечных алгоритмов

5.1. Алгоритм работы программы

Для решения задачи (11) был написан программный продукт, осуществляющий поиск оптимальной стратегии макрозамен. Входными данными оптимизационной системы являются: исходный код программы на c++, верхняя граница памяти и скорость выделения стека. Выходные данные: функции, рекомендуемые для макрозамены. Программа может самостоятельно модифицировать код: объявления функций, соответствующих найденному оптимальному плану $\vec{z}$ модели (11) модифицируются добавлением ключевого слова inline.

Алгоритм включает следующие шаги:

1) лексический и синтаксический анализ исходного кода;

2) нахождение весов для всех функций

3) Поиск оптимальной стратегии макрозамен в соответствии с критериями, сформулированными в задаче (11)

4) Модификация объявлений функций в соответствии с найденным оптимальным планом задачи (11).

5.2. Интерфейс программы

Программная реализация была выполнена на языке C#. Кроме того, для обеспечения удобства пользователя, программный продукт был интегрирован в среду Visual Studio 2013 с помощью технологии «Add-In», позволяющей создавать сторонним разработчикам программы расширения для большинства продуктов Microsoft. Доступ к информации о проекте осуществляется с помощью набора специальных программных интерфейсов – для удобства разработчиков выполненных в виде набора классов, основным из которых является интерфейс «DTE». Одним из свойств данного интерфейса является «ActiveSolutionProjects», в котором хранится список проектов, содержащихся в текущем открытом решении (solution). Каждый элемент этого списка является элементом типа «Project» в свойстве «ProjectItems», которого хранится список файлов с исходными кодами пользователя. Интерфейс «DTE» также позволяет расширить функциональность стандартного интерфейса Visual Studio. В частности, в данной программе была создана дополнительная панель, которая автоматически появляется в нижней части экрана в момент активации оптимизационного дополнения.

В процессе установки оптимизационного дополнения стандартное меню Visual Studio автоматически модифицируется: После запуска Visual Studio, в пункте «Сервис» главного меню становится доступен подпункт «InlineSet» для активации оптимизационной надстройки.

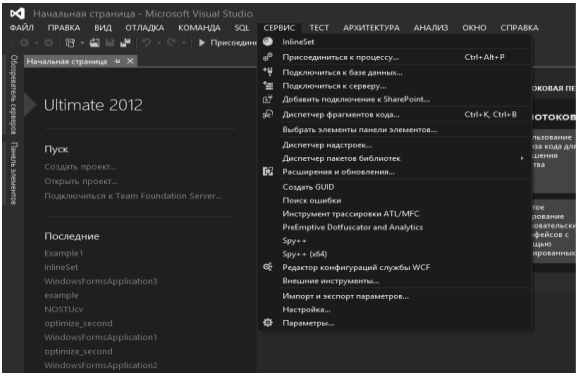


Рис. 11. Меню «Сервис», подпункт «InlineSet», активирующий надстройку

Оптимизационное преобразование выполняется для активного в настоящий момент файла и начинается с загрузки файла с исходным кодом на языке C+. На рисунке 12 видно, изображено окно Visual Studio с активным документом, содержащим код на языке C++.

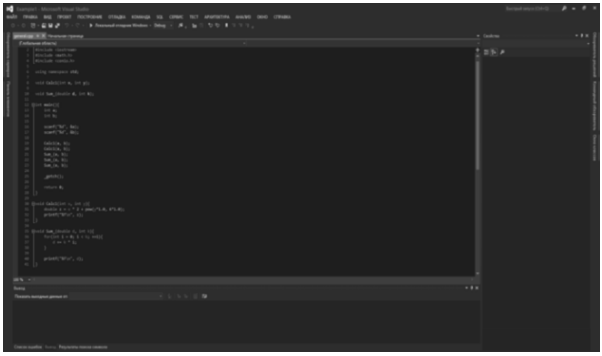


Рис. 12. Открытие файла с исходным кодом на языке C++

Далее следует активировать надстройку (с помощью подпункта «InlineSet» в меню «Сервис»). Непосредственно в момент активации происходит анализ исходного кода – в том числе определяется список

функций, содержащихся в файле, подсчитывается объем операторов в блоке каждой функции (используется для вычисления ограничений) и объем параметров (используется для вычисления времени на передачу управления функции t_k в модели (11)). По окончании анализа оптимизационная надстройка предлагает указать папку, для сохранения данных в файл формата XML (рис. 13).

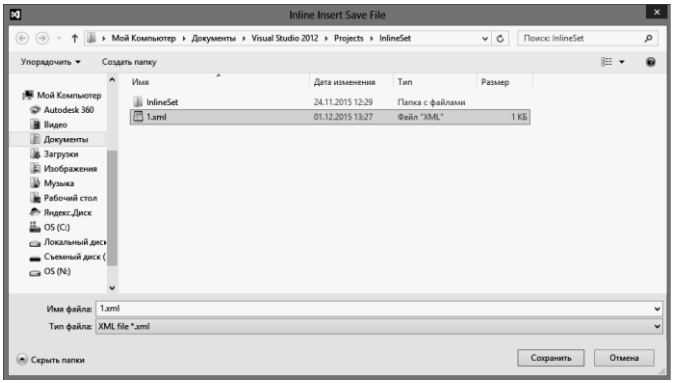


Рис. 13. Пример работы программы. Сохранение в файл

Далее отображается диалоговое окно, в котором пользователю предлагается ввести верхнюю границу объема оперативной памяти «*V*», доступного для макрозамен, а также определить значение «S» – скорости выделения стековой памяти (рис. 14).

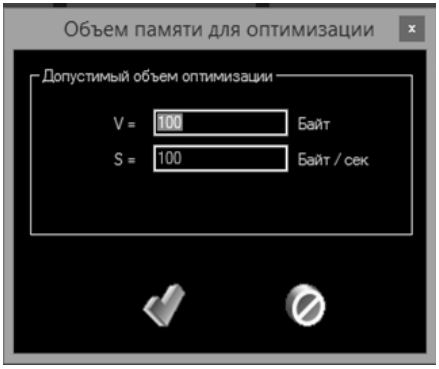


Рис. 14. Пример работы программы. Диалоговое окно «Объем памяти для оптимизации»

Далее, выбрав вкладку «User’s function grid», мы увидим таблицу данных по исходному коду (рис. 15), которая содержит: Имя функции, Проект, Имя файла, Позиция, Количество вызовов, Количество аргументов, Объем памяти, Объем памяти для аргументов и время вызова функции.

Рис. 15. Пример работы программы. Вкладка «User’s function grid»

В соответствии с найденной оптимальной стратегией в исходный код пользователя в активном документе также вносятся изменения с помощью ключевого слова «inline» (рис. 16).

Рис. 16. Пример работы программы. Оптимизированный код

Основным преимуществом данной программной разработки является масштабируемость – функциональность оптимизационного дополнения можно в дальнейшем расширять за счет новых методов оп-

тимизации, соответствующим образом модифицируя и дополняя стандартный интерфейс Visual Studio; в связи с этим прикладная программная разработка будет иметь большое значение и в научном плане – наличие готовой платформы ускорит реализацию и экспериментальную тестирование новых перспективных методов оптимизации.

Литература

1. *Гроппен В. О.* Принципы оптимизации программного обеспечения ЭВМ. Ростов н/Д: Изд-во РТУ, 1993.

2. *Гроппен В. О., Томаев М. Х.* Модели, алгоритмы и средства программной поддержки проектирования оптимальных программных продуктов // Автоматика и телемеханика. 2000. № 11. С 175–183.

3. *Томаев М. Х.* Выбор оптимальной стратегии // Труды молодых ученых. Владикавказ. 2005. № 4. СКГМИ.

Томаев М. Х.,
канд. техн. наук, доцент кафедры
автоматизированной обработки информации
СКГМИ (ГТУ).
e-mail: murat@vosesoftware.com

Горькавая Е. Ю.,
студентка СКГМИ (ГТУ).
e-mail: aiprilly@yandex.ru

Литвинов Д. И.,
студент СКГМИ (ГТУ).
e-mail: litvinov15rus@yandex.ru

CHOOSING THE OPTIMAL STRATEGY OF MACRO SUBSTITUTIONS IN PROGRAMS WRITTEN IN "C ++"

Associate Professor **M. K. Tomaev**,
student **E. J. Gorkavaja**, student **D. I. Litvinov**

An approach to optimize the function call by macro substitution using inline-substitutions. In the optimality criteria formulated different strategies for macro substitutions applied to looped and end algorithms. It describes methods for the solution, and also features a software implementation. The resulting studies program product allows us to increase the effectiveness of software systems as implemented in the models and approaches may be used in complex optimization means as one of the tools.

Keywords: *macro substitution, inline-substitutions, optimality criteria, complex optimization tools, C++.*

УДК 004.021

Томаев М. Х.,
Джабиева З. Ю.,
Козаева А. В.

МОДЕЛИ И ТЕХНОЛОГИИ ОПТИМАЛЬНОЙ ДЕКОМПОЗИЦИИ ФУНКЦИЙ ПРОГРАММНОЙ СИСТЕМЫ НА DLL-БИБЛИОТЕКИ

В работе предлагается подход к оптимальной декомпозиции программных систем с помощью технологии DLL. Формулируются критерии оптимальности различных стратегий декомпозиции применительно к зацикленным и склонным к зацикливанию пользовательским алгоритмам. Предлагаются алгоритмы решения оптимизационных задач, приводится описание программного продукта, созданного на базе разработанной технологии. Результаты исследований – модели и алгоритмы, а также программный пакет оптимизации – могут быть использованы в прикладном программировании, а также стать базой для дальнейших исследований в рамках проблематики оптимизации программного обеспечения.

Ключевые слова: *программа, оптимизация, декомпозиция, память, модель, производительность, зацикленные алгоритмы, склонные к зацикливанию алгоритмы*.

1. Организация взаимодействия подсистем с помощью технологии DLL

С развитием технологий и средств разработки ПО степень информатизации всех отраслей человеческой деятельности стремительно возрастает. Объем исполняемого кода сложных программных комплексов зачастую может превышать десятки гигабайт. В случае, когда подобные сверхбольшие системы функционируют в условиях ограниченного объема оперативной памяти, актуальной становится задача эффективной декомпозиции (разбиения) программной системы на модули. Задача организации взаимодействия модулей на платформе Windows решается достаточно просто благодаря поддержке ядром ОС технологии DLL (dynamiclinklibrary) [3].

Динамически подключаемая библиотека – это особый вид объектного кода, который после запуска выполняется в адресном пространстве процесса, который его вызвал. Разработчик самостоятельно выбирает те функции и переменные, которые будут доступны вызывающему процессу, и объявляет их экспортируемыми либо с помощью специального текстового файла с расширением «.def», либо с помощью особых конструкций языка – в частности, в языке C++ предусмотрено ключевое слово *_declspec*(*dllexport*), которое указывается перед объявлением экспортируемых функций. Существуют два способа подключения DLL-библиотек к основному (вызывающему) процессу: «раннее связывание» и «позднее связывание» [3].

