IT-ТЕХНОЛОГИИ:
ТЕОРИЯ И ПРАКТИКА

Материалы семинара

Владикавказ 2017

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:

ТЕОРИЯ И ПРАКТИКА

Материалы семинара

Владикавказ 2017

УДК 004
ББК 73
И74

И74 **IT-технологии: теория и практика:** Материалы семинара / Коллектив авторов; Северо-Кавказский горно-металлургический институт (государственный технологический университет). – Владикавказ: Северо-Кавказский горно-металлургический институт (государственный технологический университет). Изд-во «Терек», 2017. – 106 с.

ISBN 978-5-9500069-3-7

В сборник вошли лучшие доклады, представленные на ежегодном декабрьском семинаре «ИТ-технологии: теория и практика» в 2016 году. В них подведены итоги и отражены результаты по ряду основных направлений исследований, осуществлявшихся преподавателями и студентами кафедры автоматизированной обработки информации СКГМИ (ГТУ) в 2016 году. В рамках этих направлений были предложены новые технологии принятия решений, получили развитие современные методы экстремального программирования, параллельной обработки данных и обработки изображений.

УДК 004
ББК 73

Редактор: *Иванченко Н. К.*

Компьютерная верстка: *Цишук Т. С.*

ISBN 978-5-9500069-3-7

НОВЫЕ ТЕХНОЛОГИИ ПРИНЯТИЯ РЕШЕНИЙ

УДК 519.834

Гроппен В. О., Будаева А. А.

ЭТАЛОНЫ КАК УНИКАЛЬНЫЙ ИНСТРУМЕНТ ПОСТАНОВКИ И РЕШЕНИЯ ЗАДАЧ ТЕОРИИ ПРИНЯТИЯ РЕШЕНИЙ

Предлагается ряд новых технологий, предназначенных для решения задач различных разделов теории принятия решений и базирующихся на применении эталонов. К вышеназванным разделам относятся задачи, сводимые к игровым моделям, к принятию решений голосованием, к использованию экспертных оценок, к многокритериальной оптимизации, к оценке качества многокритериальной таксономии и к решению многокритериальных задач дискретной оптимизации методами типа ветвей и границ. Основные результаты иллюстрируются примерами.

Ключевые слова: *матричная игра, эталон, голосование, метод ветвей и границ, бинарные сравнения, экспертные оценки, многокритериальная оптимизация, таксономия.*

1. Введение

Широко известно применение эталонов как одного из методов решения многокритериальных задач [1; 2], а также их использование в рамках методов типа ветвей и границ – в этом случае термин «эталон» заменяется словом «оценка». Ниже предлагается ряд новых технологий решения задач, охватывающих практически все области теории принятия решений, причем ядром, объединяющим предлагаемые подходы, является использование эталонов. Эффективность предлагаемых технологий доказывается рядом теорем и иллюстрируется примерами. При этом используются следующие ниже обозначения и определения.

2. Обозначения и определения

M – матрица игры двух лиц, строки которой соответствуют стратегиям максимизирующего игрока, а столбцы – минимизирующего;

y_j – j-я стратегия минимизирующего игрока;

x_i – i-я стратегия максимизирующего игрока;

h – максимальный выигрыш максимизирующего игрока:

$$h = \max_i \max_j M(i, j);$$

g – минимальный проигрыш минимизирующего игрока:

$$g = \min_i \min_j M(i, j);$$

Величины "h" и "g" являются эталонами, позволяющими игрокам оценить отклонение текущего выигрыша/проигрыша от наилучшего значения.

Остальные обозначения вводятся ниже по ходу применения в соответствии со спецификой рассматриваемых задач.

3. Эталоны в поиске оптимальных чистых стратегий в кооперативных играх

Решение в чистых стратегиях антагонистических матричных игр двух лиц с полной информацией и нулевой суммой базируется на двух принципах, одним из которых является применение гарантирующих стратегий, а другим, связанным с предыдущим принципом, можно считать запрет игрокам на любые договоренности [3; 4]. Основой предлагаемого ниже подхода являются:

а) отказ от обоих вышеприведенных принципов [5; 6];

б) поиск оптимальных стратегий с использованием эталонов [5; 6].

Иными словами в рамках предлагаемого подхода полагаем, что до начала игры игроки вырабатывают процедуру определения цены игры, после чего каждый выбирает соответствующую стратегию. Так как интересы игроков противоположны, формально задача поиска оптимальной стратегии может быть многокритериальной:

$$\begin{cases} \sum_i \sum_j M(i, j) x_i y_j \to \max; \\ \sum_i \sum_j M(i, j) x_i y_j \to \min; \\ \sum_i x_i = 1; \\ \sum_j y_j = 1; \\ \forall i : x_i = 1{,}0; \\ \forall j : y_j = 1{,}0. \end{cases} \quad (1)$$

Использование эталонов в соответствии с принятыми выше обозначениями позволяет преобразовать (1) к однокритериальному виду:

$$\begin{cases} \left(\sum_i \sum_j M(i,j) x_i y_j - h \right)^2 + \left(\sum_i \sum_j M(i,j) x_i y_j - g \right)^2 \to \min; \\ \sum_i x_i = 1; \\ \sum_j y_j = 1; \\ \forall i : x_i = 1{,}0; \\ \forall j : y_j = 1{,}0. \end{cases} \qquad (2)$$

Решение системы (2) базируется на следующей теореме:

Теорема 1. Оптимальная цена игры, отвечающая системе (2), определяется ячейкой $M(p, q)$ матрицы игры M, для которой справедливо:

$$\left[M(p,q) - \frac{h+g}{2} \right]^2 = \min_i \min_j \left[M(i,j) - \frac{h+g}{2} \right]^2. \qquad (3)$$

Доказательство теоремы приводится в Приложении 1.

Так, например, цена S_1 антагонистической игры Γ двух лиц в чистых стратегиях, которой соответствует матрица M (см. рис. 1 ниже), при применении гарантирующих стратегий равна девяти, оптимальной стратегией максимизирующего игрока является третья, а минимизирующего – вторая. Однако, если игроки способны договариваться, цена игры S_2 и оптимальные стратегии игроков изменятся: $S_2 = 8$, оптимальной стратегией максимизирующего игрока в этом случае является первая стратегия, а минимизирующего – третья.

$M =$

2	10	8
12	6	3
4	9	14

Рис. 1. Матрица антагонистической игры двух лиц

Следует отметить, что предложенный выше подход не позволяет выбрать стратегии игроков, если матрице игры M принадлежит не-

сколько ячеек, удовлетворяющих (3). Формально этот случай для двух ячеек такого рода $M(p, q)$ и $M(r, f)$ может быть описан системой:

$$\begin{cases} \exists p \neq r, \exists q \neq f : [M(p,q) - G]^2 = [M(r,f) - G]^2; \\ G = 0{,}5(h+g). \end{cases} \quad (4)$$

Примером (4) может служить матрица M_1 (рис. 2):

$M1 =$

1	10	8
12	6	3
4	9	13

Рис. 2. Матрица M_1, удовлетворяющая системе (4)

Так как в этом случае $h = 13$, а $g = 1$, оптимальная величина G равна 7, и две ячейки $M1(1, 3)$ и $M1(2, 2)$ удовлетворяют (3) и (4). В таком случае очевидна необходимость дополнительных условий, которые бы позволили однозначный выбор оптимальных стратегий игроков. Примером может служить пара стратегий, удовлетворяющих (3) и (4), и одновременно определяющих цену игры, ближайшую к той, которую фиксируют гарантирующие стратегии.

4. Технология подведения итогов голосования, базирующаяся на эталонах

Используемые сегодня для обработки результатов голосования различные технологии, такие, как методы абсолютного и относительного большинства [3], метод Борда [8] и ряд других, не гарантируют получения однозначного результата. Так, применительно к результатам голосования, приведенным ниже в таблице 1, подведение итогов с помощью метода относительного большинства приводит к победе кандидата «а», в то время как проведение двух туров голосования с использованием метода абсолютного большинства отдаст победу претенденту «б». В модифицированном методе Борда [2, 10] каждый выборщик, присваивает кандидату, попавшему на j-е место, j баллов. Выигрывает «с», набравший минимальную сумму баллов:

$$n_a = 1 \cdot 6 + 3 \cdot 7 = 27;$$
$$n_b = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 4 = 26;$$
$$n_c = 1 \cdot 3 + 2 \cdot 8 + 3 \cdot 2 = 25 \text{ и т. д.}$$

Таблица 1

Место	Голоса избирателей			
	2	3	4	4
1	a	c	a	b
2	b	b	c	c
3	c	a	b	a

Вышеприведенное различие результатов, получаемых различными технологиями обработки результатов голосования, доказывает актуальность дальнейшего развития технологий такого рода. Ниже приводится новая технология обработки результатов голосования, базирующаяся на применении эталонов. Простота этой технологии объясняется простотой определения эталона – ему соответствует случай, когда все выборщики голосуют за одного претендента. Пусть существует "*n*" мест, на которые претендуют "*m*" кандидатов, причем каждому *k*-му кандидату соответствует вектор

$$v_k = \{i_1^k, i_2^k, i_3^k, \ldots . i_n^k\},\ k = 1, 2, \ldots, m,$$

где i_j^k – число голосов, поданных за то, чтобы *k*-й кандидат занял *j*-е место. В этом случае справедливы равенства:

$$\forall j \neq q: \sum_k i_j^k = \sum_k i_q^k = w, \tag{5}$$

где "*w*" – общее число голосов.

Очевидно, что эталону соответствует вектор $v_s = \{w, 0, 0, \ldots, 0\}$. Применительно к табл. 1 он равен: $v_s = \{13, 0, 0\}$. Победителем является *q*-й претендент, удовлетворяющий системе:

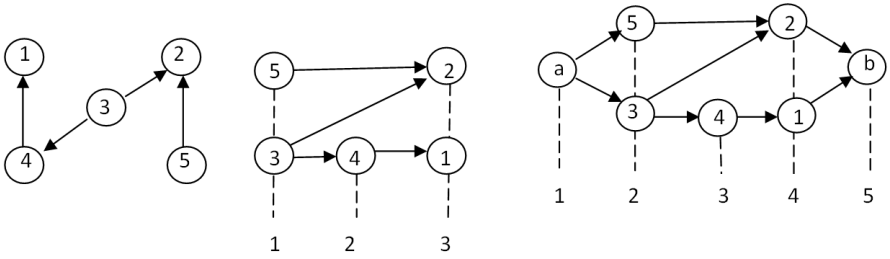
$$\begin{cases} \forall k: L_k = \sqrt{(w - i_1^k)^2 + \sum_{j=2}^{n} (i_j^k)^2}; \\ L_q = \min_k L_k. \end{cases} \tag{6}$$

Поиск решения системы (6) можно интерпретировать как выбор точки "*q*", расположенной ближе всего к эталону в *n*-мерном евклидовом пространстве. Возвращаясь к табл., 1 легко убедиться, что в соответствии с (12) победителем является "*a*".

5. Применение эталонов для обработки экспертных оценок

В тех случаях, когда получение количественных оценок ранжируемых объектов затруднено, их ранжирование может осуществляться на основе экспертных оценок. При этом для каждой пары объектов «*а*» и «*б*» эксперт выбирает одно из четырех высказываний: «*а*» лучше, чем «б»; «*б*» лучше, чем «*а*»; «*а*» эквивалентно «*б*» и «сравнительная оценка этих двух объектов невозможна» [5, 6]. Обычно используется графовая интерпретация такого множества оценок – вершины взвешенного ориентированного графа $G(X, U)$ соответствуют ранжируемым объектам, а дуги множества U – мнениям экспертов (рис. 3*а*):

- дуга (i, j) означает, что i-й объект «лучше» j-го;
- вес дуги (i, j), $r(i, j)$ определяется компетенцией эксперта.



а) Орграф $G(X, U)$

б) Разбиение вершин графа $G(X, U)$ на слои

в) модифицированный граф $G'(X', U')$ и распределение его вершин по слоям.

Рис. 3. Порядок преобразования исходного орграфа $G(X, U)$ в $G'(X', U')$

При этом наличие противоречий в экспертных оценках соответствует существованию на графе $G(X, U)$ непустого множества контуров $A(G)$, а выделение «наиболее весомого» подмножества непротиворечивых оценок экспертов отвечает решению задачи о минимальном разрезе в сильносвязном орграфе [7]:

$$\begin{cases} \sum_{i=1}^{n}\sum_{j=1}^{n} z(i,j)\cdot r(i,j) \to \min; \\ \forall a \in A(G): \sum_{(i,j)\in U(a)} z(i,j) \geq 1; \\ \forall (i,j) \in U : z(i,j) = 1{,}0; \end{cases} \quad (7)$$

где $z(i, j)$ – булева переменная, принимающая значение, равное единице, если мнение эксперта, полагающего, что i-й объект лучше j-го игнорируется; $U(a)$ – подмножество дуг графа $G(X, U)$, принадлежащих контуру $a \in A(G)$.

Если орграф $G(X, U)$ не содержит контуров, то всегда существует такая перестановка его n вершин $\pi = \{i_1, i_2, \ldots i_n\}$, для которой справедливо: если $k < j$, то i_k лучше, чем i_j. Так, орграфу, изображенному на рис. 3*а*, отвечает расположение вершин по слоям, представленное на рис. 3*б*, которому, в свою очередь, соответствуют перестановки, приведенные в таблице 2.

Таблица 2

i	i-я перестановка
1	3, 5, 1, 2, 4
2	5, 3, 1, 2, 4
3	3, 5, 1, 4, 2
4	5, 3, 1, 4, 2

Неоднозначность полученных упорядочений вершин может быть сокращена, благодаря следующей процедуре:

Шаг 1. В орграф $G(X, U)$ добавляются две фиктивные вершины "a" и "b", первая соответствует наилучшему эталону, вторая – наихудшему (рис. 3*в*).

Шаг 2. Каждой i-й вершине полученного орграфа ставится в соответствие вектор, содержащий два числа: первое представляет собой длину кратчайшего пути из вершины "a" в вершину "i", $r(a, i)$, а второе – длину максимального пути $r(i, b)$ из i-й вершины в "b".

Шаг 3. Для каждой вершины полученного графа $G'(X', U')$ вычисляется отношение $\eta(i) = r(a, i) / r(i, b)$, после чего, упорядочивая вершины по возрастанию величины η, получаем новое подмножество перестановок, мощность которого не превышает мощности множества упорядочений вершин исходного графа $G(X, U)$.

Так, реализуя шаги 2 и 3 применительно к графу $G'(X', U')$, изображенному на рис. 3*в*, получим табл. 3, которой соответствует единственное упорядочение вершин графа $G(X, U)$: $\pi = \{3, 5, 1, 2, 4\}$.

Таблица 3

Вершина	$r(a, i)$	$r(i, b)$	$\eta\ (i)$
a	0	4	0,0
1	3	1	3,0
2	2	1	2,0
3	1	3	0,333
4	2	2	1,0
5	1	2	0,5
b	3	0	∞

6. Использование эталонов при решении многокритериальных оптимизационных задач с булевыми переменными методами типа ветвей и границ

Далее используются следующие обозначения:

$\vec{X}$ – вектор переменных;

I – множество индексов переменных ($\left| I \right| = n$);

I_1 – множество индексов введенных в базис переменных ($\left| I_1 \right| \le \left| I \right|$);

$C_{j,i}$ – коэффициент j-го критерия, отвечающий i-й переменной;

X_k – множество значений, принимаемых k-й переменной;

$F_i(\vec{X})$ – i-й критерий (i = 1, 2, ….., n);

$\varphi_j(\vec{X})$ – j-е ограничение;

K_i – величина, соответствующая наилучшему значению i-го критерия.

Далее будем полагать, что решаются задачи с булевыми переменными, i-й критерий является аддитивным с неотрицательными коэффициентами, $F_i(\vec{X}) \to \max$ и $\left| I_1 \right| \ge 1$, а оценка его отклонения от величины K_i, равная $\beta_i(I_1)$ определяется выражением:

$$\beta_i(I_1) = \left[K_i - \left(\sum_{j \in I_1} C_{i,j} x_{i,j} + \sum_{j \in I \setminus I_1} C_{i,j} \right) \right]^2. \qquad (8)$$

Аналогично, если в задаче с булевыми переменными *i*-й критерий является аддитивным с неотрицательными коэффициентами, $F_i(\vec{X}) \to \min$ и $|I_1| \geq 1$, то далее полагаем, что оценка его отклонения от величины K_i, определяется выражением:

$$\beta_i(I_1) = \left[K_i - \sum_{j \in I_1} C_{i,j} x_{i,j} \right]^2. \qquad (9)$$

Далее формальная постановка многокритериальной задачи с булевыми переменными имеет вид:

$$\begin{cases} \forall i : \sum_j C_{i,j} x_{i,j} \to \max \ (\min); \\ \forall q : \varphi_q(\vec{X}) \leq \ b_q; \\ \forall k : x_k \in \{0,1\}. \end{cases} \qquad (10)$$

Тогда величина эталона K_i определяется решением однокритериальной задачи вида:

$$\begin{cases} K_i = \sum_j C_{i,j} x_{i,j} \to \max \ (\min); \\ \forall q : \varphi_q(\vec{X}) \leq b_q; \\ \forall k : x_k \in \{0,1\}. \end{cases} \qquad (11)$$

Решая задачу (9) применительно к каждому критерию, получаем вектор:

$$\vec{K} = \{K_1,\ K_2,\ .....,K_n\}, \tag{12}$$

которому в *n*-мерном пространстве критериев соответствует «идеальный» объект "*a*".

Если критерии однородны, т. е. измерены в совпадающих шкалах, то дистанция между объектами определяется, как расстояние в евклидовом пространстве, т. е., как квадратный корень из суммы квадратов разностей [5, 8]. Таким образом, задача (10) заменяется системой вида:

$$\begin{cases} \Delta = \sqrt{\sum_{i=1}^{n}[K_i - F_i(\vec{X})]^2} \to \min; \\ \forall j : \varphi_j(\vec{X}) \le b_j; \\ \forall k : x_k \in \{0,1\}; \quad \vec{X} = \{x_1, x_2, ..., x_m.\}. \end{cases} \tag{13}$$

Можно показать, что вектор переменных задачи (13) является Парето-оптимальным решением задачи (10) [9]. Более того, если нормировать значения целевых функций системы (10) таким образом, чтобы все они были безразмерны и заключены в диапазоне {0–1}:

$$\forall i : f_i(\vec{X}) = \frac{F_i(\vec{X}) - F_{i\min}(\vec{X})}{F_{i\max}(\vec{X}) - F_{i\min}(\vec{X})}, \tag{14}$$

то, подставляя левые части системы (14) в (13) вместо $F_i(\vec{X})$, легко убедиться [9], что вектор переменных полученной системы тоже является оптимальным по Парето решением системы (8).

Так как оптимальные векторы переменных в системе (15):

$$\begin{cases} \delta = \Delta^2 = \sum_{i=1}^{n} [K_i - \sum_j C_{i,j} x_{i,j}]^2 \to \min; \\ \forall q : \varphi_q(\vec{X}) \le b_q; \\ \forall k : x_k \in \{0,1\}; \quad \vec{X} = \{x_1, x_2, ..., x_m.\}, \end{cases} \tag{15}$$

и в системе (13) совпадают, далее для решения системы (10) используется (15). Для того, чтобы доказать применимость методов типа ветвей и границ для решения системы (15), достаточно показать, что по мере спуска по дереву ветвлений оценка величины δ, определяемая компонентами (9) и (10), не улучшается. Справедлива следующая теорема:

Теорема 2. Если, в рамках принятых выше допущений $I_1 \subseteq I_2$, то оценка $\beta_j(I_1)$ «лучше», чем оценка $\beta_j(I_2)$.

Доказательство теоремы 2 приводится в Приложении 2.

7. Оптимальная таксономия

Ниже используются следующие обозначения:

N – число таксонов;

$z(i, j)$ – булева переменная, равная единице, если i-й объект принадлежит j-ому 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.

Очевидно, что выбор критерия оптимальности в таксономии зависит от решаемой задачи. Приведем несколько примеров задач оптимальной таксономии [13].

Пусть m объектов следует распределить между n таксонами таким образом, чтобы суммарное расстояние между объектами, принадле-

жащими 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):

$$\begin{cases} \sum_{i=1}^{n}\sum_{p=1}^{m}\sum_{q\neq p} r(p,q)\cdot z(p,i)\cdot z(q,i) \to \min; \\ \forall p : \sum_{i=1}^{n} z(p,i) = 1 \\ \forall p, \forall i : z(p,i) = 1{,}0. \end{cases} \tag{16}$$

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 от объекта до центра «своего» таксона было минимальным. Формальная постановка такой задачи отличается от (16) только целевой функцией:

$$\begin{cases} \max\limits_{i}\max\limits_{p} \; r(c_i, p)\cdot z(p,i) \to \min; \\ \forall p : \sum_{i=1}^{n} z(p,i) = 1 \\ \forall p, \forall i : z(p,i) = 1{,}0. \end{cases} \tag{17}$$

Возможна модификация задачи (17). Пусть требуется распределить максимальное число объектов по *N* таксонам таким образом, чтобы расстояние от объекта до центра «своего» таксона не превышало величины R_i. Примером прикладной задачи, сводимой к сформулированной выше, является поиск оптимального распределения макси-

мального количества пользователей между передатчиками в системах эфирного вещания и сотовой связи, каждый из которых характеризуется радиусом действия R_i (система (18))

$$\begin{cases} \sum_{i=1}^{n}\sum_{p=1}^{m} Z(p,i) \to \max \\ \forall i, p : r(c_i, p)\cdot z(p,i) \le R_i \\ \forall p : \sum_{i=1}^{n} z(p,i) \le 1 \\ \forall p, \forall i : z(p,i) = 1,0. \end{cases} \quad (18)$$

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 оптимальности группировок. В общем виде для перечисленных выше критериев такая задача примет вид (19):

$$
\begin{cases}
F_1 = \sum_{i=1}^{n} \sum_{p=1}^{m} \sum_{q \neq p} r(p,q) \cdot z(p,i) \cdot z(q,i) \to \min; \\
F_2 = \sum_{i} \sum_{p} r(c_i, p) \cdot z(p,i) \to \min; \\
F_3 = \min_{i} \sum_{p} z(p,i) \to \max \\
F_4 = \max_{i} \max_{p} r(q,p) \cdot Z(q,i) \cdot Z(p,i) \to \min; \\
F_5 = \min_{i, j \neq i} \min_{p, q \neq p} r(p,q) \cdot z(p,i) \cdot z(q,j) \to \max; \\
F_6 = \max_{i} \max_{p} \; r(c_i, p) \cdot z(p,i) \to \min; \\
\forall p : \sum_{i=1}^{n} z(p,i) = 1 \\
\forall p, \forall i : z(p,i) = 1,0.
\end{cases}
\tag{19}
$$

Данная задача относится к задачам дискретной многокритериальной оптимизации с булевыми переменными. Для ее решения можно воспользоваться методами типа ветвей и границ, рассмотренными в предыдущем разделе.

8. Заключение

Обычно инструментарий, используемый для построения математических моделей, отличается от подходов, используемых для поиска решений на этих моделях. В этом плане место, занимаемое эталонами среди инструментов, применяемых для постановки и решения задач теории принятия решений, является уникальным. Выше было показано, как их использование позволяет формулировать новые задачи этой теории, по-новому определять старые и либо предложить новые процедуры их решения, либо расширить возможности применения существующих подходов. При этом следует отметить, что вышеприведенными задачами не исчерпывается применение эталонов: за рамками статьи остались задачи информатики, социологии, образования, экономики, и т.п., включающие распознавание образов, прогнозирование, цифровой фильтрации помех и ряд других, демонстрирующих возможность применения и высокую эффективность предлагаемого подхода.

Литература

1. *Podinovski V. V., Podinovskaya O. V.* New Multicriterial Decision Rules in Criteria Importance Theory // ISSN 1064-5624, Dollady Mathematics. 2013. Vol. 88, № 1. P. 486–488.
2. *Joseph J. Moder, Salah E.* Elmaghraby. Handbook of Operations Research // Vol. 1: Foundations and Fundamentals. *Van Nostrand Reinhold Company*, 1978. P. 513–548.
3. *Фон Нейман Дж., Моргенштерн О.* Теория игр и экономическое поведение. М.: Наука, 1970.
4. *Давыдов Е. Г.* Модели и методы теории антагонистических игр. Изд МГУ, 1978. 208 с.
5. Гроппен В. О. Принципы принятия решений с помощью эталонов // Автоматика и Телемеханика, № 4, 2006. С. 167–184.
6. *Будаева А. А., Гроппен В. О.* Выбор оптимальной технологии ранжирования // Устойчивое развитие горных территорий. 2014. № 3. С. 3–7.
7. *Бурков В. Н., Гроппен В. О.* Решение задачи о минимальном разрезе в бисвязном орграфе алгоритмами типа ветвей и границ // Автоматика и телемеханика. 1974. № 9.
8. *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.
9. *Groppen V. O.* New Solution Principles of Multi-Criteria Problems Based on Comparison Standards, www.arxiv.org/ftp/math/papers/0501/0501357.pdf , 2004.
10. *Dresher M., Shapley L. S., Tucker A.W., Eds.* Advances in Game Theory, Annals of Mathematics Study № 52. *Princeton University Press*, Princeton, New Jersey, 1964.
11. *Emerson, Peter*. Designing an All-Inclusive Democracy. Part 1. *Springer Verlag*, 2007. P. 15–38.
12. *John L. Daly*. Pricing for Profitability: Activity-Based Pricing for Competitive Advantage. *Wiley*, 2001.
13. *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.

Доказательство теоремы 1

Доказательство базируется на графической интерпретации поиска оптимальной цены игры. Так, матрице антагонистической игры двух лиц Г соответствует (рис. 4), ось абсцисс которого отображает выигрыши максимизирующего игрока, а ось ординат – проигрыши минимизирующего.

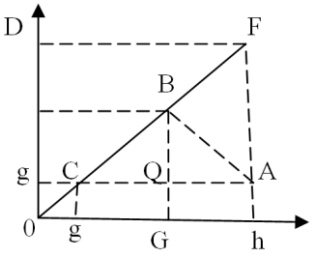


Рис. 4. Графическая интерпретация матричной игры двух лиц с нулевой суммой

В соответствии с принятыми ранее обозначениями, цене игры – эталону, удовлетворяющему обоих игроков, соответствует точка А, в то время как действительные выигрыши/проигрыши игроков принадлежат отрезку CF биссектрисы 0F прямого угла D0h. Очевидно, что кратчайшее расстояние между точкой А и отрезком CF определяется перпендикуляром BA, опущенным из точки А на 0F, причем точка В определяет оптимальную цену игры Г, которая может отличаться от цены той же игры, полученной с помощью гарантирующих стратегий. Так как треугольник CBA является прямоугольным и равнобедренным , длина гипотенузы CA известна и равна $(h - g)$, в то время, как длина катетов равна BA. Но, одновременно катет BA является гипотенузой прямоугольного и равнобедренного треугольника QBA, что позволяет определить размер его катетов $z = 0{,}5\ (h - g)$. Так как длина отрезков CQ, QB и QA совпадает, длина отрезка 0G определяет оптимальную цену игры Г, равную $z + g$:

$$z + g = 0{,}5\ (h + g). \qquad (20)$$

Очевидно, что оба игрока должны следовать стратегиям, определяемым ячейкой $M\ (p, q)$, содержимое которой является ближайшим к цене игры, полученной в правой части (20). Последнее определяется выражением (3). Теорема 1 доказана.

Приложение 2

Доказательство теоремы 2

Ниже рассмотрены два варианта *j*-го критерия: максимизируемый и минимизируемый.

1. $F_j(\vec{X}) \to \max$. Тогда компоненты оценки, соответствующие *j*-у критерию и отвечающие базисам I_1 и I_2, соответственно равны:

$$\begin{cases} \beta_j(I_1) = (K_j - \sum\limits_{i \in I_1} C_{i,j} x_{i,j} - \sum\limits_{i \in I \setminus I_1} C_{i,j})^2; \\ \beta_j(I_2) = (K_j - \sum\limits_{i \in I_2} C_{i,j} x_{i,j} - \sum\limits_{i \in I \setminus I_2} C_{i,j})^2; \\ I_1 \subseteq I_2. \end{cases} \quad (21)$$

Благодаря последнему условию системы (21), оценки $\beta_j(I_t)$, $t = 1, 2$ могут быть преобразованы к виду:

$$\begin{cases} \beta_j(I_1) = (K_j - \sum\limits_{i \in I_1} C_{i,j} x_{i,j} - \sum\limits_{i \in I_2 \setminus I_1} C_{i,j} - \sum\limits_{i \in I \setminus I_2} C_{i,j})^2; \\ \beta_j(I_2) = (K_j - \sum\limits_{i \in I_1} C_{i,j} x_{i,j} - \sum\limits_{i \in I_2 \setminus I_1} C_{i,j} x_{i,j} - \sum\limits_{i \in I \setminus I_2} C_{i,j})^2. \end{cases} \quad (22)$$

Но в соответствии с принятыми выше допущениями справедливо неравенство:

$$\sum_{i \in I_2 \setminus I_1} C_{i,j} \geq \sum_{i \in I_2 \setminus I_1} C_{i,j} x_{i,j}. \quad (23)$$

Отсюда следует:

$$\beta_j(I_2) \geq \beta_j(I_1). \quad (24)$$

2. Пусть теперь $F_j(\vec{X}) \to \min$. Тогда компоненты оценок, соответствующие *j*-у критерию и отвечающие базисам I_1 и I_2, соответственно равны:

$$\begin{cases} \beta_j(I_1) = (K_j - \sum\limits_{i \in I_1} C_{i,j} x_{i,j})^2; \\ \beta_j(I_2) = (K_j - \sum\limits_{i \in I_2} C_{i,j} x_{i,j})^2; \\ I_1 \subseteq I_2. \end{cases} \quad (25)$$

Благодаря тому, что в этом случае $K_j = 0$, а также благодаря последнему условию системы (25), оценки $\beta_j(I_t)$, $t = 1, 2$, могут быть преобразованы к виду:

$$\begin{cases} \beta_j(I_1) = (\sum\limits_{i \in I_1} C_{i,j} x_{i,j})^2; \\ \beta_j(I_2) = (\sum\limits_{i \in I_1} C_{i,j} x_{i,j} + \sum\limits_{i \in I_2 \setminus I_1} C_{i,j} x_{i,j})^2. \end{cases} \quad (26)$$

Но в соответствии с принятыми выше допущениями справедливо неравенство:

$$\sum_{i \in I_2 \setminus I_1} C_{i,j} x_{i,j} \geq 0. \quad (27)$$

Отсюда следует: $\beta_j(I_2) \geq \beta_j(I_1)$. Таким образом, независимо от критериев, входящих в состав системы (10), по мере спуска по дереву ветвлений оценки величины δ в системе (15) не улучшаются. Теорема доказана.

Сведения об авторах

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

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

УДК 519.834

Гроппен В. О.

КОМПОЗИТНЫЕ АЛГОРИТМЫ ПОИСКА ГЛОБАЛЬНО ОПТИМАЛЬНЫХ РЕШЕНИЙ ЭКСТРЕМАЛЬНЫХ ЗАДАЧ С БУЛЕВЫМИ ПЕРЕМЕННЫМИ

Предложен новый класс алгоритмов, предназначенных для поиска глобально оптимальных решений задач дискретной оптимизации с булевыми переменными. Показано, что в рамках сделанных допущений быстродействие предложенных процедур не ниже быстродействия используемых в этом случае «классических» алгоритмов. Работа всех предлагаемых алгоритмов иллюстрируется примерами.

Ключевые слова: *глобальный оптимум, булевы переменные, дискретная оптимизация, динамическое программирование, методы отсечения планов, методы типа ветвей и границ.*

1. Введение

Рост числа различных приложений систем искусственного интеллекта [1–3] повышает требования к используемым в рамках этой технологии оптимизационным алгоритмам. Это особенно важно применительно к алгоритмам поиска глобально оптимальных решений задач дискретной оптимизации, для которых отсутствуют эффективные процедуры, т. е. такие, время счета которыми полиномиально зависит от размерности задачи [4; 8]. К классическим алгоритмам поиска глобально оптимальных решений экстремальных задач дискретной оптимизации можно отнести полный перебор и методы, развитые во второй половине прошлого века: алгоритмы типа ветвей и границ и динамическое программирование [5–8]. До последнего времени повышение их интеллекта было связано с распараллеливанием вычислений и совершенствованием способов вычисления оценок, адаптируя их применительно к специфике решаемых задач [3; 9; 10]. Ниже анализируется интеллект композитных алгоритмов, ориентированных на решение задач дискретной оптимизации с булевыми переменными. По аналогии с композитными материалами [12–15], под композитными алгоритмами ниже понимаются процедуры, в которых присутствуют компоненты не менее двух классических алгоритмов. Так, ниже про-

цедура выбора направления спуска по дереву ветвлений в методах типа ветвей и границ дополняется технологией отсечения «плохих» планов, развитой в динамическом программировании, а процедуры отсечения, используемые в динамическом программировании, расширяются за счет применения оценок, присущих методам типа ветвей и границ.