Технология раннего связывания заключается в том, что в исходный код вызывающего процесса включается информация о таблице адресов экспортируемых библиотекой функций. При запуске основного процесса все DLL-библиотеки, подключенные таким способом, будут загружены в оперативную память автоматически. Таким образом, данный способ не позволяет управлять логикой загрузки или выгрузки вспомогательных модулей.

Позднее связывание – это способы программной загрузки библиотек непосредственно при работе основной программы, реализуемые с помощью функций APIWindowsLoadLibary и FreeLibrary [3], соответственно, для загрузки библиотеки в оперативную память и выгрузки из нее. Данный способ связывания подходит для реализации разработчиком различных стратегий загрузки вспомогательных подсистем.

В рамках проведенных исследований были разработаны математические модели оптимальной декомпозиции зацикленных и склонных к зацикливанию программных алгоритмов, а также предложены алгоритмы решения оптимизационных задач.

2. Модели оптимальной декомпозиции зацикленных программных алгоритмов

В сложных программных системах, алгоритм которых не имеет вариантов завершения, программа может зациклиться в любом из контуров. В [1] и [2] предложен вариант формулировки критерия оптимальности декомпозиции, основанный на двух положениях:

1) для оценки производительности зацикленных систем принимается показатель времени однократного зацикливания;

2) выбор оптимальной декомпозиции осуществляется только на участке программы, соответствующем максимальному либо мини-

мальному (по времени однократного зацикливания) контуру на графе переходов состояний программы – выбор одной из двух перечисленных стратегий часто является субъективным – зависит от психологии разработчика.

Основным отличием предлагаемого подхода от технологии, описанной в [1] и [2], является то, что декомпозиция выполняется на более высоком уровне – состав каждой из функциональных DLL-подсистем выбирается из набора существующих функций, но содержимое каждой функции и порядок их вызова не пересматривается. В общем виде задача оптимальной декомпозиции, минимизирующей время однократного зацикливания, может быть представлена в следующем виде:

$$\begin{cases} F_1 = \max\limits_{a_k \in A(G)} \sum\limits_{j \in H(a_k)} t_j sign \sum\limits_{i}^{n} z_{ij} \to \min \\ \forall i, \sum\limits_{j} z_{ij} = 1 \\ \max\limits_{j} (v_j sign \sum\limits_{i} z_{ij}) \le V \\ z_{ij} = 1,0; i = 1,2,..,n;\ j = 1,2,... \end{cases}, \qquad (1)$$

где z_{ij} – булева переменная, равная 1, если *i*-ая функция размещена в *j*-м DLL-модуле, и нулю – в противном случае;

v_j – размер *j*-го варианта подпрограммы (DLL-модуля);

t_j – время загрузки *j*-го варианта подпрограммы (DLL-модуля);

V – верхняя граница доступного объема оперативной памяти;

a_k – *k*-й контур, принадлежащий множеству $A(G)$ на графе, описывающем алгоритм пользователя;

n – число функций;

$H(a_k)$ множество индексов вариантов таких DLL-модулей, которые включают одну или несколько функций, лежащих на *k*-м контуре.

Постановка задачи (1) представляет собой игровую модель (антагонистическая позиционная игра 2 лиц с полной информацией и нулевой суммой [1]). Для того чтобы перейти к дискретной модели разобьем задачу (1) на два этапа. Задачей первого этапа станет поиск контура, соответствующего верхней границе однократного зацикливания, т. е. такого контура $a_{\max}$, для которого выполняется равенство (2):

$$\sum_{i \in H^f(a_{\max})} t_i^f = \max_{a_k \in A(G)} \sum_{i \in H^f(a_k)} t_i^f , \qquad (2)$$

где t_i^f – время работы *i*-й функции;

$H^f(a_k)$ – множество индексов функций принадлежащих *k*-му контуру.

Задача заключительного этапа – поиск оптимальной декомпозиции функций, входящих в состав найденного максимального контура (3):

$$\begin{cases} F_2 = \sum\limits_{j \in H(a_{\max})} t_j sign \sum\limits_{i}^{n} z_{ij} \to \min \\ \forall i, \sum\limits_{j} z_{ij} = 1 \\ \max\limits_{j} (v_j sign \sum\limits_{i} z_{ij}) \le V \\ z_{ij} = 1,0; i = 1,2,..,n; \; j = 1,2,... \end{cases} , \qquad (3)$$

Для решения данной задачи можно использовать различного рода комбинаторные процедуры: полный перебор, методы типа ветвей и границ, динамическое программирование.

3. Модели оптимальной декомпозиции склонных к зацикливанию программных алгоритмов

При оценке производительности склонных к зацикливанию алгоритмов следует учитывать их структуру. Так как подобные алгоритмы содержат непустое множество контуров, то одним из критериев качества может являться время однократного зацикливания. В то же время наличие вариантов завершения позволяет использовать второй критерий – время поиска решения. В общем случае, минимизация времени однократного зацикливания представляется более предпочтительной, поэтому поиск оптимальной декомпозиции склонных к зацикливанию алгоритмов на DLL-модуле можно представить в виде двух последовательно решаемых задач, первая из которых представляет собой модель (3), а вторая осуществляет поиск оптимальной декомпозиции, минимизирующей верхнюю границу времени поиска решения (завершение алгоритма), причем решение второй задачи (модель (4)) не должно противоречить решению (3).

$$\begin{cases} F_3 = \sum\limits_{j\in\{L\max(0,q^{\max})\}} t_j sign \sum\limits_{i}^{n} z_{ij}^2 \to \min \\ \forall i, \sum\limits_{j} z_{ij}^2 = 1 \\ \max\limits_{j}(v_j sign \sum\limits_{i} z_{ij}^2) \le V \\ \forall j \in H(a_{\max}), \forall i = 1,n : z_{ij}^2 = z_{ij} \\ z_{ij}^2 = 1,0; i = 1,2,..,n;\ j = 1,2,... \end{cases}, \quad (4)$$

Непротиворечивость решения (4) и соответствующего оптимального вектора переменных $\overrightarrow{Z^2}$ решению, полученному при решении (3) (вектор переменных $\vec{Z}$) формулируется в третьем ограничении.

Аналогично подходу, использованному при формулировке задачи (3), при формулировке задачи (4) предполагается, что выбор оптимальной декомпозиции осуществляется на предварительно найденном максимальном пути $L\max(0, q^{\max})$:

$$\sum_{i\in H^f(L\max(0,q^{\max}))} t_i^f = \max_{x_q \subset X^T} \max_{j} \sum_{i\in H^f(L^j(0,q))} t_i^f \ . \quad (5)$$

Так как склонные к зацикливанию алгоритмы являются наиболее распространенными видами пользовательских алгоритмов, то создание инструментальных средств, учитывающих особенности их функционирования при оптимизации исходных кодов, является актуальной задачей.

4. Программная реализация метода оптимальной декомпозиции зацикленных программных алгоритмов

Для решения задачи оптимальной декомпозиции зацикленных программных алгоритмов (модель (3)) было разработано приложение на языке C# с использованием виртуальной платформы Framework 4.5. Интерфейс программы представляет собой диалоговое окно, содержащее элементы управления для загрузки документа пользователя с исходным кодом на языке C++ (рис. 1), а также ряд окон для просмот-

ра списка функций, последовательности вызовов функций и оптимальной декомпозиции, предложенной программой.

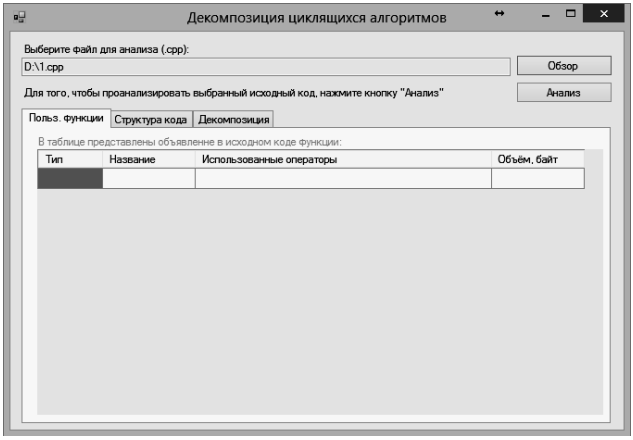


Рис. 1. Внешний вид интерфейса программы

После загрузки кода пользователя (кнопка «Обзор»), пользователю необходимо активировать этап анализа кода с помощью соответствующей кнопки («Анализ»). При этом на закладке «Польз. функции» (рис. 2) будет указан список функций, для каждой из которых будут перечислены элементы кода, идентифицированные программой как часть функции. В последней колонке списка приводится рассчитанный программой размер кода.

На рисунке 2 изображен список функций, сгенерированный на основе анализа следующего файла (Листинг 1):

```
void f1(int &x);
void f2();
void f3(int p1, int p2);
void main()
{
    int x,k,z;
    x = sin(10);
    m10: f1(x);
    k = x + 1;
    m20: f2();
    z = k * k;
```

```
    if (z < 5) goto  m10;
    f3(k, z);
    goto m20;
}

void f1(int &x)
{
    int z;
    z = x * x;
    x = z – 5;
}
void f2()
{
    int k = sin(10);
}
void f3(int p1, int p2)
{
    int z;
    z = p1 + p2;
}
```

Листинг 1. Исходный код программы пользователя на языке C++.

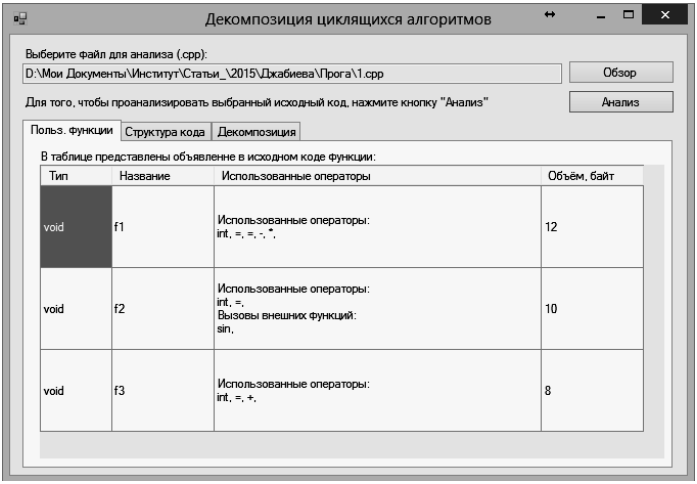


Рис. 2. Список функций с указанием объема кода, а также входящих в них операторов

На этапе анализа кода программой также составляется алгоритм вызовов функций, который воспроизводится на закладке «Структура кода» в виде матрицы переходов (рис. 3), описывающей графы состояний программы, где дуги представляют собой функции, переводящие программу из одного состояния в другой. Для упрощения анализа кода операторы «goto» включаются в матрицу переходов, однако, в отличие от функций, имеют нулевой вес (дуги 3–1 и 4–2 в данном примере).