Существуют различные способы сравнительной оценки «интеллекта» алгоритмов, используемых для принятия решений с помощью математических моделей [16; 17]. Применительно к процедурам, гарантирующим глобально оптимальные решения, мерой их интеллекта могут служить затраты ресурсов компьютера, например время поиска решения и объем использованной при этом памяти: если алгоритм A находит решение i-й задачи за время $t(A, i)$, причем объем использованной при этом оперативной памяти не превышает $V(A, i)$, а алгоритм B находит решение той же задачи за время $t(B, i)$ при соответствующих затратах памяти, равных $V(B,i)$, причем справедлива система неравенств:

$$\begin{cases} \forall i : t(A,i) \geq t(B,i); \\ \forall i : V(A,i) \geq V(B,i) \end{cases}, \tag{1}$$

то интеллект алгоритма B выше, чем интеллект алгоритма A. Полагая, что существует некий эталонный алгоритм, позволяющий находить глобально оптимальные решения задач с нулевыми затратами машинных ресурсов, количественные оценки интеллекта алгоритмов такого рода применительно к некоторой i-й задаче можно определять расстоянием $R(C, i)$ от начала координат точки «C», характеризующей некоторый алгоритм «C» в системе координат t_0V параметрами $t'(C, i)$ и $V'(C,i)$, определяемыми на основании системы [18]:

$$\begin{cases} R(C,i) = \sqrt{[t'(C,i)]^2 + [V'(C,i)]^2}; \\ t'(C,i) = \dfrac{t(C,i)}{t_{\max}}; \\ V'(C,i) = \dfrac{V(C,i)}{V_{\max}}; \\ t_{\max} = \max\limits_{A \in \{A\}} t(A,i); \\ V_{\max} = \max\limits_{A \in \{A\}} V(A,i), \end{cases} \tag{2}$$

где $\{A\}$ – множество алгоритмов, использованных для решения i-й задачи.

Далее (2) используется для исследования сравнительного интеллекта классических и композитных процедур поиска глобально оптимальных решений экстремальных задач с булевыми переменными, причем применяются следующие обозначения, допущения и определения.

2. Обозначения, допущения и определения

Ниже анализируется сравнительный интеллект различных алгоритмов поиска решения однокритериальных задач с булевыми переменными вида:

$$\begin{cases} F = \sum_i C_i z_i \to \text{extr}; \\ \forall j : \sum_i b_{i,j} z_i \le a_j; \\ \forall i : z_i = 1,0, \end{cases} \qquad (3)$$

где: $\forall i,\ \forall j$: C_i и $b_{i,j}$ – коэффициенты,

a_j – константы, причем далее полагаем, что все коэффициенты и константы неотрицательны. К такому виду могут быть сведены такие популярные задачи дискретной оптимизации, как задача коммивояжера, задача о ранце, разрыв контуров на взвешенном орграфе и т. п. [5, 8, 11]. При этом все численные примеры, иллюстрирующие эффективность предлагаемых подходов, ниже приводятся для случая, когда $F \to \max$, $j = 1$, что отвечает задаче о ранце вида [3]:

$$\begin{cases} F = 7z_1 + 3z_2 + 5z_3 + 9z_4 + 2z_5 \to \max; \\ 3z_1 + 4z_2 + 2z_3 + 7z_4 + 6z_5 \le 10; \\ z_i = 1, 0;\ i = 1, 2, \ldots, 5. \end{cases} \qquad (4)$$

Далее, применительно к методам неявного перебора, осуществляющим поиск решения на множестве частичных планов, используются следующие определения:

1. «Висячей» вершиной на любой итерации считается такая вершина построенной части дерева ветвлений, из которой не исходят дуги.

2. Корневая вершина дерева ветвлений считается «висячей» вершиной построенной части дерева на первой итерации.

3. Случаю, когда анализируется единовременное введение в базис h ($h \geq 1$) переменных, отвечает «куст» с высотой h. Таким «кустом» с корнем в вершине $x_k \in X$ является ориентированный подграф, обладающий следующими свойствами (см. рис. 1 ниже):

- в каждую вершину куста, за исключением корневой, входит только одна дуга;
- из каждой вершины куста, за исключением множества висячих вершин, исходят две дуги;
- из вершины $x_k \in X$, в каждую висячую вершину куста существует только один путь, число дуг которого равно h;
- число вершин куста равно $2^{h+1} - 1$, число висячих вершин равно 2^h, число дуг куста определяется выражением: $2(2^h - 1)$;
- все рассмотренные ниже примеры в разделах 3 – 6 используют случаи, когда число единовременно вводимых в базис переменных $h = h_1 = 1$, только в примере, рассмотренном в седьмом разделе, $h = h_2 = 2$ (см. рис. 1 и рис. 6).

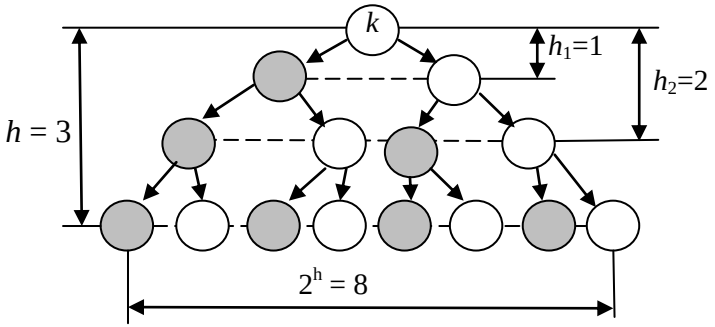


Рис. 1. Пример куста для случая $h = 3$, «серые» вершины отвечают значениям соответствующих переменных, равным единице, «белые» – равным нулю

4. Ветвлением из заданной вершины $x_k \in X$, которой соответствует подмножество введенных в базис переменных $I(x_k)$, является построение «куста» с корнем в этой вершине, причем каждой из 2^h висячих вершин этого куста соответствует $\left| I(x_k) \right| + h$ введенных в базис переменных.

5. Верхняя оценка $\Delta(x_k)$ величины F, отвечающей вершине $x_k \in X$, соответствующей вектору переменных задачи (3), в базис ко-

торого введены $I_1(x_k)$ переменных, далее определяется следующим образом:

$$\Delta(x_k)=\begin{cases} \sum\limits_{i\in I_1(x_k)} C_i z_i + \sum\limits_{j\in I\setminus I_1(x_k)} C_j, \text{ если ограничения системы (3) выполняются;} \\ -\infty, \text{ в противном случае,} \end{cases} \quad (5)$$

где I – множество индексов всех переменных.

6. Аналогично определяется нижняя оценка $\delta(I_1)$ величины F задачи (3):

$$\delta(x_k) = \begin{cases} \sum\limits_{i\in I_1(x_k)} C_i z_i, \text{ если ограничения системы (3) выполняются;} \\ +\infty, \text{ в противном случае, если } F \to \min; \\ -\infty, \text{ в противном случае, если } F \to \max. \end{cases} \quad (6)$$

В то время как нижняя граница, определяемая на основании (6), в силу сделанных выше допущений, не зависит от специфики задачи, величина $\Delta(x_k)$ непосредственно связана с этой спецификой: выражение (5) справедливо только для задачи о ранце. Примеры способов вычисления верхней оценки $\Delta(x_k)$ применительно к ряду других задач дискретной оптимизации, приведены в [3; 5; 8; 11]. Далее полагаем, что время счета в первом приближении линейно зависит от числа вершин построенного в ходе поиска решения дерева ветвлений $G(X, U)$, где X – множество вершин, U – множество дуг.

Общей чертой приводимых графических иллюстраций поиска решения системы (4) и рис. 1 является кодировка вершин дерева ветвлений: «серые» вершины дерева отвечают значениям соответствующих переменных, равным единице, «белые» – равным нулю. Вершине последнего яруса с «жирным» контуром соответствует оптимальный вектор переменных Z. Остальные обозначения вводятся по мере надобности.

3. Методы неявного перебора на частичных планах, гарантирующие глобально оптимальное решение

Ниже рассмотрено два метода такого рода: динамическое программирование и методы типа ветвей и границ.

3.1. Метод типа ветвей и границ, осуществляющий фронтальный спуск по дереву ветвлений

Содержательно работа такого алгоритма может быть представлена последовательным построением дерева ветвлений $G(X,U)$ (см. рис. 2):

Алгоритм 1

1. На множестве висячих вершин $X_1 \subseteq X$ построенной части дерева ветвлений $G(X,U)$ выбирается вершина x_j с наилучшей оценкой. Если это осуществляется на первой итерации, то такой вершиной *a priori* считается корневая вершина дерева.

2. Если выбранной вершине отвечает равенство $I_1 = I$, то нужно перейти к шагу 5, в противном случае – к следующему шагу.

3. Ветвление осуществляется из выбранной на шаге 1 последней итерации вершины x_j. Новое множество висячих вершин дерева вновь обозначаем X_1.

4. Вычисляются оценки, отвечающие висячим вершинам построенного на предыдущем шаге «куста». Для этого можно воспользоваться (4), если целевая функция системы (2) является максимизируемой и (5) – если целевая функция минимизируется. Перейти к шагу 1.

5. Алгоритм закончен. Вектор переменных, соответствующий выбранной вершине, является оптимальным.

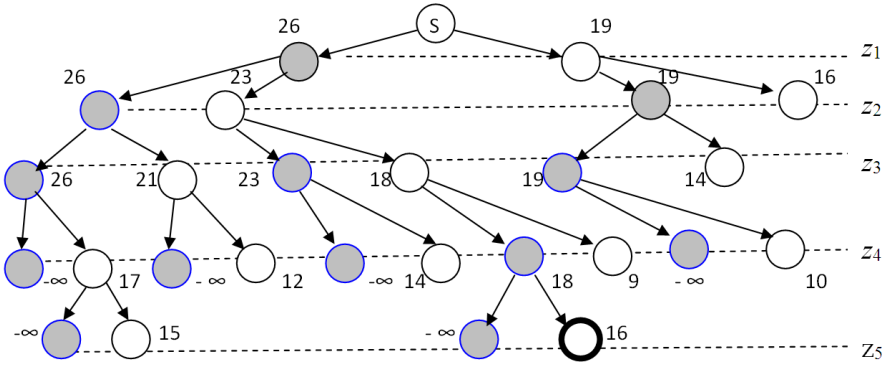


Рис. 2. Дерево ветвлений $G_1(X_1,U_1)$, построенное алгоритмом 1 применительно к задаче (3)

Оптимальный вектор переменных задачи (3), $Z = \{1,0,01,0\}$, соответствующее значение целевой функции $F = 16$, число вершин дерева $G_1(X_1, U_1)$, $|X_1| = 27$, число висячих вершин дерева ветвлений $X_1^h \subseteq X_1$ в ходе поиска не превышало шести.

3.2. Динамическое программирование

Содержательно решение задачи дискретной оптимизации с помощью динамического программирования может быть проиллюстрировано последовательным, ярус за ярусом, построением дерева ветвлений $G_2(X_2, U_2)$ (рис. 3), стратегия которого существенно отличается от стратегии, описанной в приведенном выше алгоритме 1. Ниже приводится агрегированное описание процедуры такого рода при условии, что все коэффициенты системы (2) неотрицательны.

Алгоритм 2

1. На множестве висячих и не помеченных вершин $X' \subseteq X_2$ построенной части дерева ветвлений $G_2(X_2, U_2)$ выбирается вершина x_j. Если таковой вершины нет, то необходимо перейти к шагу 4. Если выбор осуществляется на первой итерации, то такой вершиной *a priori* считается корневая вершина дерева.

2. Осуществляется ветвление из выбранной на шаге 1 последней итерации вершины x_j и вычисляются компоненты каждого вектора $R(x_k)$, отвечающего всем висячим вершинам построенного «куста» (величина $h = 1$). Для определения первой компоненты этого вектора используется (6), остальные компоненты вычисляются по формуле:

$$\forall j : r_j(x_k) = a_j - \sum_{q \in I_1(x_k)} b_{q,j} z_q, \qquad (7)$$

где x_k – вершина, принадлежащая множеству висячих вершин куста, в которую заходит дуга из выбранной на шаге 1 последней итерации вершины x_j.

3. Все вершины куста, построенного на шаге 2 последней итерации, помечаются, далее – перейти к шагу 1.

4. Убираются все пометки подмножества висячих вершин дерева ветвлений.

5. Если на множестве висячих вершин существует вершина x_k, для которой справедливо хотя бы одно из условий:

$$\begin{cases} \min\limits_j r_j(x_k) < 0; \\ \text{существует вершина } x_q, \text{ для которой справедливы ограничения } а)-в): \end{cases}$$

а) $I_1(x_k) = I_1(x_q)$;

б) $\delta(x_k)$ «хуже» чем $\delta(x_q)$;

в) $\forall j : r_j(x_q) \geq r_j(x_k)$,

то эта вершина вычеркивается.

6. Если число введенных в базис переменных висячих вершин дерева ветвлений равно числу переменных решаемой задачи, то можно перейти к следующему шагу, в противном случае перейти к шагу 1.

7. На множестве не вычеркнутых висячих вершин выбирается вершина x_k с наилучшей первой компонентой соответствующего ей вектора $R(x_k)$.

8. Алгоритм закончен. Вектор переменных Z, соответствующий выбранной на шаге 7 вершине x_k, является оптимальным.

Ниже (рис. 3) изображено дерево $G_2(X_2, U_2)$, построенное с помощью алгоритма 2 применительно к задаче (3), векторы у вершин определены на шагах 3 каждой итерации. При этом перечеркнуты вершины, вычеркнутые на шаге 6 алгоритма 2, и отсутствуют векторы перечеркнутых вершин, для которых справедливо условие в) шага 6, причем число вершин построенного дерева $G_2(X_2, U_2)$, $|X_2| = 35$, а подмножество вершин этого дерева $X_2^h \subseteq X_2$, численные характеристики которых единовременно присутствовали в оперативной памяти компьютера, не превышало пятнадцати ($\left|X_1^h\right| \le 15$).

Пользуясь системой (2) для сравнения интеллектов первого и второго алгоритмов применительно к задача (4) и учитывая, что $\{A\}=\{1, 2\}$, получим: $R(1,4) = 0{,}8685$; $R(2,4) = 1{,}4142$, что говорит в пользу первого алгоритма.

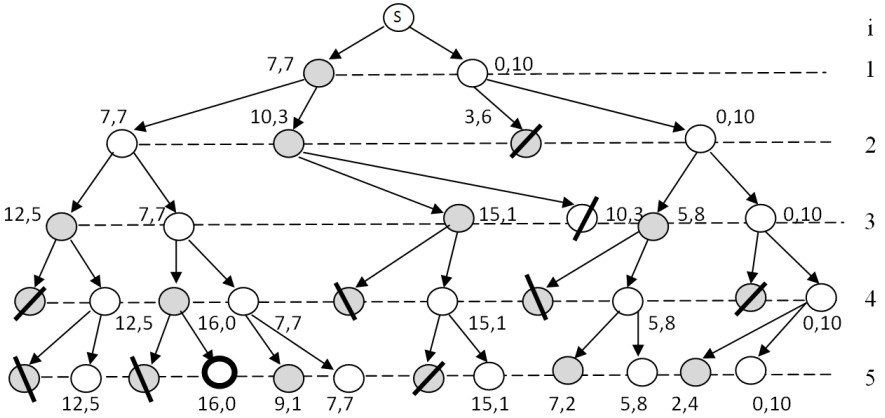


Рис. 3. Дерево ветвлений $G_2(X_2,U_2)$, построенное с помощью алгоритма 2 применительно к задаче (3)

4. Композитные алгоритмы, реализующие неявный перебор на частичных планах и гарантирующие глобально оптимальное решение

Ниже приводятся:

- модификация методов типа ветвей и границ, методика отсечения в которой включает технологии, используемые в динамическом программировании;
- модификация алгоритма, реализующего динамическое программирование с привлечением оценок, используемых методами типа ветвей и границ.

4.1. Композитная реализация метода типа ветвей и границ

В рамках реализации этого подхода каждой вершине x_j строящегося дерева ветвлений $G_3(X_3, U_3)$ ставится в соответствие вектор $V(x_j)$, первой компонентой которого является оценка, используемая в традиционной реализации этого алгоритма, а остальные компоненты совпадают с компонентами вектора $R(x_j)$, описание которого приводится в предыдущем разделе, посвященном динамическому программированию. Ниже приводится содержательное описание такой процедуры.

Алгоритм 3

1. На множестве не вычеркнутых висячих вершин $X_3^T \subseteq X_3$ построенной части дерева ветвлений $G(X, U)$ выбирается вершина x_j с «наилучшей» первой компонентой вектора $V(x_j)$. Если это осуществляется на первой итерации, то такой вершиной *a priori* считается корневая вершина дерева.

2. Если выбранной вершине отвечает равенство $I_1 = I$, то необходимо перейти к шагу 8, в противном случае – к следующему шагу.

3. Ветвление осуществляется из выбранной на шаге 1 последней итерации вершины x_j. Новое множество висячих вершин дерева вновь обозначаем X_3^T.

4. Для каждой висячей вершины построенного на предыдущем шаге «куста» x_j формируется вектор $V(x_j)$. Для формирования первой компоненты этого вектора используется процедура, применявшаяся для вычисления соответствующей оценки в методе типа ветвей и границ, для определения второй компоненты вектора $V(x_j)$ используется (5), остальные компоненты вычисляются по формуле (6).

5. Если на множестве висячих вершин X_3^T существует вершина x_k, для которой справедливо хотя бы одно из условий:

$$\begin{cases} \min\limits_j r_j(x_k) < 0; \\ \text{существует вершина } x_q \in X_3^T, \text{ для которой справедливы ограничения } а)-в): \end{cases}$$

а) $I_1(x_k) = I_1(x_q)$;

б) $\delta(x_k)$ «хуже» чем $\delta(x_q)$;

в) $\forall j : r_j(x_q) \geq r_j(x_k)$,

то эта вершина вычеркивается.

6. Если существуют вершины $x_j \in X_3^T$ и $x_q \in X_3^T$, для которых справедливо:

а) $I_1(x_j) = I_1(x_q)$; б) $\delta(x_j)$ «лучше» чем $\Delta(x_q)$, то вершина x_q вычеркивается.

7. Перейти к шагу 1.

8. Алгоритм закончен. Вектор переменных, соответствующий выбранной вершине, является оптимальным.

На рис. 4 изображено дерево $G_3(X_3, U_3)$, построенное алгоритмом 3 применительно к задаче (4).

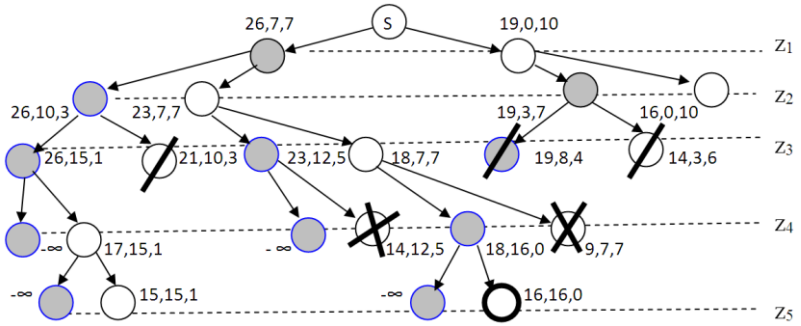


Рис. 4. Дерево ветвлений $G_3(X_3, U_3)$, построенное алгоритмом 3 при решении задачи (3)

Число вершин построенного дерева ветвлений $G_3(X_3, U_3)$, $|X_3|$=23, причем однократно перечеркнуты вершины, отвечающие применению шага 5, а дважды перечеркнутые вершины отображают реализацию шага 6. Верхняя граница числа вершин, принадлежащих фронту висячих вершин, не превышала шести ($\left|X_3^h\right| \leq 6$). Учитывая, что теперь множество $\{A\} = \{1,2,3\}$, применительно к задаче (4) ин-

теллект $R(3,4)$ метода типа ветвей и границ, осуществляющего фронтальный спуск по дереву ветвлений с привлечением технологий отсечения, присущих динамическому программированию, определяется выражением: $R(3,4) = 0{,}76918$. Это соответствует относительному увеличению интеллекта алгоритмом 3 по сравнению с традиционной реализацией метода типа ветвей и границ алгоритмом 1 равному: $\eta(1,3) = [R(1,4) - R(3,4)] / R(1,4) \approx 14{,}4\ \%$.

4.2. Композитная реализация динамического программирования

Предлагаемый далее подход отличается от приведенного выше алгоритма 2 следующими параметрами:

1. Каждой вершине x_j дерева ветвлений ставится в соответствие вектор $V(x_j)$, аналогичный тому, который использовался в алгоритме 3.

2. Если на i-й итерации i-у ярусу построенной части дерева ветвлений принадлежат две такие вершины x_j и x_q , у которых вторая компонента вектора $V(x_j)$ «лучше» первой компоненты вектора $V(x_q)$, то вершина x_q вычеркивается. Практически эта операция совпадает с шагом 6 алгоритма 3.

Ниже приводится содержательное описание композитной реализации динамического программирования, включающей алгоритм 2 и приведенные выше параметры.

Алгоритм 4

1. На множестве висячих не вычеркнутых и не помеченных вершин $X' \subseteq X_4$ построенной части дерева ветвлений $G_4(X_4,U_4)$ выбирается вершина x_j с наилучшей первой компонентой вектора $V(x_j)$. Если таковой вершины нет, то перейти к шагу 5. Если выбор осуществляется на первой итерации, то такой вершиной *a priori* считается корневая вершина дерева $G_4(X_4, U_4)$.

2. Если выбранной вершине x_j отвечает число переменных, введенных в базис, равное числу переменных решаемой задачи, то нужно перейти к шагу 9, в противном случае – перейти к шагу 3.

3. Осуществляется ветвление из выбранной на шаге 1 последней итерации вершины x_j и вычисляются компоненты каждого вектора $V(x_k)$, отвечающего всем висячим вершинам построенного «куста». Процедура вычисления компонент этого вектора совпадает с процедурой, описанной на шаге 4 алгоритма 3.

4. Пометить все полученные на шаге 3 последней итерации висячие вершины куста и перейти к шагу 1.

5. Убрать пометки всех висячих вершин построенной части дерева ветвлений $G_4(X_4, U_4)$.

6. Если на множестве висячих вершин дерева $G_4(X_4, U_4)$ существует вершина x_k, для которой справедливо хотя бы одно из условий:

$$\begin{cases} \min\limits_j r_j(x_k\} < 0; \\ \text{существует вершина } x_q, \text{ для которой справедлив ы ограничени я } а) - в): \end{cases}$$

а) $I_1(x_k) = I_1(x_q)$;

б) $\delta(x_k)$ «хуже» чем $\delta(x_q)$;

в) $\forall j : r_j(x_q) \geq r_j(x_k)$,

то вершина x_k вычеркивается.

7. Если на множестве висячих вершин построенной части дерева $G_4(X_4, U_4)$ существуют хотя бы две такие вершины x_j и x_q , у которых вторая компонента вектора $V(x_j)$ «лучше» первой компоненты вектора $V(x_q)$, то вершина x_q вычеркивается.

8. Перейти к шагу 1.

9. Алгоритм закончен. Вектор переменных Z, соответствующий выбранной на шаге *1* последней итерации вершине x_j, является оптимальным.

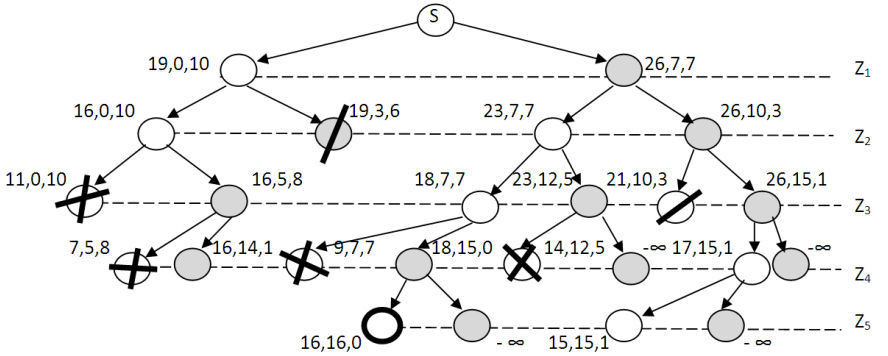


Рис. 5. Дерево ветвлений, иллюстрирующее поиск решения задачи (3) алгоритмом 4

На рис. 5 изображено дерево $G_4(X_4, U_4)$, построенное с помощью алгоритма 4 применительно к задаче (4), векторы у вершин определены на шагах 3 каждой итерации. Знаком « – ∞» помечены вершины, отвечающие векторам переменных, нарушающим ограничения системы (3), однократно перечеркнуты вершины, отвечающие применению шага 6, а дважды перечеркнутые вершины отображают реализацию шага 7. Легко убедиться, что удаление из алгоритма 4 шагов 4 и 5,

преобразует его в алгоритм 3. Так как на этих шагах генерируются и удаляются пометки, ограничивающие реализацию шагов 6 и 7, предназначенных для отсечения «плохих» направлений спуска по дереву ветвлений, можно утверждать, что применительно к одной и той же задаче дискретной оптимизации дерево $G_4(X_4, U_4)$ будет содержать не меньше вершин, чем дерево $G_3(X_3, U_3)$. Иными словами $\left| X_4 \right| \geq \left| X_3 \right|$. В соответствии с системой (2) применительно к задаче (4), учитывая, что $\{A\} = \{1,2,3,4\}$, интеллект алгоритма, реализующего динамическое программирование с привлечением технологий оценки перспективности направлений спуска по дереву ветвлений, присущих методам типа "ветвей и границ", равен $R(4, 4) = 1{,}0538$. Сравнивая эту величину с ранее полученными оценками, легко убедиться:

- относительное увеличение интеллекта алгоритма 4 по сравнению с алгоритмом 2 равно:

$$\eta(2,4) = [R(2,4) - R(4,4] / R(2,4) \approx 25\ \%;$$

- на множестве $\{A\} = \{1, 2, 3, 4\}$рассмотренных выше алгоритмов наибольший интеллект применительно к задаче (4) продемонстрировал алгоритм 3.

5. Поиск глобально оптимального решения на полных планах

Поиск глобально оптимальных решений экстремальных задач с булевыми переменными на полных планах связан с последовательным анализом 2^n их вариантов, где n – число переменных. Учитывая, что при полном последовательном переборе в памяти постоянно присутствует только два вектора переменных, применительно к задаче (4) интеллект такого подхода равен 0,9236.

Сокращение объема перебора может быть достигнуто разбиением всего множества полных планов на 2^h подмножеств ($h < n$), каждое из которых анализируется на предмет отсечения с помощью рассмотренных в предыдущем разделе методов. Графически это можно интерпретировать построением дерева, содержащего два яруса вершин, причем первому ярусу соответствует только корневая вершина, а второму – 2^h вершин, каждая из которых отвечает «своему» подмножеству полных планов с совпадающими значениями первых h переменных (рис. 6). При этом каждой вершине второго яруса x_j, $j \in \{0 \div (2^h - 1)\}$, ставится в соответствие вектор $V(j) = \{v_{1,j}, v_{2,j}, v_{3,j} \ldots\}$, формирование которого описано в шаге 3 приводимого ниже алгоритма 5.

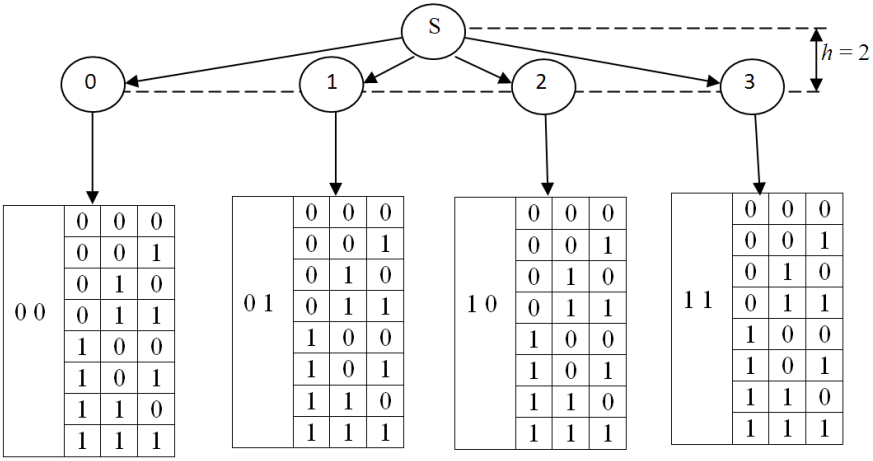


Рис. 6. Разбиение полных планов на подмножества для случая, когда $n = 5$, $h = 2$

В приводимом ниже содержательном описании композитной версии полного перебора выражение "$a \succ b$".означает, что объект «*a*» лучше чем объект «*b*».

Алгоритм 5

1. Величине *W* присваивается значение, равное + ∞, если целевая функция задачи (2) минимизируется, и значение, равное –∞ в противном случае.

2. Множество всех полных планов разбивается на 2^h подмножеств, где *h* – число зафиксированных значений вектора булевых переменных *Z* (см. рис. 1).

3. Каждому полученному на предыдущем шаге подмножеству присваивается номер

$$"j": \ j = \sum_{i=0}^{i=h-1} z_i 2^i. \qquad (8)$$

4. Каждому *j*-у подмножеству ставится в соответствие вектор *V*(*j*), компоненты которого определяются следующим образом: для формирования первой компоненты используется процедура вычисления оценки, применяемая в методах типа ветвей и границ, при условии, что первые h компонент вектора переменных *Z* фиксированы и определяются двоичным представлением числа *j*. Для определения второй компоненты вектора *V*(*j*) используется (5), остальные компоненты вычисляются по формуле (6).

5. Если существуют два таких вектора j и k, для которых справедливо: $\forall i > 1 : v_{i,j} \succ v_{i,k}$, то *k*-е подмножество исключается из числа рассматриваемых.

6. Осуществляется упорядочение $\pi = \{j_1, j_2, \ldots\}$ оставшихся после реализации шага 4 подмножеств по мере ухудшения первой компоненты вектора $V(j)$: $\forall i < 2^h : v_{i,j} \succ v_{i,j+1}$.

7. $q = 1$.

8. Исследуется q-е подмножество полных планов перестановки π. Фиксируется наилучший найденный полный план $_q$ и соответствующее значение целевой функции F_q. Если $F_q \succ W$, то прежнее значение *W* забывается, а новое, равное F_q, запоминается.

9. Если существует вектор $V(j)$, отвечающий необследованному подмножеству полных планов, для которого справедливо: $W \succ v_{j,1}$, то *j*-е подмножество полных планов исключается из числа рассматриваемых, а индексы всех следующих за ним векторов уменьшаются на единицу.

10. Если существует не анализировавшееся на предыдущих шагах подмножество полных планов, то перейти к шагу 11, в противном случае – к шагу 12.

11. $q = q + 1$, перейти к шагу 8.

12. Конец алгоритма. Оптимальное значение целевой функции равно W.

Пример применения алгоритма 5 при решении приведенной ранее задачи (4) приводится ниже. При этом следует учитывать, что первый шаг иллюстрируется рисунком 6.

Шаги 2 и 3 – на них формируются векторы $V(j)$, $j = 0, 1, 2, 3$, которые представлены на рис. 7 *а–г*:

Шаг 4: Поскольку все компоненты вектора $V(2)$ лучше одноименных компонент вектора $V(1)$, все полные планы, первые две компоненты которых равны, соответственно, «0» и «1», исключаются из числа анализируемых.

j = 0	
0 0	16,0,10

а)

j = 1	
0 1	19,3,6

б)

j = 2	
1 0	23,7,7

в)

j = 3	
11	26,10,3

г)

Рис. 7. Векторы, характеризующие подмножества полных планов задачи (5), образованные частичными планами с введенными в базис двумя первыми переменными ($h = 2$)

Шаг 5. Перестановка π оставшихся векторов: $\pi = V(3), V(2), V(0)$.

Шаги 6 и 7. $q = 1$. Результаты анализа всех полных планов, первые две компоненты которых равны единице: $F_{max} = 15$, $Z_{opt} = 11100$, $R = 15$.

Шаги 10 и 7. $q = 2$. Результаты анализа всех полных планов, первые две компоненты которых соответственно равны единице и нулю: $F_{max} = 16$, $Z_{opt} = 10010$, $R = 16$.

Шаг 8. Поскольку первая компонента вектора $V(0)$ не больше величины R, подмножество полных планов, первые две компоненты которых являются нулевыми, исключается из рассмотрения. Поскольку все множество полных планов исчерпано, переход к шагу 11.

Шаг 11. Конец алгоритма. Полученные оптимальные значения целевой функции и вектора переменных: $F_{max} = 16$, $Z_{opt} = 10010$.

Легко убедиться, что интеллект этого алгоритма $R(5, 4) = 0{,}6615$, что соответствует относительному приращение им интеллекта при решении задачи (4) по сравнению полным перебором на величину $\eta_5 \approx 28{,}4$ %. Учитывая, что теперь $\{A\} = \{1, 2, 3, 4, 5\}$, можно утверждать, что применительно к задаче (4) интеллект композитного варианта полного перебора оказался выше, чем интеллект всех вышеанализировавшихся алгоритмов.

6. Заключение

Предложенный выше подход позволяет в рамках сделанных допущений повысить интеллект алгоритмов поиска глобально оптимальных решений задач дискретной оптимизации с булевыми переменными. При этом легко прослеживается взаимосвязь между приведенными выше композитными алгоритмами: обладая аналогичными процедурами отсечения «плохих» планов, они разнятся лишь стратегиями ветвления, причем, как показано в разделе 3.2, предпочтительной является стратегия присущая алгоритму 3. Что касается алгоритма 5, то неоднократное повторение шагов 2–7, при котором разбиение на подмножества полных планов на шаге 2 охватывает лишь планы, соответствующие первой компоненте перестановки π, получаемой на текущей итерации, сближает такую модификацию этого алгоритма с алгоритмом 3. Если же повторяются шаги 2 – 5, причем шаг 2 заменяется следующим: 2. Все ранее полученные подмножества полных планов одновременно разбиваются на 2^h подмножеств, где h – число введенных в базис значений вектора булевых переменных Z (в рассмотренном в предыдущем параграфе примере $h = 2$), то такая модификация алгоритма 5 оказывается близка к алгоритму 4.