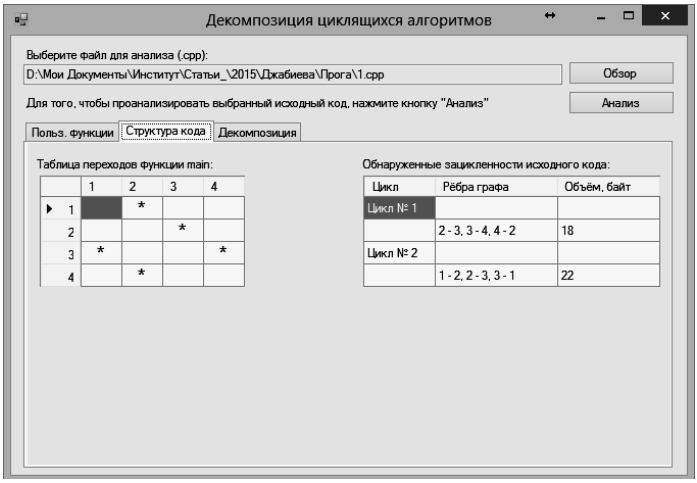


Рис. 3. Матрица переходов, описывающая последовательность вызова функций

На данной закладке также отображается список контуров, с указанием дуг, составляющих каждый контур, и суммарного объема входящих в него функций.

Для того, чтобы выполнить поиск оптимальной декомпозиции, необходимо перейти на закладку «Декомпозиция» (рис. 4), ввести значение верхней границы доступного объема оперативной памяти и нажать кнопку «Распределить».

Так как оба контура в алгоритме программы имеют одинаковую длину, т. е. оба соответствуют верхней границе времени однократного зацикливания, то программой предлагаются два альтернативных решения задачи (3):

1) Функции *f*2(), *f*3(), составляющие контур (2–3, 3–4, 4–2) размещаются в одной DLL, так как их суммарный вес (10 + 8) не превышает ограничения (18).

2) Функции *f*1() и *f*3(), составляющие контур (1–2, 2–3, 3–1), размещаются каждая в своей DLL, так как их суммарный вес (12 + 10) не позволяет их объединение в одном DLL-модуле с учетом имеющегося ограничения.

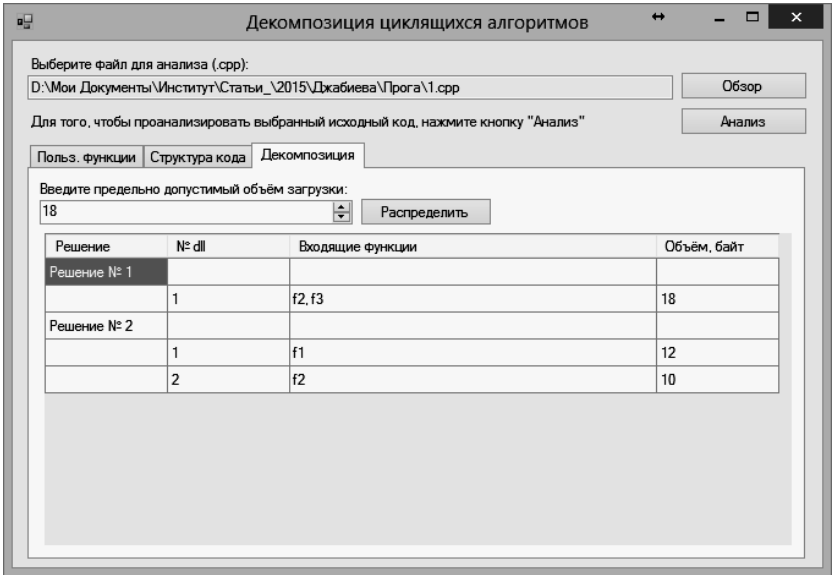


Рис. 4. Рекомендованные программой стратегии декомпозиции

Полученный программный продукт представляет собой важный прикладной результат в рамках работ по развитию технологий оптимизации программных продуктов, а также инструментальных средств их поддержки.

Литература

1. *Гроппен В. О.* Принципы оптимизации программного обеспечения ЭВМ. Ростов н/Д: Изд-во Ростовского университета, 1993.

2. *Гроппен В. О., Томаев М. Х.* Модели, алгоритмы и средства программной поддержки проектирования оптимальных программных продуктов // Автоматика и телемеханика. 2000. № 11. С. 175–183.

3. What is a DLL? – Microsoft Support. Эл. адрес: http://support.microsoft.com/ru-ru/kb/815065.

Томаев М. Х.,
канд. техн. наук, доцент кафедры
автоматизированной обработки информации
СКГМИ (ГТУ).
e-mail: murat@vosesoftware.com

Джабиева З. Ю.,
студентка СКГМИ (ГТУ).
e-mail: zarina9292.92@mail.ru

Козаева А. В.,
студентка СКГМИ (ГТУ).
e-mail: Kozaeva.Azau2015@yandex.ru

MODELS AND TECHNOLOGIES OF OPTIMAL DECOMPOSITION OF FUNCTIONS IN THE SOFTWARE SYSTEM OF DLL-LIBRARY

Associate Professor **M. K. Tomaev**,
student **Z. Y. Dzhabieva**, student **A. V. Kozaeva**

This paper proposes an approach to the optimal decomposition of software systems using technology DLL. In this work are optimality criteria formulated for different strategies applied to the decomposition of fixated and prone to loops custom algorithms. Also in this paper are presented algorithms for solving optimization problems, a description of the software product, created on the basis of the DLL technology. Research results – models and algorithms as well as optimization software package – can be used in application programming, and serve as a basis for further studies within the perspective of optimization software.

Keywords: *program, optimization, decomposition, memory, model, performance, fixated algorithms, inclined to looping algorithms.*

ОБРАБОТКА ИЗОБРАЖЕНИЙ

УДК 004.896

Проскурин А. Е.,
Дзарасов Д. А.

ПОИСК ОПТИМАЛЬНОГО КОЛИЧЕСТВА ТОЧЕК В АЛГОРИТМЕ ВЫДЕЛЕНИЯ СКЕЛЕТА РАСТРОВОГО ИЗОБРАЖЕНИЯ

Рассматривается задача поиска оптимального количества точек при выделении скелета растрового изображения.

***Ключевые слова**: скелетизация изображений, волновой алгоритм, растровое изображение, сферическая волна, оптимизация.*

Введение

В настоящее время актуальна задача разработки методов распознавания образов, так как имеет место большое количество задач, связанных с анализом формы объектов. Проблема распознавания формы пространственных объектов является весьма актуальной в различных приложениях [1]. Задача распознавания формы может вызывать существенные трудности. Кроме того, зачастую требуется построить алгоритмы, работающие в режиме реального времени, что накладывает дополнительные ограничения на эффективность.

В задачах анализа формы плоских изображений хорошо зарекомендовало себя применение аппарата непрерывных скелетов. Скелетом двумерной области обычно называется ее серединная ось – множество центров, максимально вписанных в эту область кругов. Скелет плоской области представляет собой планарную укладку некоторого графа, удачно схватывающего основные геометрические свойства фигуры. Визуально скелет выглядит как утончение исходной формы до набора одномерных линий. Извлекать из такого графа признаковую информацию для задач распознавания и классификации значительно проще, чем из исходного гранично-контурного описания двумерной области [2].

Для скелетизации растровых изображений применяются алгоритмы Зонга – Суня, волновой алгоритм [3].

Волновой метод отслеживания центральной линии и соединения отрезков

Метод заключается в анализе пути прохождения сферической волны по изображению. На каждом шаге анализируется смещение центра масс точек, образующих новый шаг (генерацию) волны, относительно его предыдущих положений.

Алгоритм:

1) создаем пустой стек для хранения генераций волны;
2) заносим в него любую точку изображения как генерацию волны;
3) пока стек не пуст продолжаем шаги 4–8;
4) выбираем генерацию волны из стека;
5) продолжаем волну из выбранной точки изображения, пока не произойдет разделение волны на полуволны или затухание волны;
6) если произошло затухание волны, то пройденный путь является кривой, заносимой в граф (возможно замкнутой, если затухание волны произошло на границе с помеченной волной областью); переходим к п. 3;
7) если волна разделилась на полуволны, то мы нашли место соединения двух отрезков, и в граф заносится пройденный путь. В стек заносим обе полуволны;
8) переходим к п. 3.

Волновые алгоритмы часто используются в компьютерных играх для определения минимального расстояния от одного объекта до другого, в ограниченном дискретном пространстве. Для этого в исходной точке генерируется волна, распространяющаяся по определенным законам, помечающая пройденные точки номером шага. Процесс заканчивается по достижении целевой точки. Номер шага, которым помечена целевая точка, и будет расстоянием от исходной до целевой точки [4].

После получения скелета его необходимо оптимизировать, уменьшив количество вершин скелета (рис. 1).

Метод заключается в переборе всех возможных комбинаций вершин, включенных в скелет, и уменьшении суммы площадей треугольников, изображенных на рис. 2.

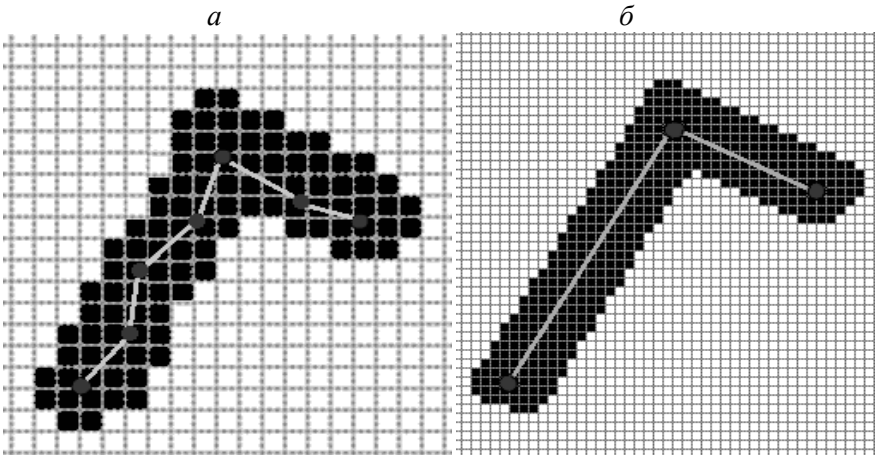


Рис. 1. Скелет, полученный с помощью волнового алгоритма (*а*), оптимизированный скелет (*б*)

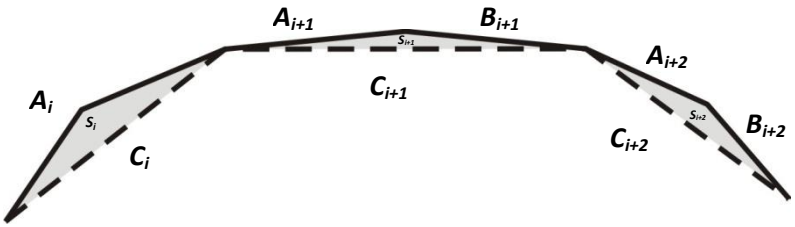


Рис. 2. Иллюстрация поиска оптимального скелета

Жирной линией показан полученный скелет, пунктирная линия – это оптимизированный скелет, а S_i обозначает площадь треугольника.

Формальная постановка задачи

$$\begin{cases} \sum_{j}^{m} Z_j \cdot \sum_{i}^{n} \sqrt{\frac{A_{ij}+B_{ij}+C_{ij}}{2} \cdot \left(\frac{A_{ij}+B_{ij}+C_{ij}}{2} - A_{ij}\right) \cdot \left(\frac{A_{ij}+B_{ij}+C_{ij}}{2} - B_{ij}\right) \cdot \left(\frac{A_{ij}+B_{ij}+C_{ij}}{2} - C_{ij}\right)} \rightarrow min \\ \forall j: Z_j = 1{,}0. \\ \sum_{j}^{m} Z_j = 1, \end{cases}$$

где A_{ij}, B_{ij}, C_{ij} – стороны треугольников,
Z_j – булева переменная, обозначающая выборку.

Заключение

В данной статье рассмотрен метод, осуществляющий поиск оптимального количества точек в алгоритме выделения скелета растрового изображения. В статье представлена формальная постановка задачи и содержательное описание метода.

Литература

1. *Шикин Е. В., Боресков А. В.* Компьютерная графика. М.: Мир, 1982.
2. *Местецкий Л., Хромов Д.* Криволинейные скелеты трехмерных форм // Доклады 15-й Всероссийской конференции «Математические методы распознавания образов». М., 2011.
3. *Денисов И., Кузьмин Е.* Эффективный алгоритм построения остова растрового изображения // Доклад на конференции Графикон-99. Москва, 26 августа, 1999 г.
4. *Местецкий Л. М.* Непрерывный скелет бинарного изображения. http://ocrai.narod.ru

Проскурин А. Е.,
канд. техн. наук, доцент кафедры
автоматизированной обработки информации
СКГМИ (ГТУ)
e-mail: alexander@skgmi-gtu.ru

Дзарасов Д. А.,
студент СКГМИ (ГТУ)
e-mail: david-dzarasov@mail.ru

SEARCH FOR THE OPTIMAL NUMBER OF POINTS IN THE ALGORITHM SELECTION SKELETON BITMAP

Associate Professor **A. E. Proskurin,** student **D. A. Dzarasov**

The problem search for an optimal number of points in the allocation of the skeleton of the bitmap.

Keywords*: skeletonization image wave algorithm, a bitmap, a spherical wave, optimization.*

УДК 004.896

Проскурин А. Е.,
Тотров Д. В.

ПРИМЕНЕНИЕ МЕТОДА ЭТАЛОНОВ В ЗАДАЧЕ ПОИСКА КОНТУРОВ ЗАШУМЛЕННЫХ ОБРАЗОВ С ПОМОЩЬЮ НЕЙРОННОЙ СЕТИ

Статья посвящена применению метода эталонов для разрешения задачи распознавания образов на зашумленных изображениях с помощью нейронной сети. Рассматривается, в частности, применение метода эталонов к данной задаче, постановка задачи, описание алгоритма, графический пример работы.

Ключевые слова: *метод эталонов, нейронные сети, распознавание образов, зашумленные изображения.*

Введение

Методы распознавания образов представляют собой наиболее математически зависимый раздел теории искусственного интеллекта, в котором решаются задачи, связанные с классификацией объектов произвольной природы. Распознавание образов – одна из тех задач, которые постоянно решаются «естественным» интеллектом. Поэтому усилия ученых уже на протяжении полувека направлены на разработку методов и алгоритмов «автоматического» решения этой задачи. Распознавание образов в той или иной конкретной ситуации связано с учетом неопределенностей различной природы. В простейшем случае, когда образы однозначно определяются конечным набором признаков, границы классов точно описываются, а сами классы не пересекаются, степень неопределенности можно считать минимальной. В данной статье рассматривается задача иного класса, в которой определение границ классов затруднено различными шумами.

Изначально поданное изображение для опознания образов (классов) необходимо обработать. Кластеризация [3] – разбиение множества входных сигналов на классы, при том, что ни количество, ни признаки классов заранее не известны. После обучения такая сеть способна определять, к какому классу относится входной сигнал. Сеть также может сигнализировать о том, что входной сигнал не относится

ни к одному из выделенных классов – это является признаком новых, отсутствующих в обучающей выборке, данных. Таким образом, подобная сеть может выявлять новые, неизвестные ранее классы сигналов. Соответствие между классами, выделенными сетью, и классами, существующими в предметной области, устанавливается человеком.

Наиболее удобным математическим описанием [2] считается векторное описание образов. В этом случае каждому образу X ставится в соответствие некоторый вектор $x = (x, x1, \ldots, n)$. Такое векторное пространство называется пространством признаков. Как правило, это пространство является конечномерным и метрическим. Если признаки такого пространства являются вещественными величинами, то такое пространство изоморфно метрическому пространству $R(n, n)$ – размерность пространства признаков.

Рис. 1. Пример зашумленного изображения

Рисунок 1 является типичным примером поставленной задачи. Изображение достаточно зашумленное и тяжело поддается обработке. Линией примерно выделен получаемый контур.

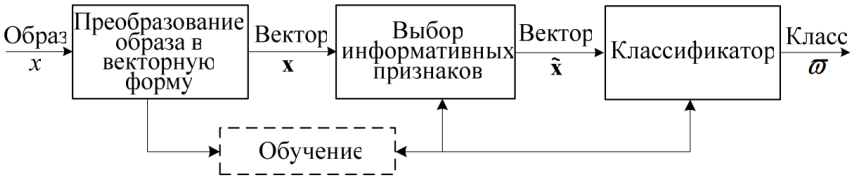


Рис. 2. Общая схема системы распознавания образов

Поиск оптимального решения интерпретируется как выбор некоторого объекта, удовлетворяющего определенным условиям, связанным с расстояниями до эталонов, на множестве такого рода объектов. В зависимости от рассматриваемой предметной области под объектом ниже понимается:

- сочетание значений критериев экстремальной оптимизационной задачи,
- претендент на призовое место, избираемый голосованием,
- компонента некоторого счетного конечного множества, все элементы которого подлежат упорядочению на основе экспертных оценок [1].

При этом полагаем, что на множестве объектов можно выделить либо создать дополнительно два объекта – наилучший и наихудший. Одним из критериев качества анализируемого объекта, обозначаемым ниже символом Δ, может служить степень близости его к наилучшему объекту, другим, обозначаемым далее как θ, – его удаленность от наихудшего объекта. Естественно, что $\Delta \rightarrow \min$, а $\theta \rightarrow \max$. Недостатком такого подхода является его «однобокость», т. к. учитывается расстояние только до одного из эталонов.

Поскольку характеристики объектов и определение расстояния между ними зависят от конкретной решаемой задачи, используемые ниже обозначения привязаны к соответствующим предметным областям, общими для которых являются приведенные выше обозначения критериев и определенные далее обозначения эталонов:

a – объект, обладающий наилучшими характеристиками;

b – объект, характеристики которого являются наихудшими.

Формальная постановка задачи

$$\begin{cases} \sum_{j=1}^{m} Z_j \cdot \sum_{i=1}^{n} \sqrt{(x_i - b_{i,j})^2} \rightarrow \min \\ \sum_{j=1}^{m} Z_j < 1 \\ \forall j: Z_J = 1{,}0 \\ x_i \epsilon X, \qquad 1 < i < n \\ b_{i,j} \epsilon Bj, \qquad 1 < j < m, \end{cases} \tag{1}$$

где X – вектор характеристик изображения проверяемого объекта;

x_i – i-я компонента вектора X;

$b_{i,j}$ – i-я компонента вектора B_j;

B_j – вектор характеристик j-го эталона;

n – число компонент вектора X;

m – число эталонов.

Задача усложняется тем, что под входными значениями подразумеваются неточные или искаженные значения растрового изображения, появляющиеся из-за цифрового шума — дефект изображения, вносимый фотосенсорами и электроникой устройств (рис. 3–4).

а *б*

Рис. 3. Оригинальное (*б*) и зашумленное (*а*) изображения

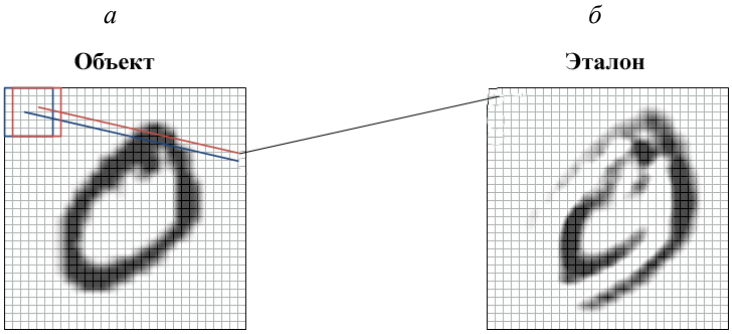


Рис. 4. Иллюстрация попиксельного сравнения изображения (*а*) объекта с эталоном (*б*)

Заключение

Данная статья раскрывает лишь часть объемной темы искусственного интеллекта. В ней впервые рассмотрена возможность применения метода эталонов для задачи распознавания зашумленных образов с помощью нейронной сети, с приведением математической модели возможного решения задачи. Дальнейшее развитие этого направления может опираться на минимизацию суммарного времени, включающего обучение и распознавание; а также на повышение качества распознавания.

Литература

1. *Будаева А. А., Гроппен В. О.* Принятие решений: теория, технология, приложения: Учебное пособие. Владикавказ: Фламинго, 2010. 184 с.

2. *Лепский А. Е, Броневич А. Г.* Математические методы распознавания образов: Курс лекций. Таганрог, 2009.

3. *Хант Э.* Искусственный интеллект. М.: Мир, 1978.

Проскурин А. Е.,
канд. техн. наук, доцент кафедры
автоматизированной обработки информации
СКГМИ (ГТУ)
e-mail: alexander@skgmi-gtu.ru

Тотров Д. В.,
студент СКГМИ (ГТУ)
e-mail: dampe@mail.ru

THE APPLICATION OF THE METHOD STANDARDS IN THE TASK OF FINDING THE CONTOURS OF NOISY IMAGES USING A NEURAL NETWORK

Associate Professor **A. E. Proskurin,** student **D. V. Totrov**

The article is devoted to the application of standards to solve the problem of noisy images recognition using a neural network. We consider, in particular, the application of this method to the given problem, formal its description, description of solving it algorithm and graphic example.