Увеличение числа тестов на отсечение подмножеств «плохих» планов в композитных алгоритмах по сравнению с соответствующими традиционными процедурами в соответствии с (1) *a priori* свидетельствует об их более высоком интеллекте. При этом следует учитывать, что допущение о линейной зависимости времени поиска решения от числа вершин построенного дерева ветвлений применительно к методам типа ветвей и границ, осуществляющим фронтальный спуск по дереву ветвлений $G(X, U)$, справедливо для сочетания задач сравнительно небольших размерностей с эффективным способом вычисления оценки: с ростом числа висячих вершин дерева ветвлений $G(X, U)$ основные затраты времени на каждой итерации смещаются с вычисления компонент векторов $V(x_j)$, $x_j \in X$, к сравнению их первых компонент. Таким образом, дальнейшее развитие предлагаемого подхода может быть связано с:

а) экспериментальным анализом границ его эффективности применительно к методам поиска глобально оптимальных решений задач дискретной оптимизации;

б) априорным и экспериментальным анализом его эффективности применительно к детерминированным и рандомизированным методам локальной оптимизации.

Литература

1. *Лорьер Ж. Л.* Системы искусственного интеллекта, М.: Мир, 1991. 568 с.

2. *Luger G. F.* Artificial Intelligence // Structures and Strategies for Complex Problem Solving, 4-th edition. Williams, 2005. 864 p. ISBN 5-8459-0437-4.

3. Handbook of Operations Research, V. 2 // Models and Applications. Edited by Joseph J. Moder and Salah E. Elmaghraby, North Carolina State University, 1978, 677 p.

4. *Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн, К.* Алгоритмы: построение и анализ, 2-е издание. М.: «Вильямс», 2005. С. 833–839.

5. *Land A. H., Doig A. G.* An automatic method of solving discrete programming problems, Econometrica. Vol. 28. No. 3. (Jul., 1960). P. 497–520.

6. *Беллман Р.* Динамическое программирование. М.: Изд-во иностранной литературы, 1960.

7. Chvátal V. et al. Selected combinatorial research problems // Technical Report STAN-CS-72-292. Computer Science Department, Stanford University, 1972.

8. *Корбут А. А., Финкельштейн Ю. Ю.* Дискретное программирование. М.: Наука, 1969. 368 с.

9. *Гроппен В.О.* Повышение эффективности методов типа ветвей и границ для комбинаторных задач с булевыми переменными // "Автоматика и телемеханика". 1978. № 5, С. 105–112.

10. *Гроппен В. О.* Эффективная организация комбинаторных алгоритмов на однородных вычислительных средствах. Труды 5-го Международного семинара «Прикладные аспекты теории автоматов», Варна, 1979, С. 358–364.

11. *Гроппен В. О.* Экстремальные задачи на взвешенных графах. Владикавказ, Изд. Фламинго, 2012, 90 с.

12. *Кербер М. Л.,* Полимерные композиционные материалы. Структура. Свойства. Технологии. СПб.: Профессия, 2008, 560 с.

13. *Перепелкин К. Е.* Армирующие волокна и волокнистые полимерные композиты, СПб.: Научные основы и технологии, 2009, 380 с.

14. *Мелешко А. И., Половников С. П.*Углерод, углеродные волокна, углеродные композиты, 2007, ISBN 5-88070-119-0. С. 192.

15. *Васильев В. В.* Механика конструкций из композиционных материалов. М.: Машиностроение, 1988, 272 с.

16. *Федунов Б. Е.* Методика экспресс-оценки реализуемости графа решений оператора антропоцентрического объекта на этапе разработки спецификаций алгоритмов бортового интеллекта // Известия РАН. Теория и системы управления (ТиСУ). М.: 2002. № 1.

17. *Разборов А. А.* О сложности вычислений // МЦНМО, Математическое просвещение. 1999. № 3. С. 127–141.

18. *Groppen V. O.* New Solution Principles of Multi-Criteria Problems Based on Comparison Standards
www.arxiv.org/ftp/math/papers/0501/0501357.pdf , 2004.

Сведения об авторе

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

ТЕХНОЛОГИИ ЭКСТРЕМАЛЬНОГО ПРОГРАММИРОВАНИЯ

УДК 681.343.001

Томаев М. Х., Асланов Г. А.,
Ванюшенкова Н. В.

ИСПОЛЬЗОВАНИЕ ОПТИМИЗАЦИОННЫХ МОДЕЛЕЙ «ЭКСТРЕМАЛЬНОГО ПРОГРАММИРОВАНИЯ» В ПРОЕКТИРОВАНИИ ПО

Предлагаются подходы оптимизации программных продуктов, относящиеся к классу «экстремального программирования», способы формулировок данного класса моделей на примерах задачи оптимального выбора класса памяти для данных в пользовательской программе ЭВМ, а также задачи выбора оптимальной стратегии макрозамен. Приводятся алгоритмы решения, а также особенности программной реализации. Использование полученных подходов позволит повысить адекватность прогнозов по достижению заданных показателей качества при использовании оптимизационных подходов «экстремального программирования». Разработанные в работе модели и методы могут быть использованы при создании автоматизированных средств проектирования ПО.

Ключевые слова: *программа, оптимизация, модель, качество, производительность*

1. Введение

Под методами «экстремального программирования» понимают нетрадиционные подходы к разработке ПО, в том числе методы оптимизации программ, основанные на улучшении одного из критериев программы за счет экстенсивного использования одного из ресурсов вычислительной системы, причем стоимость этого ресурса может быть достаточно высокой. Актуальность «экстремальных» методов программирования обуславливается двумя основными причинами:

1. Во-первых, развитие элементной базы вычислительных систем привело к значительному снижению стоимости различных компонент вычислительных систем. В результате цена единицы процессорного времени, а также стоимость хранения единицы информации (как в «быстрой» оперативной, так и в «медленной» внешней памяти) в течение нескольких лет значительно сократилась. В этих условиях использование экстремальных подходов довольно часто становится экономически оправданным.

2. Вторая причина – наличие классов задач, для которых заданные параметры качества должны быть достигнуты «любой» ценой. К этой категории относятся: программные системы военного назначения; АСУ технологических процессов; системы мониторинга состояния жизненно важных систем (различного рода контрольные и следящие системы).

В большинстве случаев целью оптимизационных преобразований является улучшение производительности программы, а наиболее популярным «вспомогательным» ресурсом – оперативная память.

Оптимизационные подходы условно можно разделить на две группы:

1. Методы оптимизации управляющих операторов и выражений, когда один оператор заменяется на группу эквивалентных по назначению инструкций, выполняющих ту же операцию значительно быстрее, но занимающих значительно большее место в оперативной памяти. Примером такого рода является оптимизация поиска информации – замена последовательных алгоритмов поиска информации в массиве, поиском информации в многомерных индексных массивах либо в хэш-таблицах. Расход памяти стремительно возрастает с ростом длины хэш-ключа и ужесточением требования к его уникальности, что, однако, компенсируется исключительно высокой эффективностью подхода. Другой пример – метод макрозамен, – когда код пользовательской функции вставляется во все места её вызова, прирост производительности достигается в этом случае за счет времени на передачу управления (инструкция Ассемблера CALL) и возврат из функции, а также на создание локального стека для переменных и массивов, объявленных внутри блока функции. Наличие огромного числа функций в любой сложной системе позволяет создавать эффективные оптимизационные инструменты автоматического преобразования кода на основе этого метода.

2. Методы, позволяющие изменить способ хранения переменных таким образом, чтобы минимизировать время обслуживания (которое

составляют время выделения и освобождения блока памяти, необходимого для хранения элемента данных). Архитектура современных ОС предоставляет программам прикладного уровня три способа хранения переменных в быстрой памяти:

1) Память статическая, выделяемая единым блоком при запуске приложения, соответственно алгоритм программы не несет никаких расходов по выделению либо освобождению памяти для статических данных. Эта область используется для размещения «глобальных» по времени жизни данных, т. е. существующих от момента загрузки процесса до момента его завершения;

2) Стековая память, используемая для размещения «временных» данных, т. е. переменных и массивов, объявленных в пределах любого из блоков программы. Начало блока сопровождается автоматическим выделением стековой памяти, необходимой для размещения локальных переменных. Завершение такого блока приводит к автоматическому освобождению памяти, выделенной для стековых переменных. Таким образом, для обслуживания каждой стековой переменной программа затрачивает время, необходимое на выполнение пары ассемблерных инструкций push/pop (вставка в стек выборка из стека);

3) Динамическая память – позволяет выделять для работы программы участки оперативной памяти произвольного размера. Однако высокая дефрагментация динамической памяти (в случае запуска большого числа приложений) может существенно снизить скорость поиска подходящего по размеру блока. Таким образом, накладные расходы прикладного алгоритма составляет время на обслуживание менеджером динамической памяти ОС всех данных программы переменной длины. Кроме того, следует учитывать, что по очевидным причинам вероятность успешного выделения динамического блока падает с ростом размера запрашиваемого участка, а также с ростом загруженности ОС. В общем случае накладные расходы на обслуживание динамической памяти значительно больше по сравнению со стековой.

Очевидно, что замена модели памяти для пользовательской переменной, заключающаяся в её перемещении из «медленной» памяти в более «быструю», позволит получить прирост производительности. Причем этот прирост будет более заметен на повторяющихся участках кода.

Далее в работе рассматриваются подходы к формулировке задач экстремального программирования на основе анализа структуры программы.

2. Формулировка задач экстремального программирования

Высокая требовательность к ресурсам оперативной памяти приводит часто к невозможности на практике достижения для оптимизируемых прикладных систем уровня качества «идеального» с точки зрения выбранного оптимизационного подхода. К примеру макрозамена всех без исключения функций в сложных системах, исходный код которых включает сотни тысяч и более строк, приведет к тому, что объектный код приложения увеличится многократно и вероятнее всего перестанет удовлетворять возможностям конфигурации клиентских вычислительных устройств (в частности доступному объему оперативной памяти).В этой связи актуальной является задача формулировки объективных критериев качества, позволяющих выделить на пользовательском прикладном алгоритме участки, оптимизация которых даст максимальный прирост производительности. В работе [1] дается обоснование подхода, основанного на особенностях человеческого мышления. В частности, программист – «Пессимист» будет стремиться оптимизировать участки программ, соответствующих верхней границе времени поиска решения, а «Оптимист» – те, что соответствуют нижней. Кроме того, классификация алгоритмов, предложенная в [1], позволяет выделить «ценность» тех или иных участков по их структуре – в частности, обосновывается предпочтительность в качестве первичных объектов оптимизация участков зацикливания (т. е. участков кода программы, входящих в состав тех или иных контуров) по сравнению с участками, не являющимися частью того или иного контура. Используя эти два положения, сформулируем задачи оптимизации оптимальной стратегии макрозамен и оптимальной стратегии выбора класса памяти для пользовательских переменных.

3. Задача выбора оптимальной стратегии макрозамен

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. Высо-

кая 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 (1), описывающую задачу поиска оптимальной стратегии макрозамен, минимизирующей верхнюю границу однократного зацикливания:

$$\begin{cases} \sum_{i=1}^{C(L_1,s)} T_i \cdot p_i \rightarrow \max; \\ T_i = \sum_{j \in X\left(L_1^i,s\right)} \left(z_j \cdot N_j \cdot t_j\right); \\ p_i = \begin{cases} D\left(L_1^i,s\right), \text{ пессимист} \\ \max_k D\left(L_1^k,s\right) - D\left(L_1^i,s\right)+1, \text{ оптимист} \end{cases} \\ \sum_{k=1}^{M} z_k \cdot \upsilon_k \cdot N_k < V_{max}; \\ \forall i=1..M;\ z=\overline{0,1}, \end{cases} \quad (1)$$

где $C(L_1,S)$ – число возможных путей из начального состояния в конечное L_1,s;

T_i – выигрыш во времени выполнения функций, реализующих *i*-й вариант пути L_1,s;

$X\left(L_1^i,s\right)$ – множество индексов функций, реализующих *i*-й вариант пути L_1,s;

p_i – вес, обозначающий «важность» *i*-го варианта пути.

Для стратегии «Пессимист» важность можно принять равной длине пути, т. к. наиболее важна верхняя граница времени счёта, а для стратегии «Оптимист» – наоборот;

υ_k – размер *k*-й функции;

N_j – количество вызовов *j*-й функции;

Z_j – булева переменная, равная 1, если k-ая функция заменяется макро вставкой, 0 – в противном случае;

M – количество функций в программе;

V_{max} – верхняя граница дополнительного объёма кода программы;

$D\left(L_{ij}\right)$ – длина пути (i, j);

s – вершина сток.

Для решения данной задачи можно применить различного рода комбинаторные процедуры. В частности, в ходе реализации практической части была разработана программа, осуществляющая поиск решения двумя алгоритмам: полным перебором и методом Монте-Карло.

4. Программная реализация метода макрозамен конечных алгоритмов

4.1. Алгоритм работы программы

Входными данными являются: исходный код программы на c++, верхняя граница памяти.

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) выделение всех нециклических путей на графе;

6) определение оптимальной стратегии макрозамен на множестве функций для каждого конкретного пути.

4.2. Интерфейс программы

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

Структурно программа выполнена в виде диалогового окна с элементами управления для загрузки исходного файла пользователя и запуска процесса оптимизации. При запуске программы появляется окно (рис.5).

Для того, чтобы открыть код программы, необходимо перетащить файл с исходным кодом в поле «Исходный код», либо воспользоваться пунктом меню «Файл».

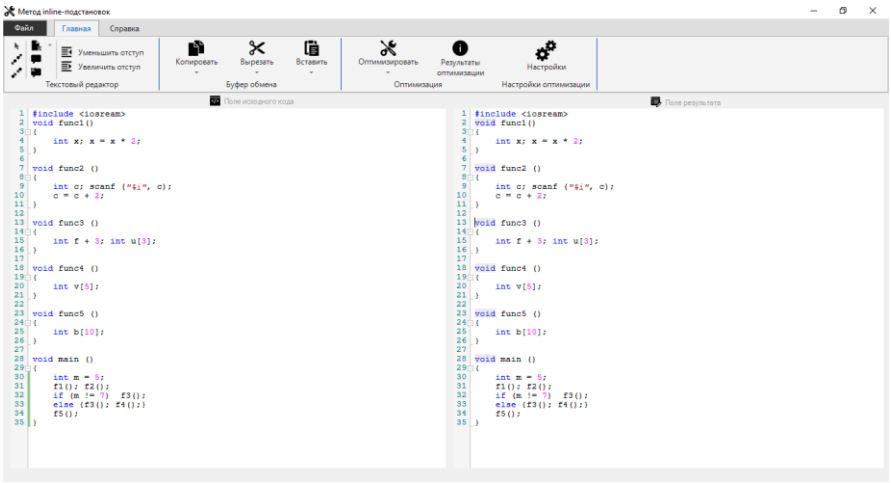


Рис. 5. Окно программы с загруженным исходным кодом пользователя

Настройки оптимизации задаются на отдельной форме, перейти на которую можно, нажав кнопку «Настройки» на «Панели управления».

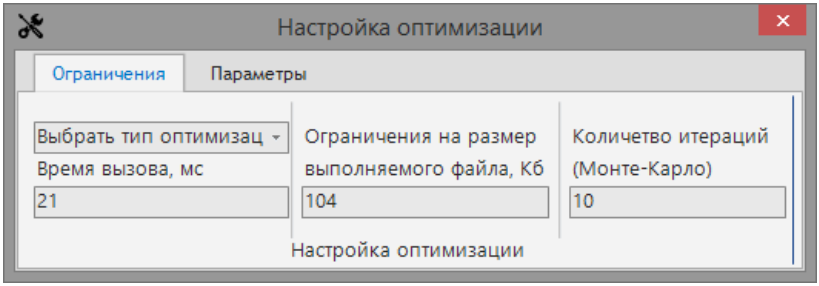


Рис. 6. Настройки оптимизации. Вкладка «Ограничения»

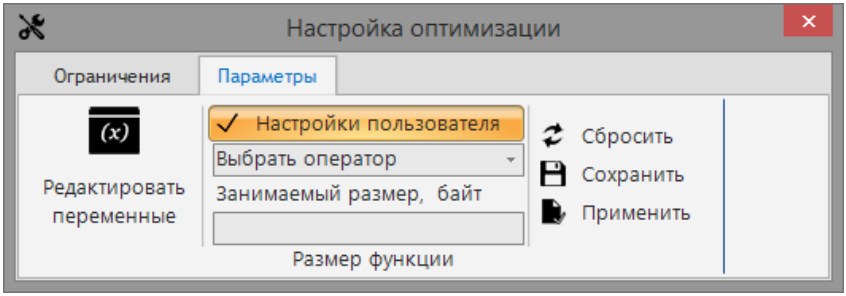


Рис. 7. Настройки оптимизации. Вкладка «Параметры»

Чтобы произвести оптимизацию исходного кода, нужно нажать на кнопку «Оптимизировать» на «Панели управления» и выбрать алгоритм оптимизации: «Полный перебор» или «Монте-Карло».