Keywords: *method of standards, neural networks, pattern recognition, noisy images.*

МОДЕЛИРОВАНИЕ СИСТЕМ

УДК 531.5

Гроппен В. О.,
Проскурин А. Е.

КОНТРОЛЬ СИЛ ГРАВИТАЦИОННОГО ВЗАИМОДЕЙСТВИЯ В ВАКУУМЕ: МОДЕЛИРОВАНИЕ И РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ*

В статье представлены результаты первых экспериментов по контролю сил гравитационного взаимодействия в вакууме с помощью развернутых конденсаторов, базой которых является альтернативная Стандартной Модели интерпретация закона Хаббла. Показана независимость предложенного подхода от эффекта Бифельда – Брауна. Полученные результаты дают шанс на создание принципиально новых двигателей и систем управления, не обладающих подвижными частями.

***Ключевые слова**: гравитация, контроль, развернутый конденсатор, высокое напряжение, вакуум.*

1. Введение

Эволюцию наших представлений о Вселенной можно интерпретировать постепенным дрейфом от наивных сочетаний «очевидных» понятий к постепенно усложняющимся математическим моделям. Иллюстрацией служит последовательная замена геоцентрической модели, основой которой служила плоская, покоящаяся на спинах слонов Земля, гелиоцентрической моделью, на смену которой пришла предложенная в 1927 году Джорджем Леметром идея расширяющейся Вселенной, возникшей в результате Большого Взрыва. Эта теория лежит в основе принятой сегодня Стандартной Модели благодаря экспериментальному закону, открытому Эдвином Пауэллом Хабблом в 1929 году, базой которого послужило объяснение красного смещения

* Работа выполнена в рамках гос. заказа № 3760 Министерства образования и науки РФ

спектров галактик, их «разбеганием», фиксируемым с помощью эффекта Доплера [1].

Параллельно с эволюцией моделей Вселенной развивались связанные с ними модели, описывающие силы гравитации. Одна из первых математических моделей такого рода была предложена Исааком Ньютоном в 1667 году [2]. В 1748 году Лесаж предложил теорию гравитационного взаимодействия, базирующуюся на взаимодействии массивных тел и сталкивающихся с ними мельчайших частиц, движущихся во всех направлениях с высокими скоростями [3]. Эта теория не подтвердилась: в результате этих столкновений физические тела должны были нагреваться, что не соответствует наблюдениям; и в 1915 году Альберт Эйнштейн предложил теорию гравитации, базирующуюся на идеях теории относительности и стабильности Вселенной [4]. В 1931 году Альберт Эйнштейн, под влиянием закона Хаббла, был вынужден внести коррективы в свою теорию, заметив, что эти изменения имеют место, только если использование эффекта Доплера для объяснения закона Хаббла справедливо [5]. Альтернативное объяснение закона Хаббла, основанное на вариабельности эталонов, используемых для измерения расстояний и времени, было предложено в 2009–2013 годах в [6–8]. Следствием этого явился вывод о спонтанной потере массы всеми физическими телами, использованный в [8–12] для объяснения сил гравитационного взаимодействия и их контроля.

2. Контроль сил гравитационного взаимодействия с помощью развернутых конденсаторов

Далее полагаем, что спонтанная эмиссия массы физическим телом «*A*» в направлении φ, величина $\left.\frac{dm_A}{dt}\right|_{\varphi}$ которой зависит от разности плотностей тела «*A*», ρ_A и ближайшей к его поверхности в направлении φ плотности окружающей среды ρ_φ. В первом приближении эта зависимость имеет вид:

$$\left.\frac{dm_A}{dt}\right|_{\varphi} = f_A \cdot (\rho_A - \rho_\varphi), \qquad (1)$$

где f_A – неизвестная константа, характеризующая тело «*A*».

Если тело «*A*» находится в изотропной среде, т. е. $\forall \varphi : \rho_\varphi = const = \rho_0$, то общая потеря массы в единицу времени этим телом выражается зависимостью вида:

$$\frac{dm_A}{dt} = 4\pi \cdot f_A \cdot (\rho_A - \rho_0). \qquad (2)$$

Учитывая, что в вакууме имеет место равенство [11]:

$$\frac{dm_A}{dt} = -H \cdot m_A, \qquad (3)$$

где H – постоянная Хаббла, а величина f_A равна:

$$f_A = \frac{H \cdot \vartheta_A}{4\pi}, \qquad (4)$$

где ϑ_A – объем тела «*A*».

В соответствии с уравнением Мещерского [13], реактивная сила $F_{-\varphi}$, действующая на тело «*A*» в направлении, противоположном φ, с учетом релятивистской зависимости массы от скорости, имеет вид:

$$F_{-\varphi} = \frac{H \cdot \vartheta_A}{4\pi} \frac{(\rho_A - \rho_\varphi)}{\sqrt{\frac{1}{V_\varphi^2} - \frac{1}{c^2}}}; \qquad (5)$$

где V_φ – скорость, теряемая телом «*A*» в направлении φ массы;

c – скорость света в вакууме.

Очевидно, что если выполняются условия:

$$\begin{cases} V_\varphi = V_{-\varphi} \\ \rho_\varphi = \rho_{-\varphi}. \end{cases}$$

то равнодействующая всех реактивных сил, вызванных потерей массы телом «*A*», равна нулю.

Для того, чтобы нарушить это равновесие, были использованы «развернутые конденсаторы» [11, 12], т. е. такие конденсаторы, все электроды которых принадлежат одной плоскости (см. рис. 2 ниже). Энергия $E_к$, запасенная таким конденсатором, влияет на плотность окружающей его среды и т. о., нарушает баланс сил реакции помещенного под него тела «*A*» (см. рис. 1),

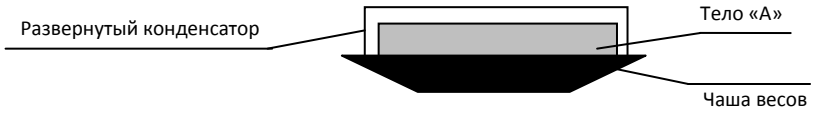


Рис. 1. Взаимное расположение в ходе экспериментов развернутого конденсатора, тела «*A*» и чаши весов

что приводит к возникновению подъемной силы ΔF:

$$\Delta F = F_{-\varphi} - F_{\varphi} \approx \frac{H \cdot \boldsymbol{\upsilon}_{\text{A}}\, \rho_{\text{e}}}{4\pi\sqrt{\dfrac{1}{V^2} - \dfrac{1}{c^2}}}, \tag{6}$$

Энергия конденсатора определяется выражением:

$$E_к = 0{,}5CU^2, \tag{7}$$

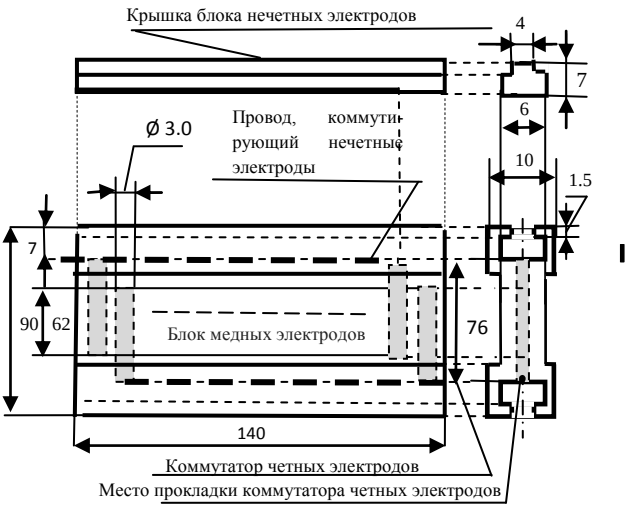
где C – емкость конденсатора,
U – приложенное напряжение.
Эквивалентная масса равна:

$$m_e = \frac{CU^2}{2c^2}. \tag{8}$$

С учетом объема конденсатора V_c , (6) преобразуется к виду:

$$\Delta F = \frac{HCU^2}{8\pi c^2\sqrt{\dfrac{1}{V^2} - \dfrac{1}{c^2}}} \cdot \frac{\boldsymbol{\upsilon}_A}{\boldsymbol{\upsilon}_c}. \tag{9}$$

а



б

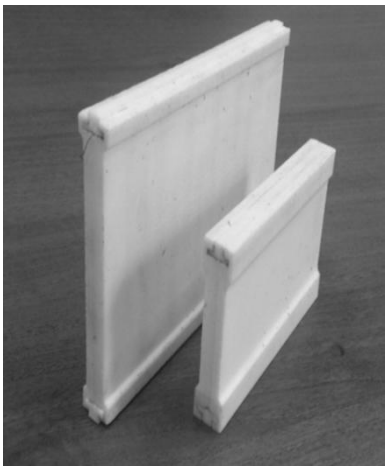


Рис. 2. Конструкция и внешний вид развернутых конденсаторов (все размеры – в мм):

а – конструкция герметичного развернутого конденсатора: серым цветом выделены медные электроды, белым – элементы конструкции из фторопласта;

б – внешний вид развернутых конденсаторов.

Приведенный выше анализ позволяет прогнозировать некоторые свойства, присущие ΔF:

1. Так как интенсивность потери массы телом «A» в направлении φ в вакууме ограничена величиной $\left.\frac{dm_A}{dt}\right|_{\varphi} = \frac{H \cdot \boldsymbol{\upsilon}_A}{4\pi}(\rho_A - \rho_\varphi)$, величина $\Delta F\varphi$ тоже должна обладать верхней границей, не зависящей от напряжения, поданного на конденсатор.

2. Для двух тел «A_1» и «A_2», обладающих разной массой и одинаковой плотностью, справедлива импликация:

$$m(A_1) < m(A_2) \Rightarrow \Delta F(m(A_1)) < \Delta F(m(A_2)). \qquad (10)$$

Ниже приводятся описания постановки и результатов экспериментов, предназначенных, с одной стороны, для проверки приведенных выше прогнозов, а с другой – для оценки влияния на величину $\Delta F\varphi$ движения заряженных частиц, возникающих при подаче на конденсатор высокого напряжения.

3. Постановка и результаты экспериментов

Для предотвращения искровых разрядов в рабочей зоне конденсатора при напряжениях до 20 кВ, последняя конструировалась герметичной (рис. 2*а*).