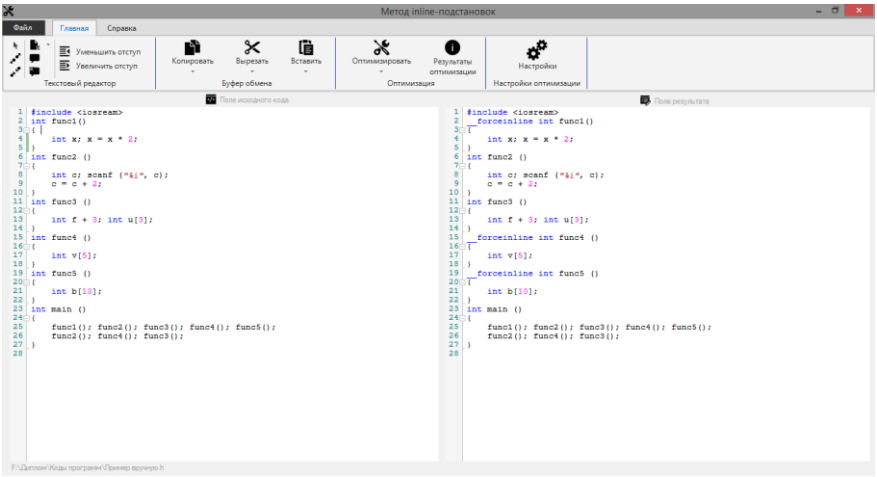


Рис. 8. Состояние формы, после оптимизации

Результат оптимизации можно посмотреть на отдельной форме, нажав на кнопку «Результаты оптимизации» на «Панели управления».

Результаты анализа отображаются на вкладках «Граф состояний» «Функции и веса», как показано на рис. 9.

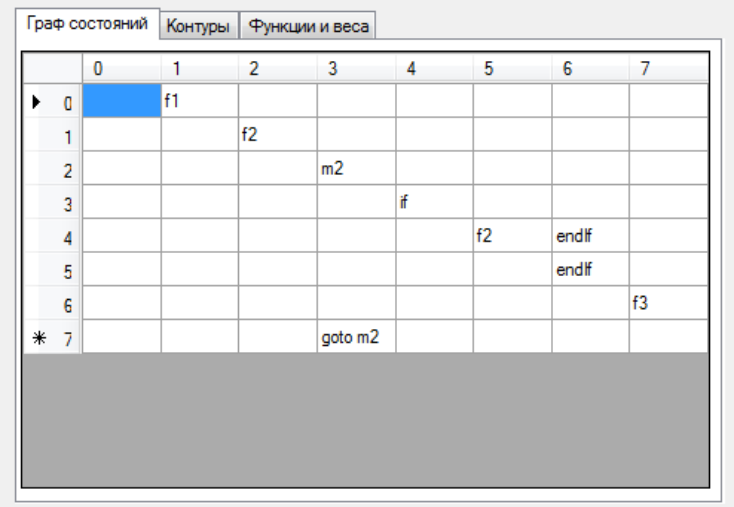


Рис. 9. Матрица смежности, изображенная по результатам анализа исходного кода пользовательского алгоритма

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

На вкладке «Функции и веса» (рис.10) 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).

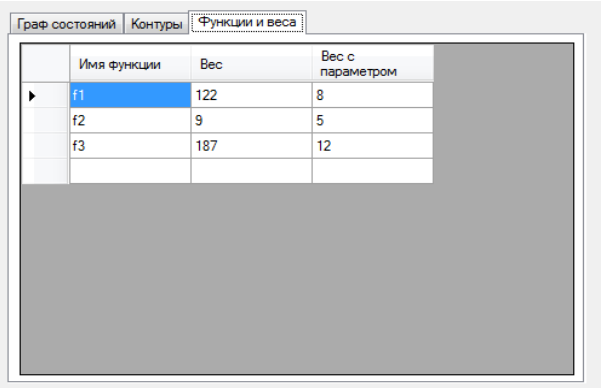


Рис. 10. Пример работы программы. Вкладка «Функции и веса»

Результат оптимизации можно посмотреть на отдельной форме, нажав на кнопку «Результаты оптимизации» на «Панели управления».

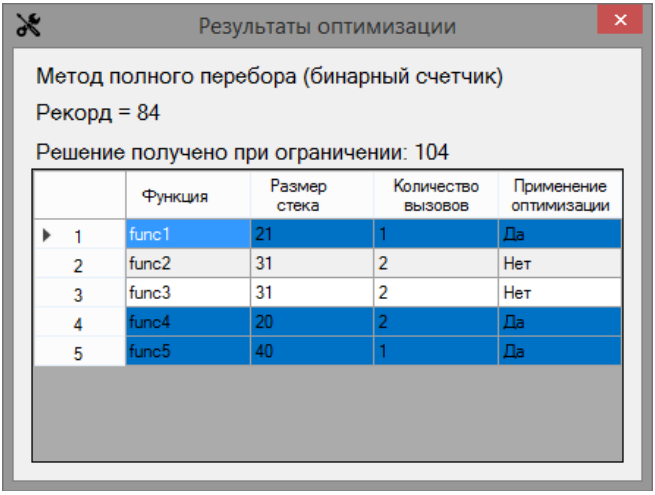


Рис. 11. Результат оптимизации

После окончания работы с программой результаты оптимизации можно сохранить в файл с расширениями.c, .cpp, .h, .txt, а также .rtf.

Полученный программный продукт позволяет выполнять макрозамены в соответствии с оптимальной стратегией, сформулированной в задаче (1), 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 включены в состав комплексного решения, осуществляющего оптимизацию пользовательского исходного кода.

5. Задача выбора оптимального класса памяти пользовательских переменных

Основная идея формулируемого подхода – замена стековых переменных (на обслуживание и размещение которых в стеке затрачиваются ресурсы процессорного времени) на статические, глобальные, по времени жизни. Неконтролируемая замена такого рода всех локальных переменных приведет к многократному росту исполняемого (.exe, .dll) кода программы, поэтому для создания модели оптимизации используем подход, аналогичный предложенному в главе 2, для метода макрозамен и сформулируем задачу выбора оптимального класса памяти для локальных (стековых) переменных множества функций, составляющих максимальный контур в виде модели (2):

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

$$\begin{cases} \sum_{i=1}^{C(L_1,s)} T_i \cdot p_i \to \max; \\ T_i = \sum_{j \in X(L_1^i,s)} \left(z_j \cdot N_j \cdot \left(\sum_k^{Q_j} \frac{\upsilon_{jk}}{S} \right) \right); \\ p_i = \begin{cases} D(L_1^i,s), \text{ пессимист} \\ \max_k D(L_1^k,s) - D(L_1^i,s) + 1, \text{ оптимист} \end{cases} \\ \sum_{k=1}^{M} z_k \cdot \upsilon_k \cdot N_k < V_{max}; \\ \forall i = 1..M; z = \overline{0,1} \end{cases} \quad (2)$$

где $C(L_1, S)$ – число возможных путей из начального состояния в конечное L_1,s;

T_i – выигрыш во времени выполнения функций, реализующих *i*-й вариант пути L_1,s;

$X(L_1^i,s)$ – множество индексов функций, реализующих *i*-й вариант пути L_1,s;

p_i – вес, обозначающий «важность» *i*-го варианта пути. Для стратегии «Пессимист» важность можно принять равной длине пути, т. к. наиболее важна верхняя граница времени счёта, а для стратегии «Оптимист» – наоборот;

$\boldsymbol{v_{ik}}$ – размер *i*-й переменной *k*-й функции;

$\boldsymbol{N_k}$ – количество вызовов *k*-й функции;

$\boldsymbol{z_j}$ – булева переменная, равная 1, если *j*-я переменная заменяется на статическую, 0 – в противном случае;

M – количество функций;

V_{max} – верхняя граница дополнительного объёма кода программы;

$D(L_{ij})$ – длина пути (*i*, *j*);

s – вершина сток;

S – скорость выделения стековой памяти.

Как и в случае (1) данная задача (2) может быть решена с помощью комбинаторных подходов.

6. Постановка и результаты экспериментов

Значение прироста производительности в результате макрозамен включает два слагаемых:

1) время на передачу управления в функцию и возврат из неё;

2) время на выделение стековой памяти функции, необходимой для размещения всех локальных переменных и массивов, – и может быть представлено в виде выражения (3):

$$T = t^{call} + t^{stack}, \tag{3}$$

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} – 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} и их значимость в выражении (1) 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 «In linefunction expansion» был установлен в «Only __in line (/Ob1)» – данный флаг запрещает компилятору игнорировать инструкцию in line при построении исполняемого кода (рис. 1). Замеры проводились в режиме «Release» для исключения диагностического кода.

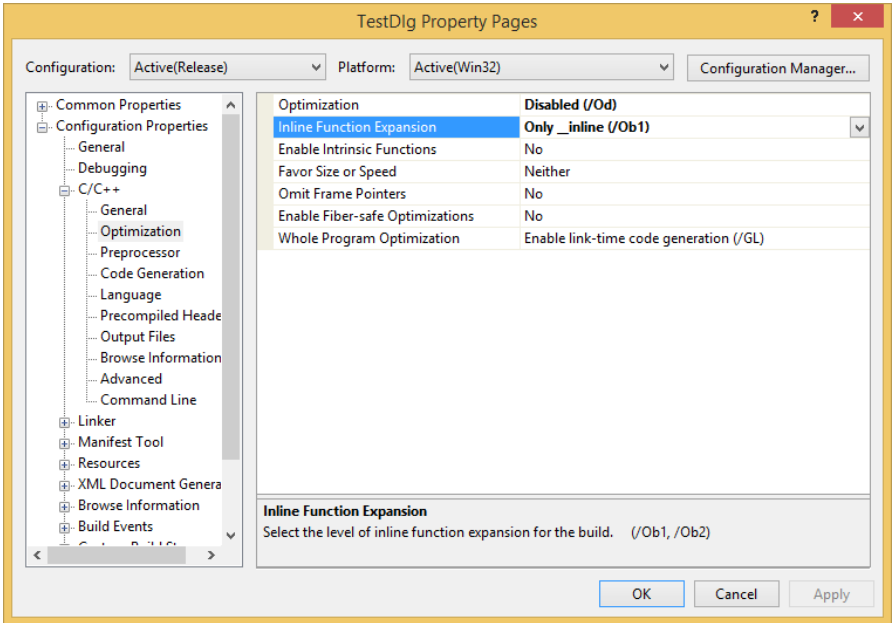


Рис. 12. Настройки тестового проекта

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 1000000 до 20000000 с шагом 1000000. 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)

```
voidf()
{

}
```

Листинг 2. Объявление функции f().

Во втором случае с использованием инструкции _force in line (Листинг 3):

```
__forceinlinevoidf()
{

}
```

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 20000000 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 два 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):

```
voidf()
{
    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 неё незначительны.

Таблица 1

Результаты 2-го теста оценки t^{call}

Число итераций	Время в миллисекундах
1000000	15
2000000	47
3000000	63
4000000	78
5000000	109
6000000	125
7000000	141
8000000	172
9000000	172
10000000	203

Для оценки зависимости времени выделения стековой памяти от размера локальных данных был проведен следующий эксперимент: число итераций было зафиксировано значением 20 000 000. Далее было проведено 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 __force in line. Результаты обоих экспериментов приведены в табл. 2.

Таблица 2

Результаты замеров времени выделения локальной памяти

Размер данных	Время работы обычной функции	Время работы __force in line 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 __force in line: очевидно, что объявление данных компилятор поместил перед циклом. На графике, изображенном на рис. 4, хорошо виден линейный характер зависимости времени выделения локального стека функции от его размера: поверх кривой экспериментальных данных, взятых из первой и второй колонок табл. 2, проведена прямая (пунктиром), коэффициенты, которой получены аппроксимацией данных методом МНК.

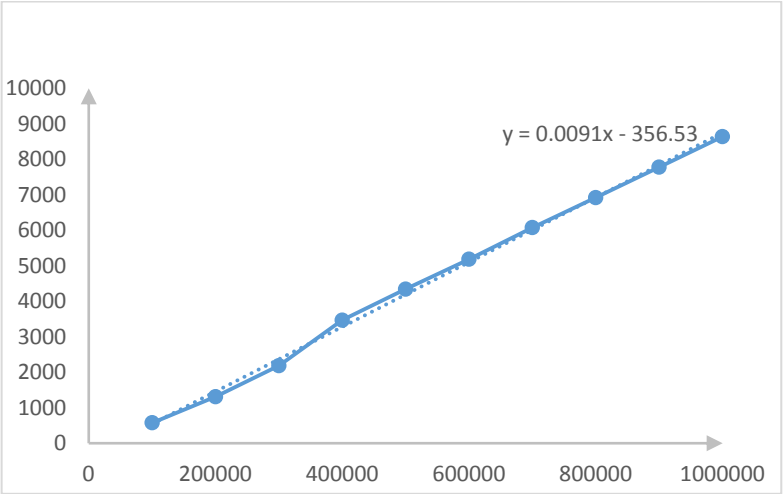


Рис. 13. Кривая экспериментальных данных и прямая, описывающая линейную зависимость времени выделения стековой памяти от её размера

На графике, изображенном на рис. 13, 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 выигрыш от макрозамены t^{call}, ввиду его фактической несущественности. 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.

7. Заключение

Сложность формулировки объективных критериев качества является одним из основных препятствий на пути широкого внедрения систем глобальной оптимизации. Работа представляет собой еще один этап исследований в данном направлении. Результаты работы могут быть использованы как в технологии разработки ПО, так и для дальнейших исследований в рамках проблематики создания теории и методов оптимизации качественных программных продуктов.

Литература

1. *Гроппен В. О.* Принципы оптимизации программного обеспечения ЭВМ. Ростов –на-Дону. Изд. Ростовского университета, 1993.

2. *Гроппен В. О., Томаев М. Х.* Модели, алгоритмы и средства программной поддержки проектирования оптимальных программных продуктов // Автоматика и телемеханика, 2000.

3. *Томаев М. Х.* Выбор оптимальной стратегии макрозамен // Труды молодых ученых ВНЦ РАН. 2005. № 4.

4. *Томаев М. Х.* Технологии глобальной оптимизации пользовательских программных кодов // Автоматизация и управление в технических системах. 2015. № 3. С. 16–30. DOI: 10.12731/2306-1561-2015-3-2. URL: auts.esrae.ru/15-277.

5. *Томаев М. Х.* Выбор оптимальной стратегии макрозамен в циклящихся программных алгоритмах, реализованных на языке "C++". Пром-Инжиниринг // Труды II международной научно-технической конференции, 19–20 мая 2016. Челябинск-Новочеркасск-Волгоград-Астана, 2016. ISBN 978-5-696-04834-5. URL: http://www.icie-rus.org/issues/ICIE-2016RU.pdf

Сведения об авторах

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

Асланов Г. А.,
студент магистратуры СКГМИ (ГТУ)

Ванюшенкова Н.,
студентка магистратуры СКГМИ (ГТУ)

УДК 004.021

Томаев М. Х., Кисиев В. П.

МНОГОКРИТЕРИАЛЬНЫЙ ПОДХОД К ОПТИМАЛЬНОЙ ДЕКОМПОЗИЦИИ ПОЛЬЗОВАТЕЛЬСКОГО ПРОГРАММНОГО КОДА

В работе формулируются принципы эффективной декомпозиции программных кодов, функционирующих в условиях недостатка доступного объема оперативной памяти. Предлагается многокритериальный подход к выбору оптимальной стратегии декомпозиции пользовательской системы на подпрограммы и методы решения задач. Приводятся особенности программной реализации экспериментальной разработки, выполняющей автоматизированный поиск оптимальной декомпозиции на базе разработанной технологии.

Модели и методы, полученные в результате исследований, могут быть использованы в составе автоматизированных средств проектирования эффективных программных продуктов.

Ключевые слова: *программа, оптимизация, декомпозиция, критерий, многокритериальная, модуль, подпрограмма, память, модель, производительность.*

1. Введение

Причина сохраняющейся актуальности оптимизационных подходов заключается в том то, что потребности в автоматизации различных областей человеческой деятельности растут со скоростью, опережающей рост возможностей элементной базы аппаратных средств вычислительных систем. Технологии автоматизации различных этапов проектирования ПО значительно повышают производительность разработки, в результате сложность современных прикладных систем стремительно возрастает. Часто размер сложных систем превышает доступный объем оперативной памяти ЭВМ целевой клиентской рабочей станции. В таком случае возникает естественная потребность в эффективной декомпозиции пользовательской системы на подпрограммы, минимизирующие время работы с «медленной» постоянной памятью ЭВМ («жестким» диском, SSD либо другими типами носителей). Формулировка объективных критериев качества (производительности) требует учета структуры программы.

2. Многокритериальный подход к оптимизации пользовательских программных продуктов

В работе [1]предлагается классификация, в соответствии с которой все алгоритмы можно отнести либо к циклящимся (не имеющим вариантов завершения), либо к конечным программным алгоритмам (не образующих контура). Очевидно, что для первых объективным критерием производительности является время однократного зацикливания, а для вторых – время поиска решения (время завершения).

Сформулируем критерий минимизации времени однократного зацикливания в отдельно взятом контуре a_k, приняв следующие обозначения:

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

t_j – время загрузки j-го варианта модуля, $t_j > 0$;

a_k – k-й контур, принадлежащий множеству всех контуров $A(G)$ 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.

Очевидно, что стратегия размещения функций в модулях должна быть выбрана таким образом, чтобы каждая функция была помещена только в одну из возможных подпрограмм, т. е.

$$\forall i, \sum_j z_{ij} = 1. \qquad (1)$$

Принимая во внимание (1) окончательно критерий минимизации времени однократного зацикливания, можно представить в виде выражения (2):

$$F_k = \sum_{j \in H(a_k)} t_j sign \sum_i^n z_{ij} \to \min. \qquad (2)$$

С учетом ограничения (1) формулировка (2) имеет идеальное решение, равное времени загрузки варианта подпрограммы, включаю-

щей сразу все функции. Если принять $\forall j : t_j = 1$, то идеальное значение $F_k = 1$. Однако для больших систем размещение всего кода в одном модуле часто невозможно. Размер каждой подпрограммы не должен превышать верхней границы доступного объема оперативной памяти. В том случае, когда модули выполняются последовательно (т. е. отработав, подпрограмма передает управление следующей, полностью освобождая для нее занятые ресурсы оперативной памяти), соответствующее ограничение может быть сформулировано следующим образом:

$$\max_j (v_j signum(\sum_i z_{ij})) \le V, \qquad (3)$$

где v_j – размер j-ого варианта подпрограммы;

V – верхняя граница доступного объёма оперативной памяти.

Смысл ограничения (3) в том, что для вариантов модулей, включенных в решение (т. е. таких в которых размещена как минимум одна функция – $signum(\sum_i z_{ij})$), размер максимального из них должен соответствовать верхней границе *V*.

Формулировки (2) достаточно для пользовательских алгоритмов, содержащих только простые контуры, т. е. таких программ, в которых нет функций, входящих в состав более чем одного контура. Если контуры не пересекаются, то всегда существует непротиворечивое решение для каждого критерия-контура, не нарушающее оптимальность решения, найденного в результате декомпозиции другого контура.

Реальные масштабы систем предполагают наличие большого числа сложных контуров с пересекающимся множеством функций. Формулировка (2) в этом случае является неоднозначной, т. к. оптимальная декомпозиция одного контура возможно сделает невозможным оптимально разбить систему на участке другого контура. Одним из способов формулировки в этом случае является использование лексико-графического упорядочения критериев. Т. е. последовательное рассмотрение контуров в порядке их важности. Для выбора приоритета критериев можно воспользоваться подходом, предложенным в [1], согласно которому существует две противоположные стратегии оптимизации, в основе которых лежат особенности человеческого мышления. Первая стратегия заключается в том, что критерии упорядочиваются в порядке уменьшения длины контуров. Такой «пессимистиче-

ский» подход предполагает в первую очередь решение задачи оптимальной декомпозиции на участках, соответствующих верхней границе однократного зацикливания, оптимальной декомпозиции которой соответствует следующее выражение:

$$f_1 = \max_{a_k^1 \in A(G)} \sum_{j \in H(a_k^1)} t_j sign \sum_i^n z_{ij} \to \min. \qquad (4)$$

Второй критерий и другие, менее важные, характеризуют качество декомпозиции в других контурах, следующих в порядке убывания их длины:

$$f_R = \max_{a_k^R \in (A(G) \setminus \bigcup_{q=1}^{R-1} a_{\max}^q)} \sum_{j \in H(a_k^R)} t_j sign \sum_i^n z_{ij} \to \min, \qquad (5)$$

где R – номер позиции критерия в упорядочении;

$a_{\max}^{R-1}$ – контур, оценка которого характеризует «предыдущий» критерий с номером, равным $(R-1)$. Контур $a_{\max}^R$ – это максимальный из рассматриваемого текущим критерием (в позиции R) подмножества контуров, которое равно исходному множеству за исключением контуров, выбранных более старшими критериями;

$$\sum_{i \in H^f(a_{\max}^R)} t_i^f = \begin{cases} \max\limits_{a_k \in (A(G) \setminus (\bigcup_{q=1}^{R-1} a_{\max}^q))} \sum\limits_{i \in H^f(a_k)} t_i^f, R > 1; \\ \max\limits_{a_k \in A(G)} \sum\limits_{i \in H^f(a_k)} t_i^f, R = 1. \end{cases} \qquad (6)$$

Здесь t_i^f – время работы i-й функции;

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

«Оптимистический» подход будет соответствовать обратному упорядочению критериев – в порядке возрастания длин контуров, а советующий критерий изменится следующим образом:

$$f_R = \min_{a_k^R \in (A(G) \setminus \bigcup_{q=1}^{R-1} a_{\min}^q)} \sum_{j \in H(a_k^R)} t_j sign \sum_i^n z_{ij} \to \min. \qquad (7)$$

Часто в программных системах существуют как варианты завершения, так и участки, на которых код зацикливается, т. е. помимо контуров появляется еще один тип критериев, описывающих качество декомпозиции на одном из вариантов линейного участка переходов состояний программы, ведущих к её завершению. Если вероятности переходов заранее неизвестны, то очевидна предпочтительность критериев, рассматривающих варианты зацикливания по сравнению с вариантами завершения, т. к. время зацикливания может быть равно бесконечности. Однако разработчик может изменить приоритет критериев, если ему известны среднее время работы в контурах (либо вероятности зацикливания) и среднее время поиска решения. В этом случае можно сформулировать оптимизационную многокритериальную задачу, включающую только два критерия. В соответствии с одним из них осуществляется оптимальная декомпозиция верхней границы однократного зацикливания, а в соответствии с другим – верхняя граница времени поиска решения.

Если в качестве наиболее важного выбран критерий минимизации верхней границы однократного зацикливания, то первый критерий будет равен сформулированному выше выражению (4). А второй критерий помогает минимизировать верхнюю границу поиска решения, причем таким образом, чтобы стратегии размещения функций в подпрограммах, образующих циклы не нарушались, т. е.

$$\forall j \in H(a_{\max}), \forall i = \overline{1,n} : z_{ij}^{linear} = z_{ij}, \qquad (8)$$

где $H(a_{\max})$ – множество индексов функций, принадлежащих максимальному контуру;

z_{ij} – стратегия размещения функций в подпрограммах, найденная в соответствии с критерием (4);

z_{ij}^{linear} – стратегия размещения функций в подпрограммах, реализующих верхнюю границу времени завершения.

С учетом ограничение (8) процедуру оптимальной декомпозиции в соответствии со вторым критерием можно представить в виде отдельной задачи (9):

$$\begin{cases} f_2 = \sum\limits_{j \in \{L\max(0,q^{\max})\}} t_j sign \sum\limits_i^n z_{ij}^{linear} \to \min; \\ \forall i, \sum\limits_j z_{ij}^{linear} = 1; \\ \max\limits_j (v_j sign \sum\limits_i z_{ij}^{linear}) \le V; \\ \forall j \in H(a_{\max}), \forall i = 1,n : z_{ij}^{linear} = z_{ij}, \\ z_{ij}^{linear} = 1,0; i = 1,2,..,n; \; j = 1,2,... \end{cases} \quad (9)$$

где $L\max(0,q^{\max})$ – на найденный максимальный путь, соответствующий верхней границе времени завершения программы:

$$\sum_{i \in H^f(L\max(0,q^{\max}))} t_i^f = \max_{x_q \subset X^T} \max_j \sum_{i \in H^f(L^j(0,q))} t_i^f \quad (10)$$

Для решения задач (4), (9) можно воспользоваться различного рода комбинаторными подходами. При небольших размерностях – прямым перебором.

3. Программная реализация многокритериальной оптимизации

Для решения задачи оптимальной декомпозиции 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 просмотра списка функций, последовательности вызовов функций и оптимальной декомпозиции, предложенной программой.

После того, как подгрузился анализируемый код (кнопка «Открыть»), пользователь может проанализировать его для того, чтобы разбить на функции и подготовить к дальнейшему анализу. После этого станет доступна кнопка «Анализ декомпозиций», которая производит оптимизацию и разбиение пользовательских функций в отдельные динамические библиотеки (DLL). Перед тем как нажать на него, необходимо задать верхнюю границу оперативной памяти, выделенную для оптимизации, (V) и выбрать приоритет оптимизации («Контура»,

«Пути»). Для оптимизации с приоритетом «Контура», программа находит самый длинный цикл в программе и записывает в отдельный DLL (только в том случае, если позволяет верхняя граница оперативной памяти), а приоритет «Пути» находит самый длинный путь.

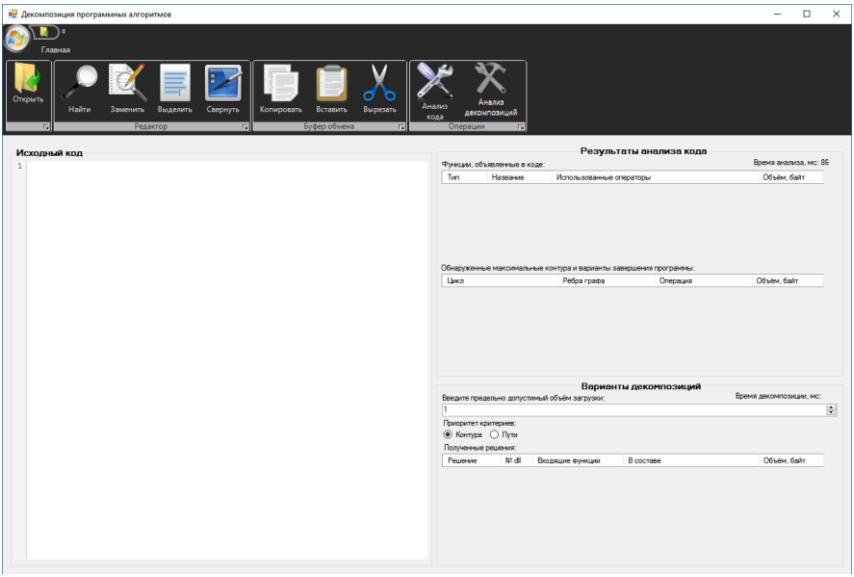


Рис. 1. Интерфейс программы

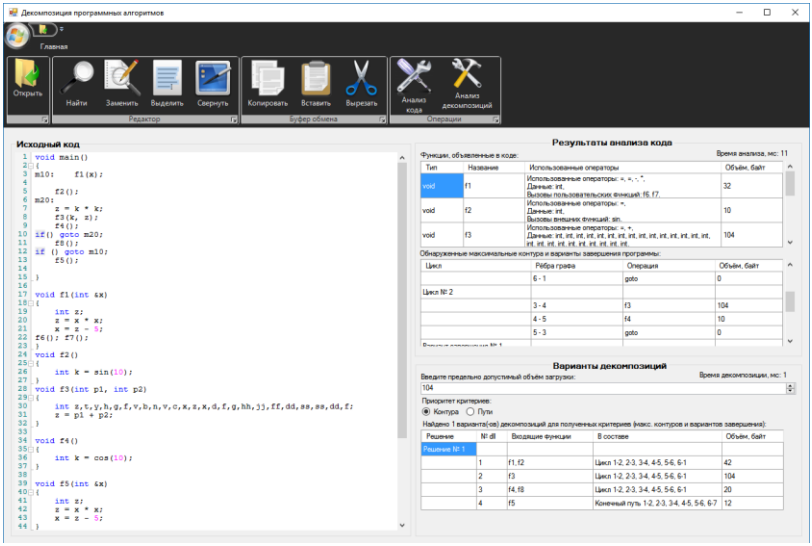


Рис. 2. Список функций, с указанием объема кода, а также входящих в них операторов

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

```
void main()
{
m10: f1(x);

 f2();
m20:
 z = k * k;
 f3(k, z);
 f4();
if() goto m20;
 f8();
if () goto m10;
 f5();

}

void f1(int &x)
{
 int z;
 z = x * x;
 x = z – 5;
f6(); f7();
}
void f2()
{
 int k = sin(10);
}
void f3(int p1, int p2)
{
 int z,t,y,h,g,f,v,b,n,v,c,x,z,x,d,f,g,hh,jj,ff,dd,ss,ss,dd,f;
 z = p1 + p2;
}