В ходе экспериментов груз (тело «A») размещался горизонтально на чашке прецизионных весов, установленных под вакуумным колпаком, вплотную под развернутым конденсатором (рис. 1), вес которого равнялся 323 г., а емкость – 10,1 пФ, объем рабочей области конденсатора V_c составлял $1656 \cdot 10^{-8}$ м^3. Подаваемое на конденсатор напряжение менялось в диапазоне 0–20 кВ с шагом 5 кВ. Все грузы представляли собой стеклянные прямоугольники, вес которых был в диапазоне 52–153 г, а площадь поверхности не превышала площади поверхности рабочей области конденсатора. Давление в зоне эксперимента менялось в диапазоне 0,007–705 мм рт. ст. Для каждой комбинации «груз – фиксированные значения давления и напряжения» осуществлялось десять замеров величины подъемной силы ΔF с интервалом 3–5 секунд и их среднеарифметическое значение заносилось в соответствующую таблицу (см. Приложение, табл. 2–5). Верхние границы величины ΔF в вакууме определялись на основании эмпирических зависимостей вида:

$$\Delta F = A(1 - \exp(-BU), \qquad (11)$$

где A и B – коэффициенты, величины которых приведены ниже в табл. 1

Таблица 1

№ опыта	Вес полезного груза (г.)	A	B	Среднеквадратичное отклонение ΔF, определенное на основании (11) от экспериментального
1	2	3	4	5
1	0,052	31,966	$6{,}36 \cdot 10^{-5}$	5,41
2	0,064	28,274	$9{,}0 \cdot 10^{-5}$	0,122
3	0,084	29,122	$9{,}99 \cdot 10^{-5}$	0,9146
4	0,153	127,851	$15 \cdot 10^{-6}$	6,5035

Легко убедиться, что:

$$\Delta F_{\max} = \lim_{U \to \infty} A[1 - \exp(-BU)] = A, \qquad (12)$$

откуда следует, что величина $\Delta F_{\max}$ применительно к каждому образцу приведена в третьем столбце таблицы 1.

Аппроксимируя, на основании табл. 1, зависимость верхней границы подъемной силы в вакууме ΔF_v от массы полезного груза m полиномом второго порядка, получим:

$$\Delta F_v = 102{,}5831 + 2131{,}872m + 15612{,}31m^2. \qquad (13)$$

Следует также отметить, что скорость V, определяемая на основании (9), в ходе экспериментов оказалась близкой к скорости света.

4. Заключение

Выводы, полученные по результатам экспериментов:

1. Полученные экспериментально результаты подтверждают справедливость основных положений и выкладок, изложенных во втором параграфе.

2. Изменения давления воздуха в зоне экспериментов не влияли существенно на характер зависимостей подъемной силы от напряже-

ния применительно к каждому образцу. Таким образом, влияние эффекта Бифельда – Брауна [14] на результаты экспериментов можно не учитывать.

3. Импликация (10) справедлива, только если $m(A_1) << m(A_2)$.

4. Полученные данные дают шанс на создание принципиально нового двигателя для космических транспортных систем на основе развернутых конденсаторов: корпус, покрытый такого рода конденсаторами, может служить одновременно двигательной установкой и рулями.

5. Литература

1. *Hubble E. P.* A relation between distance and radial velocity among extra-galactic nebulae // Proceedings of the National Academy of Sciences of the United States of America. Vol. 15. No. 3. Pp. 167–173. 1929.

2. *Newton I.* The mathematical principles of natural knowledge. 1667.

3. *Langley Samuel P.* The Le Sage theory of gravitation // Annual Report of the Board of Regents of the Smithsonian Institution. June 30. 1898. Pp. 139–160.

4. *Einstein A.* The theory of relativity // Die Physik / Under reduction of E. Lechner. 1915. V. 3. Leipzig. Pp. 703–713.

5. *Einstein A.* About the cosmological Problem of general Theory of Relativity // Sitzungsher. preuss. Akad. Wiss., phys.-math. K1. 1931. Pp. 235–237, (German).

6. *Groppen V. O.* Kinematics in space with variable linear measurement standards // Proceedings of the International Conference: Mathematics and Astronomy: A joint long Journey. Madrid, Spain, 23–27 November 2009. Pp. 149–155.

7. *Groppen V. O.* Manifestations of Measurement Standards Variability in the Universe Modeling. Saarbrucken (Germany): Lambert Academic Publishing, 2013. 76 p.

8. *Groppen V. O.* Gravity as a Sum of Reaction Forces. Recent Advances in Robotics, Aeronautical & Mechanical Engineering // Proceedings of the 1-st International Conference on Mechanical and Robotics Engineering. Proceedings of the 1-st International Conference on Aeronautical and Mechanical and Engineering. Vouliagmeni: Athens, Greece. May 14–16, 2013. Pp. 155–160.

9. *Groppen V. O.* Gravity control: modeling and experiments // Proceedings of the 2014 International Conference on Energy, Environment,

Ecosystems and Development II (EEED'14). Prague, Czech Republic. April 2 – 4, 2014, pp. 15 – 17.

10. *Groppen V. O.* The Hubble Law and Gravity as Manifestations of Linear Measurement Standards Variability // Journal of Modeling, Simulation, Identification and Control. Columbia International Publishing. Vol. 3. No. 1. 2015. Pp. 1–12.

11. *Groppen V. O.* The Gravity Control Experiments: Sensors, Equipment, Results // Proceedings of the International Conference on Applied Physics, Simulation and Computers (APSAC 2015).Vienna, Austria. March 15–17. 2015. Pp. 210–214.

12. *Groppen V. O.* Gravity forces as a tool for the experimental verification of the universe simulators based on the measurement standards variability // International Journal of Mathematical Models and Methods in Applied Sciences. ISSN: 1998-0140. Vol. 9. 2015. Pp 345–351.

13. *Miele A.* Flight Mechanics. Volume 1: Theory of Flight Paths. Addison-Wesley Pub., 1962. 416 p.

14. *Tajmar M.* Biefeld – Brown Effect: Misinterpretation of Corona Wind Phenomena // American Institute of Aeronautics and Astronautics Journal. 42: 315, DOI:10.2514/1.9095 (2004).

П р и л о ж е н и е

1. Зависимость подъемной силы ΔF (в миллиграммах) от напряжения и давления воздуха в зоне эксперимента для полезного груза (тела «*A*») весом 153 г.

Таблица 2

Давление (мм рт. ст.)	Напряжение (кВ)				
	0	5	10	15	20
705	0	10,5	19,5	29	39,5
7,05	0	10	20,5	28,5	38,5
0,70	0	9	18,5	28,5	38
0,07	0	8,5	18	26	37
0,007	0	7,5	18	24,5	34,5

2. Зависимость подъемной силы ΔF (в миллиграммах) от напряжения и давления воздуха в зоне эксперимента для полезного груза весом 84 г.

Таблица 3

Давление (мм рт. ст.)	Напряжение (кВ)				
	0	5	10	15	20
705	0	14,5	22	28	31
7,05	0	13,5	20,5	26	30,5
0,70	0	12	20	27,5	29
0,07	0	11,5	19	24,5	26
0,007	0	12	18,5	23	24,5

3. Зависимость подъемной силы ΔF (в миллиграммах) от напряжения и давления воздуха в зоне эксперимента для полезного груза весом 64 г.

Таблица 4

Давление (мм рт. ст.)	Напряжение (кВ)				
	0	5	10	15	20
705	0	13	21	26	29
7,05	0	11,5	21	24,5	27,5
0,70	0	11	18	24,5	28
0,07	0	10	18	21,5	26,5
0,007	0	10	17	21	23,5

4. Зависимость подъемной силы ΔF (в миллиграммах) от напряжения и давления воздуха в зоне эксперимента для полезного груза весом 52 г.

Таблица 5

Давление (мм рт. ст.)	Напряжение (кВ)				
	0	5	10	15	20
705	0	12	20	25	27
7,05	0	10,5	19	24	27
0,70	0	10	20	24,5	26,5
0,07	0	10	16,5	21,5	26
0,007	0	7	16,5	20	22,5

Гроппен В. О.,
доктор техн. наук, профессор,
заведующий кафедрой
автоматизированной обработки информации
СКГМИ (ГТУ)
e-mail: groppen@mail.ru

Проскурин А. Е.,
канд. техн. наук, доцент кафедры
автоматизированной обработки информации
СКГМИ (ГТУ)
e-mail: alexander@skgmi-gtu.ru

CONTROL OF FORCES OF THE GRAVITATIONAL INTERACTION IN A VACUUM: MODELING AND EXPERIMENTAL RESULTS

Professor **V. O. Groppen**, Associate Professor **A. E. Proskurin**

The article presents the results of the first experiments to control the forces of gravitational interaction in a vacuum by means of deployed capacitors, which are the based on an alternative interpretation of the Hubble Law, thus resulting in the model being alternative of the Standard Model. The experimental results show independence of the proposed approach from the effect of Biefield-Brown. The results give a chance to the creation of new motors and control systems that do not have moving parts.

Keywords: *gravity, control, deployed capacitor, high voltage, vacuum.*

УДК 004.272

Мирошников А. С.,
Кожиев В. С.

ЭФФЕКТИВНОСТЬ ИСПОЛЬЗОВАНИЯ ТЕХНОЛОГИИ CUDA ДЛЯ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ШИФРОВАНИЯ ГОСТ 28147-89

В статье описан алгоритм шифрования ГОСТ 28147-89 и его параллельная реализация с использованием технологии CUDA. Технология CUDA (Compute Unified Device Architecture) от компании NVidia позволяет достаточно легко писать приложения, выполняющиеся параллельно на десятках тысяч нитей, что позволяет получить большой прирост производительности.

Ключевые слова: *шифрование, дешифрование, ГОСТ 28147-89, нить, варп, блок, NVidia, CUDA.*

Введение

Алгоритм шифрования ГОСТ 28147-89 разрешено использовать в государственных учреждениях. Ввиду того, что в государственных структурах увеличивается объем хранимой информации в цифровом виде различного уровня важности, которую в соответствии с российским законодательством следует защищать сертифицированными криптографическими средствами. Возникает вопрос, как наиболее эффективно осуществлять защиту информации, передаваемой по сетям общего пользования и хранящейся на информационных носителях, без существенной потери производительности системы и избыточных денежных затрат. Для этих целей прекрасно подходят графические ускорители NVidia, на которых можно распараллелить данный алгоритм шифрования, тем самым увеличив скорость шифрования, при относительно невысоких денежных затратах.

Цель данной статьи:

- разработка и реализация параллельного алгоритма шифрования ГОСТ 28147-89 с использованием технологии CUDA.
- проверить, есть ли выигрыш по скорости шифрования данных между алгоритмом шифрования ГОСТ 28147-89, реализованным на графическом процессоре (GPU), в сравнении с алгоритмом шифрова-

ния ГОСТ 28147-89, реализованным на центральном процессоре (CPU).

Обозначения и определения:

CUDA (Compute Unified Device Architecture) – это технология от компании NVidia, предназначенная для разработки приложений для массивно-параллельных вычислительных устройств (в первую очередь для GPU, начиная с серии G80).

Поток (thread) – набор данных, который необходимо обработать (не требует больших ресурсов при обработке).

Варп (warp) – группа из 32 потоков. Данные обрабатываются только варпами, следовательно варп – это минимальный объем данных.

Блок (block) – совокупность потоков (от 64 до 1024) или совокупность варпов (от 2 до 32 бит).

Ключ является массивом из восьми 32-битных элементов кода, далее в статье он обозначается символом $\boldsymbol{K} : \boldsymbol{K} = \{\boldsymbol{K_i}\}_{0 \le i \le 7}$. В ГОСТе элементы ключа используются как 32-разрядные целые числа без знака: $0 \le \boldsymbol{K_i} < 2^{32}$.

Таблица замен является матрицей 8 × 16, содержащей 4-битовые элементы, которые можно представить в виде целых чисел от 0 до 15. Строки ***таблицы замен*** называются ***узлами замен***, они должны содержать различные значения, то есть каждый ***узел замен*** должен содержать 16 различных чисел от 0 до 15 в произвольном порядке. В статье таблица замен обозначается символом $\boldsymbol{H}$:

$$\boldsymbol{H} = \{\boldsymbol{H_{i,j}}\}_{0 \le \boldsymbol{j} \le 15}^{0 \le \boldsymbol{i} \le 7}, 0 \le \boldsymbol{H_{i,j}} \le 15.$$

Описание алгоритма шифрования ГОСТ 28147-89

ГОСТ 28147-89 [1] – симметричный алгоритм блочного шифрования; размер блока 64 бит, размер ключа 256 бит, количество раундов сети Фейстеля – 32 [2]. Также в процессе обработки используется дополнительный ключ, называемый S-блоком и представляющий собой перестановку чисел от 0 до 15. Способ генерации S-блоков в стандарте не оговаривается. Обычно они генерируются разработчиками с помощью генератора случайных чисел. S-блоки являются дополнительным ключевым материалом и должны храниться в секрете [3]. Существует четыре режима работы данного алгоритма: режим простой замены, режим гаммирования, гаммирование с обратной связью и режим выработки имитовставки. Режим простой замены предполагает разбиение сообщения на 64-битные блоки с последующей незави-

симой обработкой каждого блока. Таким образом, данный режим алгоритма шифрования ГОСТ 28147-89 позволяет выполнить распараллеливание обработки по данным (рис. 5). На рис. 1 представлена схема раунда сети Фейстеля для алгоритма ГОСТ 28147-89 [4].

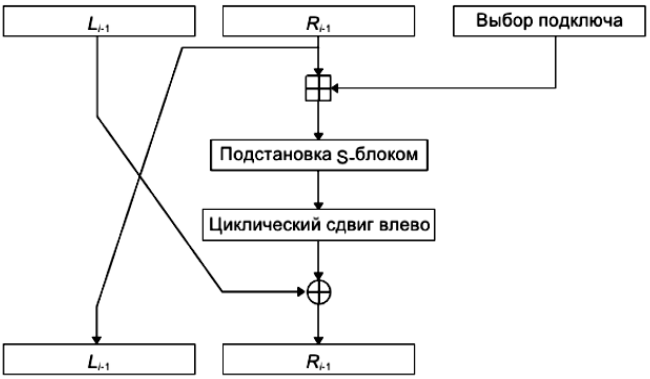


Рис. 1. Схема шифрования по алгоритму ГОСТ 28147-89

Алгоритм шифрования данных в режиме простой замены представлен на рис. 2:

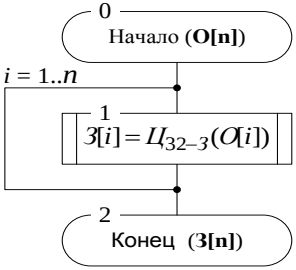


Рис. 2. Алгоритм шифрования данных в режиме простой замены ГОСТ 28147-89

$O[n]$ – массив открытых данных;

$З[n]$ – массив зашифрованных данных;

$O[i]$ – i-е по порядку 64-битовые блоки открытых данных: $O[n] = (O[1], O[2], ..., O[n])$;

$З[i]$ – i-е по порядку 64-битовые блоки зашифрованных данных: $З[n] = (З[1], З[2], \ldots, З[n])$;

n – число 64-битовых блоков в массиве данных $1 <= i <= n$;

$Ц_{32\text{-}З}$ – функция преобразования 64-битного блока данных по алгоритму базового цикла шифрования (рис. 3*а*).

$Ц_{32\text{-}Р}$ – функция преобразования 64-битного блока данных по алгоритму базового цикла дешифрования (рис. 3*б*).

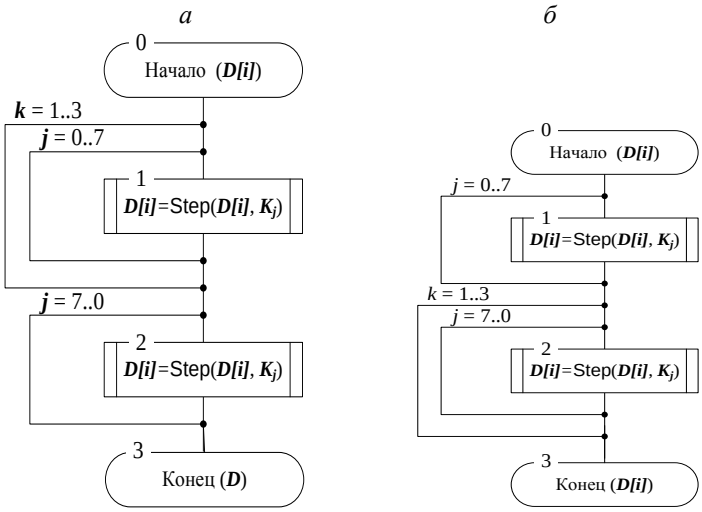


Рис. 3. Схемы циклов: *а* – шифрования; *б* – дешифрования.

Основной шаг криптопреобразования (Step (D, K_j)) (рис. 4)

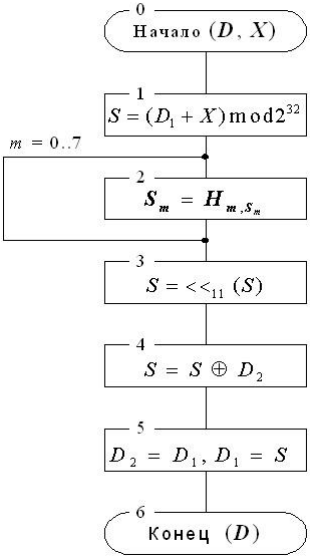


Рис. 4. Алгоритм основного шага криптопреобразования алгоритма ГОСТ 28147-89.

Шаг 0. Определяет исходные данные для основного шага криптопреобразования;

Шаг 1. D – преобразуемый 64-битовый блок данных, в ходе выполнения шага его младшая (D_1) и старшая (D_2) части обрабатываются как отдельные 32-битовые целые числа без знака. Таким образом, можно записать $D = (D_1, D_2)$.

Шаг 2. X – 32-битовый элемент ключа;

Шаг 3. Сложение с ключом. Младшая половина преобразуемого блока складывается по модулю 2^{32} с используемым на шаге элементом ключа, результат передается на следующий шаг;

Шаг 4. Поблочная замена. 32-битовое значение, полученное на предыдущем шаге, интерпретирует-

ся как массив из восьми 4-битовых блоков кода: $S = (S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7)$.

Шаг 5. Далее значение каждого из восьми блоков заменяется на новое, которое выбирается по таблице замен следующим образом: значение блока S_i заменяется на S_i-й по порядку элемент (нумерация с нуля) i-го узла замен (т. е. i-й строки таблицы замен, нумерация также с нуля).

Шаг 6. Циклический сдвиг на 11 бит влево. Результат предыдущего шага сдвигается циклически на 11 бит в сторону старших разрядов и передается на следующий шаг.

Шаг 7. Побитовое сложение: значение, полученное на шаге 3, побитно складывается по модулю 2 со старшей половиной преобразуемого блока.

Шаг 8. Сдвиг по цепочке: младшая часть преобразуемого блока сдвигается на место старшей, а на ее место помещается результат выполнения предыдущего шага.

Шаг 9. Полученное значение преобразуемого блока возвращается как результат выполнения алгоритма основного шага криптопреобразования.

На рис. 5 представлен параллельный алгоритм шифрования ГОСТ 28147-89.

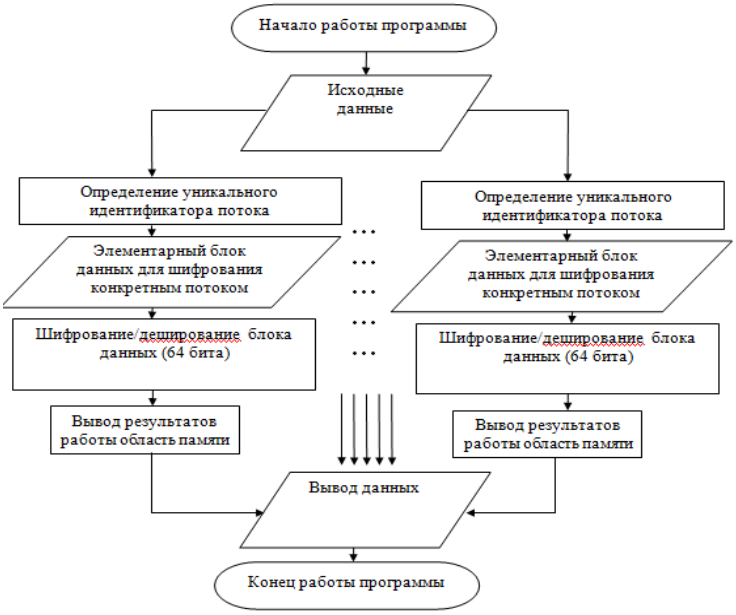


Рис 5. Параллельный алгоритм шифрования ГОСТ 28147-89

Математическая модель

Для эффективного использования графического процессора необходимо оптимизировать количество нитей, используемых при параллельном алгоритме шифрования данных из *N* байт. С возрастанием количества нитей скорость шифрования данных увеличится, но также и увеличится время на организацию работы нитей. Принимая во внимание эти факторы, была выбрана следующая математическая модель [5, 6]:

$$\begin{cases} \sum_{i=1}^{q}(c + f x_i + \frac{d_i}{x_i}) \to \min \\ \forall i : 1 \le x_i \le \min\{P; B_i N\}, \text{целое}; i = \overline{1,q}, \end{cases} \quad (1)$$

в которой используются следующие условные обозначения:

c – среднее время на организацию вычислений;

f – среднее время на организацию одной нити;

x_i – количество потоков, задействованных в параллельном алгоритме шифрования данных N байт в течение одной итерации;

d_i – время шифрования i-го элемента массива данных одним потоком;

P – максимальное количество активных потоков GPU;

q – количество моделируемых систем;

B_i – число потоковых блоков.

Решив поставленную задачу, определяем оптимальное количество потоков, необходимое для того, чтобы время работы программного комплекса было минимальным.

Для использования GPU найденное количество потоков необходимо разбить на блоки таким образом, чтобы нагрузка на видеоускоритель была максимальной. Для подсчета этих параметров была выбрана методика, предложенная компанией NVidia. В методике используются следующие параметры:

N – число нитей в блоке;

N_W – число нитей в варпе;

$N_{B\max}$ – максимальное число нитей в блоке;

$W_{\max}$ – максимальное число варпов на мультипроцессоре;

$B_{\max}$ – максимальное число блоков на мультипроцессоре;

$N_{\max}$ – максимальное число нитей на мультипроцессоре;

W – число варпов;

W_a – число активных варпов на мультипроцессоре;

B_a – число активных блоков на мультипроцессоре;

N_a – число активных нитей на мультипроцессоре;

Число активных варпов на мультипроцессоре – это произведение числа активных блоков на число варпов, и получившееся произведение должно быть меньше максимального числа варпов на мультипроцессоре. Максимальное число нитей на мультипроцессоре – это произведение максимального числа варпов на число нитей в варпе. Число варпов рассчитывается как частное числа нитей в блоке на число нитей в варпе. Число нитей в блоке – целое число, которое не может быть меньше единицы и больше максимального числа нитей в блоке.

Для эффективного использования GPU число активных нитей и активных блоков на мультипроцессоре должно быть максимальным. Принимая во внимание вышесказанное, была предложена следующая математическая модель:

$$\begin{cases} B_a = \min(B_{\max}, \left\lfloor \frac{W_{\max}}{W} \right\rfloor) \to \max \\ N_a = \min(N_{\max}, B_a N) \to \max \\ W_a = B_a W \le W_{\max} \\ N_{\max} = W_{\max} N_W \\ W = \left\lceil \frac{N}{N_W} \right\rceil \\ 1 \le N \le N_{B\max}, \text{öåëîå}, \end{cases} \qquad (2)$$

где $\lfloor \ \rfloor$ – округление в меньшую сторону до ближайшего целого,

$\lceil \ \rceil$ – округление в большую сторону до ближайшего целого.

Пример расчета

Исходные данные:

- Размер файла: N = 4800 байт.
- Время шифрования одной нитью $d = 377$ мс.
- Среднее время на организацию одной нити $f = 0{,}008$ мс.
- Среднее время на организацию вычислений $c = 100$ мс.

Исходные параметры видеокарты NVidia GeForce GT 630:

- Версия – 2.1
- N_w – 32
- $W_{\max}$ – 48
- $B_{\max}$ – 6
- SM – 2

Требуется найти оптимальное количество нитей x, при котором время работы программы минимально.

Решение:

$$N_{\max} = W_{\max} \cdot N_W = 48 \cdot 32 = 1536$$

$$N = \frac{N_{\max}}{B_{\max}} = \frac{1536}{6} = 256;$$

$$W = \left\lceil \frac{N}{N_w} \right\rceil = \left\lceil \frac{256}{32} \right\rceil = 8.$$

Следовательно, максимальное количество нитей на GPU составит:

$$p = 1536 \cdot 2 = 3072.$$

Целевая функция (1) примет вид [5, 6]:

$$T = c + f \cdot x + \frac{d}{x},$$

$$T'(x) = 0 + f + d \cdot \left(-\frac{1}{x^2}\right) = 0.$$

Следовательно, $x' = \sqrt{\frac{d}{f}}$.

При $\frac{d^2T}{dx^2} > 0$, справедливо:

$$T''(x) = 6d \cdot \left(-\frac{1}{x^4}\right) = 0.$$

Следовательно, $x'' = \sqrt[4]{6d}$,

$$x'' = \sqrt[4]{6 \cdot 377} = 6{,}896.$$

$$x = \begin{cases} 1, & \text{если } x' < 1 \\ p, & \text{если } x' > p \\ \lfloor x' \rfloor, & \text{если } T(\lfloor x' \rfloor) < T(\lceil x' \rceil) \\ \lceil x' \rceil, & \text{если } T(\lfloor x' \rfloor) \geq T(\lceil x' \rceil) \end{cases} \quad (3)$$

$$x' = \sqrt{\frac{377}{0{,}008}} = 217{,}1$$

$$T(\lfloor x' \rfloor) = 100 + 0{,}008 \cdot 217 + \frac{377}{217} = 103{,}4733 \text{ мс}$$

$$T(\lceil x' \rceil) = 100 + 0{,}008 \cdot 218 + \frac{377}{218} = 103{,}4734 \text{ мс}$$

Исходя из системы (3), получим:

$$x = \lfloor x' \rfloor = 217.$$

Время работы: $T(x) = 103{,}4733$ мс.

Количество блоков:

$$B = \left\lceil \frac{x}{N} \right\rceil = \left\lceil \frac{217}{256} \right\rceil = 1.$$

Таким образом, для данной видеокарты получены следующие оптимальные параметры:

1) количество нитей в блоке $N = 256$;

2) число блоков $B = 1$.

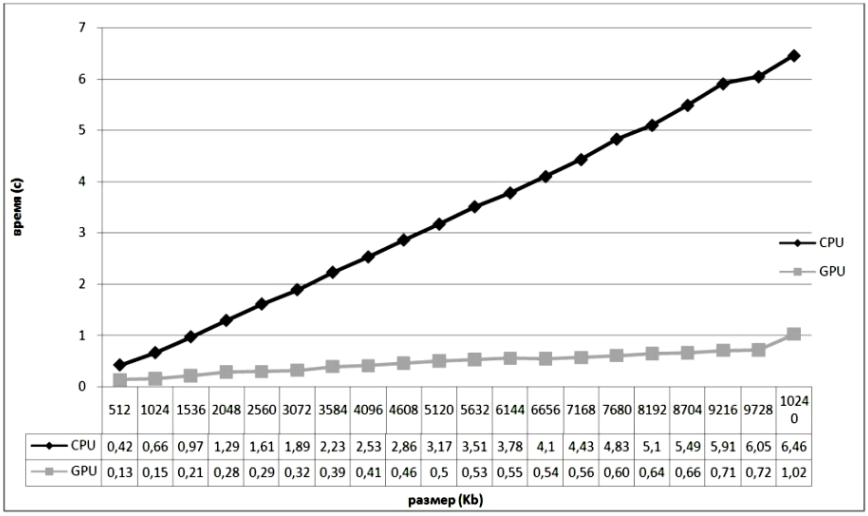
Следовательно, для шифрования данных из N байт на текущем графическом процессоре следует использовать полученные параметры.

На рис. 7 представлена экспериментальная зависимость времени шифрования от размера файла на центральном процессоре (CPU) и на графическом процессоре (GPU).

Исследование производительности

```
Run gost algorithm
File name: 10.rar
Key name: key.txt
Mode: 1
Key: 7b0b3ae4  3cd59810  64127c8a  1ad392c3  e8b0e0b1  53aec453  a5169113  78073
048  0  7       10       9        4        11       14       13       8        15
2       1       12       3        6        5
1       12      3        6        5        0        7        10       9        4
11      14      13       8        15       2
2       1       12       3        6        5        0        7        10       9
4       11      14       13       8        15
3       6       5        0        7        10       9        4        11       14
13      8       15       2        1        12
4       11      14       13       8        15       2        1        12       3
6       5       0        7        10       9
5       0       7        10       9        4        11       14       13       8
15      2       1        12       3        6
6       5       0        7        10       9        4        11       14       13
8       15      2        1        12       3
7       10      9        4        11       14       13       8        15       2
1       12      3        6        5        0
$HAG: 1
GPU compute time: 353
Finish
```

Рис. 6. Пример интерфейса, отображающего работу программы



	512	1024	1536	2048	2560	3072	3584	4096	4608	5120	5632	6144	6656	7168	7680	8192	8704	9216	9728	10240
CPU	0,42	0,66	0,97	1,29	1,61	1,89	2,23	2,53	2,86	3,17	3,51	3,78	4,1	4,43	4,83	5,1	5,49	5,91	6,05	6,46
GPU	0,13	0,15	0,21	0,28	0,29	0,32	0,39	0,41	0,46	0,5	0,53	0,55	0,54	0,56	0,60	0,64	0,66	0,71	0,72	1,02

Рис. 7. График зависимости времени шифрования от размера файла на CPU и на GPU

Заключение

В ходе данной работы был разработан и реализован параллельный алгоритм шифрования ГОСТ 28147-89 с использованием технологии CUDA.

Проведено исследования производительности реализации алгоритма шифрования на центральном процессоре и на графическом процессоре. Исследование показало значительный прирост производительности, при использование ресурсов видеокарты. Скорость при использовании технологии CUDA увеличилась более чем в 3 раза. Данные результаты обусловлены эффективностью использования вычислительных ресурсов графических процессоров для шифрования данных, позволяющих осуществлять параллельную, многопоточную обработку данных.

Литература

1. ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм.
2. *Баричев С. Г., Серов Р. Е.* Основы современной криптографии. М.: Изд-во Горячая линия – Телеком, 2002. 122 с.
3. *Шнайер Б.* Прикладная криптография. Изд-во Триумф, 2002. 816 с.
4. *Миниахметова М. С., Цымблер М. Л.* Разработка параллельного алгоритма шифрования ГОСТ 28147-89 на платформе Intel Xenon Phi // Параллельные вычислительные технологии (ПаВТ'2013): Труды Международной научной конференции (1–5 апреля 2013 г., г. Челябинск). Челябинск: Издательский центр ЮУрГУ, 2013. 606 с.
5. *Мирошников А. С.* Система управления параллельной обработки данных в локальных вычислительных сетях. Дисс…. канд. техн. наук. Владикавказ, 2000.
6. *Groppen V. O.* Smart computing. 2004. 103 p.

Мирошников А. С.,
канд. техн. наук, доцент кафедры
автоматизированной обработки информации
СКГМИ (ГТУ)
e-mail: mirandrey@mail.ru

Кожиев В. С.,
студент СКГМИ (ГТУ)
e-mail: kozhiev-vladimir@yandex.ru

THE EFFECTIVENESS OF THE USE OF CUDA TECHNOLOGY FOR PARALLEL ENCRYPTION ALGORITHM GOST 28147-89

Associate Professor **A. S. Miroshnikov,** student **V. S. Kozhiev**

This article describes the encryption algorithm GOST 28147-89, and its parallel implementation using technology CUDA. Technology CUDA (Compute Unified Device Architecture) from NVidia's allows easy enough to write applications that run parallel to the tens of thousands of threads, which allows you to get a big performance boost.

Keywords: *encryption, decryption, GOST 28147-89, thread, warp, block, NVidia, CUDA.*

Содержание

НОВЫЕ ТЕХНОЛОГИИ ПРИНЯТИЯ РЕШЕНИЙ

ТЕХНОЛОГИИ ЭКСТРЕМАЛЬНОГО ПРОГРАММИРОВАНИЯ

ОБРАБОТКА ИЗОБРАЖЕНИЙ

МОДЕЛИРОВАНИЕ СИСТЕМ