void f4()
{
 int k = cos(10);
}

void f5(int &x)
{
 int z;
 z = x * x;
 x = z – 5;
```

```
}

void f6()
{
 int k = cos(110);
}
void f7()
{
 int k = cos(10);
}
void f8()
{
 int k = cos(100);
}
```

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

На этапе анализа кода программой также составляется алгоритм вызовов функций, описывающей граф состояний программы, где дуги представляют собой функции, переводящие программу из одного состояния в другое. Для упрощения анализа кода операторы «goto» включаются в матрицу переходов, однако, в отличие от функций, имеют нулевой вес (дуги 6–1 и 5–3 в данном примере). Была выбрана оптимизация с приоритетом «Контура» (рис. 3.).

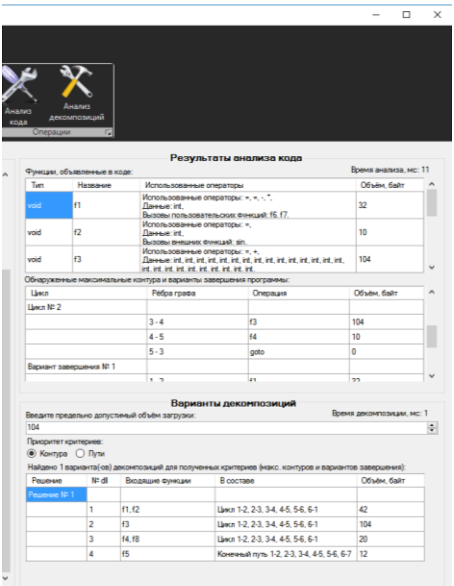


Рис. 3. Решение задачи с приоритетом оптимизации «Контура»

Проанализировав тот же код, только с приоритетом «Пути», получим следующий результат (рис 4):

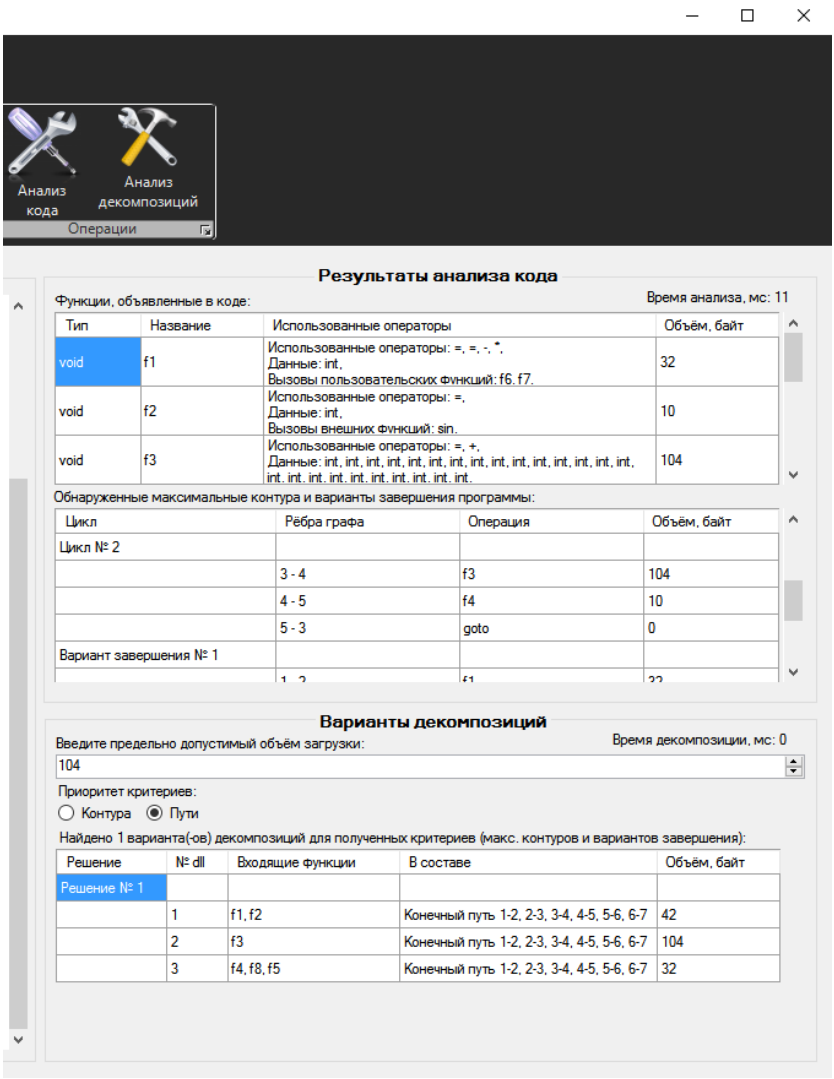


Рис. 4. Решение задачи с приоритетом оптимизации «Пути»

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

4. Заключение

Ценность полученных в результате исследований результатов в том, что получен специализированный программный продукт, осуществляющий многокритериальную оптимальную декомпозицию наиболее распространенного типа пользовательских алгоритмов – склонных к зацикливанию программ. Данный продукт будет востребован при разработке прикладного ПО различного назначение. А разработанный математический аппарат послужит основой дальнейших исследований по теме оптимизации пользовательских программных алгоритмов.

Литература

1. *Гроппен В. О.* Принципы оптимизации программного обеспечения ЭВМ. Ростов-на-Дону: Изд. Ростовского университета, 1993.

2. *Гроппен В. О., Томаев М. Х.* Модели, алгоритмы и средства программной поддержки проектирования оптимальных программных продуктов // Автоматика и телемеханика, 2000.

3. *Томаев М. Х.* Выбор оптимальной стратегии макрозамен // Труды молодых ученых ВНЦ РАН. 2005. № 4.

4. *Томаев М. Х.* Технологии глобальной оптимизации пользовательских программных кодов // Автоматизация и управление в технических системах. 2015. № 3. С. 16–30. DOI: 10.12731/2306-1561-2015-3-2. URL: auts.esrae.ru/15-277.

5. *Томаев М. Х.* Выбор оптимальной стратегии макрозамен в циклящихся программных алгоритмах, реализованных на языке “C++”. Пром-Инжиниринг. Труды II международной научно-технической конференции 19–20 мая 2016. Челябинск -Новочеркасск-Волгоград -Астана, 2016. ISBN 978-5-696-04834-5. URL: http://www.icie-rus.org/issues/ICIE-2016RU.pdf

Сведения об авторах

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

Кисиев В. П.
студент магистратуры СКГМИ (ГТУ)

ОБРАБОТКА ИЗОБРАЖЕНИЙ

УДК 004.42

Соколова Е. А.,
Бережная Е. Т.

РАЗРАБОТКА ФОРМАТА ДЛЯ ХРАНЕНИЯ 3D-МОДЕЛЕЙ С ИСПОЛЬЗОВАНИЕМ СЖАТИЯ БЕЗ ПОТЕРЬ

В работе рассмотрены наиболее часто используемые форматы для 3D-моделирования. Проанализированы их свойства, достоинства и недостатки. Предлагается новый формат для хранения 3D-моделей – .DBE, который позволит уменьшить размер модели без потери качества.

Ключевые слова: *3D-моделирование, форматы для хранения 3Dизображений, сжатие без потерь, оптимизация.*

1. Введение

Одна из основных задач 3D-форматов – это хранение различных моделей трехмерных объектов (3D-объекты), которые успешно используются в различных областях. Важную роль в сфере 3D-моделирования, для создания и обучения интеллектуальных систем, играют типы расширений трехмерных математических моделей. При создании новых виртуальных тренажеров и обучающих систем, а также виртуальной реальности для проведения научных исследований и трехмерной визуализации большинство специалистов сталкиваются с проблемой выбора правильного формата для хранения 3D-моделей и объектов.

Существует пять основных этапов для формирования информации: сбор, передача, накопление, обработка и представление информации. Именно на этапе представления информации используются технологии трехмерного моделирования. При использовании данных технологий, возникает вопрос о правильном хранении трехмерных объектов [1].

Если говорить о необходимости использования трехмерных моделей для дальнейшего развития интеллектуальных систем, то это связано с ее способностью намного упростить хранение данных. Например, при решении задач распознавания образов необходимо обраба-

тывать фотографии одного и того же объекта, полученные с разных ракурсов и в разных условиях. Хранить все возможные двухмерные фотографии подобных объектов физически невозможно. Следовательно, остается только хранить в памяти компьютера трехмерные модели объектов и сравнивать их с полученными двухмерными фотографиями и другими изображениями.

В подобных задачах важную роль играет выбор формата хранения трехмерных объектов. Следовательно, появляется необходимость исследования существующих 3D-форматов, с целью выявления наиболее эффективного типа файлов для хранения 3D-объектов [2].

Существует множество форматов для хранения 3D-моделей и объектов, многие из них давно устарели, но продолжают быть актуальными.

2. Сравнение 3D-форматов

Таблица 1

Тип данных	Преимущества	Недостатки
1	2	3
.3DS	Открытый формат, возможность чтения другими программами	Ограниченное количество полигонов, устаревшая технология по сравнению с .max
.MAX	Хорошая степень сжатия	Закрытый формат, большой объем файла, открывается только с помощью программы Autodesk 3ds Max
.OBJ	Открытый формат, маленький объем файла, бинарный и ASCII, самый распространенный среди других форматов, легко воспринимается пользователями без изучения дополнительных языков программирования	Формат не хранит иерархию и связи объектов сцены, не поддерживает анимацию
.FBX	Поддерживает анимацию, настройку освещения, расположение камер, поддерживается и обновляется компанией Autodesk, обладает бесплатной и открытой SDK, поддерживает сценическую графику, формат ASCII имеет древовидную структуру с чёткими обозначениями идентификаторов	Проприетарное программное обеспечение, закрытый исходный код

Окончание табл. 1

1	2	3
.WRL	Открытый формат, используется в качестве файлового формата для обмена 3D-моделями в САПР, используется в интернет-браузерах	Устаревшая технология по сравнению с .x3d
.STL	Кроссплатформенность, открытость исходного кода	Невысокая точность геометрии, большой объем файла для сложных моделей

Проведенное исследование выявило то, что каждый из 3D-форматов имеет свои достоинства и недостатки, которые необходимо учитывать при решении задач различной сложности [3].

В решении вопросов хранения трёхмерных математических объектов, связанных с интеллектуальными системами и системами виртуальной реальности, которые широко применяются в различных сферах деятельности человека, наиболее эффективным форматом является OBJ. Применение формата OBJ позволит минимизировать затраты на распознавание и хранение 3D-объектов [4].

Однако возможно данный формат улучшить, а именно, создать новый (.DBE), и, взяв за основу формат .OBJ, создать совершенно новую структуру файла, а также оптимизировать объем файла, что облегчит хранение 3D-модели.

В формате .OBJ каждый символ кодируется 1 байтом и при кодировании координаты из 8 цифр будет затрачено 8 байт памяти. В новом же формате после конвертации создается пакет из пяти файлов (файл координат вершин, файл координат нормали, файл координат текстур, параметры одной поверхности объекта и solution), каждый из которых содержит числа типа float. Соответственно, теперь эта же координата в файле будет занимать 4 байта, что существенно уменьшает объем файла [5].

Таблица 2

Тип данных	Объем файла модели		Объем файла после сжатия модели		Степень сжатия модели в %		Хранение текстур в файле
	низк., Кб	высок., Кб	низк., Кб	высок., Кб	низк., Кб	высок., Кб	
1	2	3	4	5	6	7	8
.3DS	9,78	621	3,28	100	7,95	7,76	+
.MAX	176	176	13,5	14	7,67	7,95	+

Окончание табл. 2

1	2	3	4	5	6	7	8
.OBJ	15,04	1638	4,7	330	20,15	31,25	+
.FBX	21,2	671	10,6	402	59,91	17,95	+
.WRL	8,6	958	2,3	176	18,37	26,79	+
.STL	12,8	1251	4,4	445	35,57	34,38	–
.DBE	15,04	1638	2,7	290	17,7	15,04	+

Как видно из табл. 2, формат .DBE показывает лучший результат в сравнении с форматом .OBJ.

3. Постановка задачи. Обозначения и определения

Любую 3D модель ($O\left(F_l F_p F_v F_n F_t\right) - 1$объект) можно представить, как совокупность единичных объектов различных по своей форме и структуре. Это нужно для того, чтобы оптимизировать размер 3D модели.

$$\begin{cases} F_o = \sum_{i=1}^{k} \left(F_{p_i} + F_{v_i} + F_{n_i} + F_{t_i}\right) * M \to min \\ i = \overline{1,k} \\ \forall\, F_o : k \geq 1, \text{целое};\ M > 0 \\ \forall F_{n_{i+1}} : F_{n_i} + \left(F_{n_i} - 1\right)(k-1) \\ \forall F_{p_{i+1}} : F_{p_i} + \left(F_{p_i} - 1\right)(k-1) \\ \forall F_{v_{i+1}} : F_{v_i} + \left(F_{v_i} - p(i-1)\right)(k-1) \\ \forall F_{t_{i+1}} : F_{t_i} + \left(F_{t_i} - p\right)(k-1), \end{cases} \quad (1)$$

где $F_{p_i}, F_{v_i}, F_{n_i}, F_{t_i}$ – объемы полигонов, вершины, нормали, координаты текстур конкретного единичного объекта (куб, трапеция, тетраэдр и т. п.);

O – объект, геометрия которого описана в четырех типизированных файлах $\left(F_p F_v F_n F_t\right)$;

F_0 – верхняя граница суммарного объема пяти файлов $\left(F_l F_p F_v F_n F_t\right)$;

$k -$ количество единичных объектов, из которых состоит объект;

F_l -файл-ссылка формата .DBE, который содержит в себе полный путь до остальных четырех файлов: файл полигонов (F_p), файл вершин (F_v), файл нормалей (F_n), файл текстурных координат (F_t);

$M-$ память в байтах отводящаяся типу данных;

p –количество вершин полигона единичного объекта.

Рассмотрим пример 3D модели (О) кубика Рубика как совокупность единичных объектов (кубов).

Если заранее просчитать выделенную или какую-либо другую фигуру, то впоследствии потребуется меньшее количество координат для ее представления, и это поможет оптимизировать размер файла.

*Рис. 1.*Объект (О) Кубик Рубика

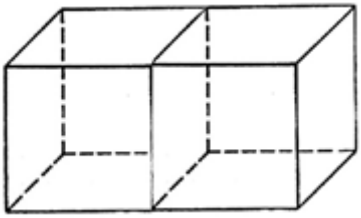


Рис. 2. Два единичных объекта – куба. $F_{p_i} = 6;\ F_{v_i} = 8;\ F_{n_i} = 6;\ F_{t_i} = 24.$

$$\begin{cases} F_o = (6+8+6+24) * k * M \ \rightarrow \min \\ \forall\, F_o : k \geq 1, \text{целое};\ M > 0 \\ \forall F_{n_{1+1}} = 6 + (6-1)(2-1) = 11 \\ \forall F_{p_{1+1}} = 6 + (6-1)(2-1) = 11 \\ \forall F_{t_{1+1}} = 24 + (24-4)(2-1) = 44 \\ \forall F_{v_{1+1}} = 8 + (8 - 4 * (2-1))(2-1) = 12 \\ \text{и т. д.} \end{cases} \quad (2)$$

где $O-$ объект, геометрия которого описана в четырех типизированных файлах $(F_p F_v F_n F_t)$;

$F_0 -$ суммарный объем пяти файлов $(F_l F_p F_v F_n F_t)$;

$k -$ количество единичных объектов, из которых состоит объект;

F_l – файл-ссылка формата .DBE, который содержит в себе полный путь до остальных четырех файлов: файл полигонов (F_p), файл вершин (F_v), файл нормалей (F_n), файл текстурных координат (F_t);

M – память в байтах отводящаяся типу данных;

p – количество вершин полигона единичного объекта.

4. Структура разрабатываемого формата

Чтобы понять работу программы, создающей разрабатываемый формат .DBE, следует более детально разобраться в структуре формата .OBJ.

Графический формат obj – это текстовый файл, в котором хранятся координаты точек, описание координат текстур к каждой из точек и координаты нормалей этих точек. Файлы данного формата являются полностью текстовыми и хранят ряд параметров, из которых состоит 3D модель. Любой трехмерный объект на простейшем уровне состоит из набора точек, соединённых линиями каждые 3-4 линии образуют полигон, на основе которого строится плоскость [6]:

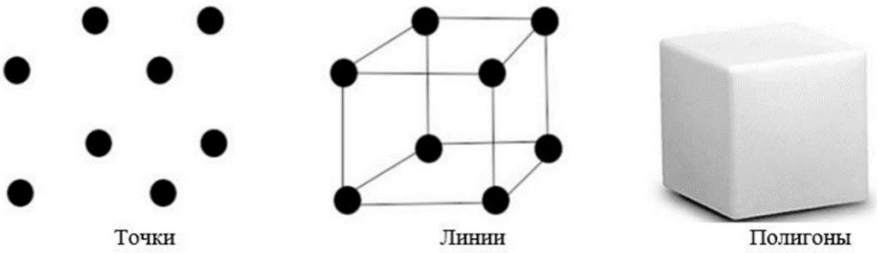


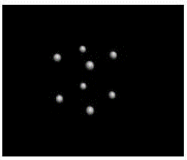
Рис. 3. Структура 3D-файла

Как говорилось выше, в формате .DBE, после конвертации создается пакет из пяти файлов (файл координат вершин, файл координат нормали, файл координат текстур, параметры одной поверхности объекта и solution). Разберем их поподробнее.

Разбор файла (v):

Координаты вершин (v)

Строка с данным параметром начинается с символа v, после данного символа следует 3 координаты, означающие XYZ позицию точки в пространстве:



Координаты x y z
Каждой точки

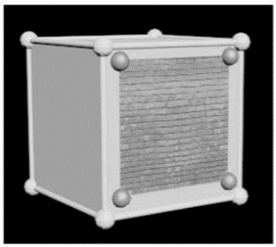
```
# 3ds Max Wavefront OBJ Exporter v0.97b - (c)2007 guruware
# File Created: 02.06.2012 13:38:18

#
# object Box01
#

v  -0.0050 0.0000 0.0050
v  -0.0050 0.0000 -0.0050
v  0.0050 0.0000 -0.0050
v  0.0050 0.0000 0.0050
v  -0.0050 0.0100 0.0050
v  0.0050 0.0100 0.0050
v  0.0050 0.0100 -0.0050
v  -0.0050 0.0100 -0.0050
# 8 vertices
```

Рис. 4. Представление координат вершин в пространстве

Разбор файла (vt):

Координаты текстуры (*vt*)

Строка с данным параметром начинается с символа vt. По сути, на каждую плоскость 3D-объекта можно натянуть текстуру, просто указав координаты картинки – как она будет располагаться на плоскости.



Координаты x y z
Для наложения
текстуры на грань
объекта

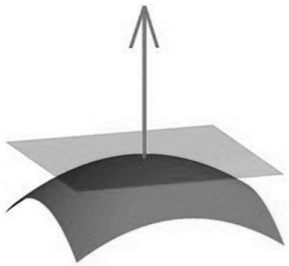
```
# 6 vertex normals

vt 1.0000 0.0000 0.0000
vt 1.0000 1.0000 0.0000
vt 0.0000 1.0000 0.0000
vt 0.0000 0.0000 0.0000
# 4 texture coords
```

Рис. 5. Представление текстуры объекта на плоскости

Разбор файла (vn):

Координаты нормали (*vn*)

Строка с данным параметром начинается с символа vt. Вектор нормали к поверхности в точке совпадает с нормалью к касательной плоскости в этой точке. По этому параметру вычисляется как будет освещаться конкретная точка объекта.



Координаты x y z
каждой нормали
Для просчета
освещения

```
# 8 vertices

vn 0.0000 -1.0000 -0.0000
vn 0.0000 1.0000 -0.0000
vn 0.0000 0.0000 1.0000
vn 1.0000 0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
vn -1.0000 0.0000 -0.0000
# 6 vertex normals
```

Рис.6. Представление вектора нормали, для освещения объекта

Разбор файла (*f*):

Параметры одной поверхности объекта (*f*)

Строка с данным параметром начинается с символа *f*.

По сути, 3D-объект — это просто набор точек, соединённых линиями в правильном порядке. Формат позволяет для каждой грани перечислить номера точек нормалей и текстурных координат, которые описаны в файле выше [6]:

Рис. 7. Описание граней объекта

В результате работы программы получаем следующую файловую структуру:

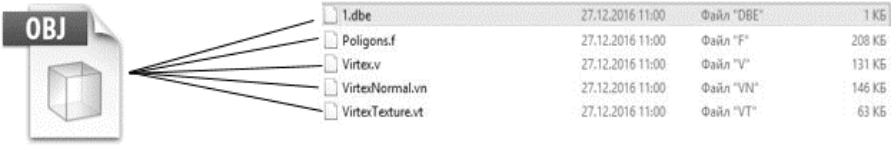


Рис.8. Преобразование формата .OBJ в пакет файлов формата .DBE

5. Постановка и результаты экспериментов.

Для проведения эксперимента было выбрано пять 3D-моделей.

В таблице 3 представлены 3D-модели, а также дан объем каждой из них в определенных форматах.

Таблица 3

Наименование объекта	Объем 3D-модели в формате .OBJ (Кб)	Объем 3D-модели в формате .MAX (Кб)	Объем 3D-модели в формате .DBE (Кб)
Автомобиль	1069	2400	882
Октаэдр	569	1500	350
Тетраэдр	160	990	52
Додекаэдр	700	1530	298
Модель человека	2950	4000	2110

Как видно из проведенного эксперимента, формат .DBE существенно сжимает размер файла .OBJ. И несмотря на то, что некоторые форматы имеют размер меньше, в случае с форматом .DBE отсутствует потеря качества 3D-модели.

6. Заключение

Результаты эксперимента, полученные в ходе тестирования форматов для 3D-изображения, наглядно доказывают эффективность использования разработанного формата .DBE, так как в сравнении с наиболее часто используемыми форматами.OBJ и .MAX графический формат .DBE действительно требует меньший объем компьютерной памяти, что и является конечной целью данной работы.

С учетом того, что размер файла в формате .DBE, в сравнении с сравниваемыми форматами 3D-изображений, является минимальным, впоследствии его можно приспособить для работы в геоинформационных системах. Так как данное направление развивается сейчас очень стремительно, а картография требует больших ресурсов от вычислительной машины, формат .DBE помог бы существенно сократить затраты на хранение 3D-карт.

Литература

1. *Соколова Е. А.* Компрессия изображений вариабельными фрагментами // Вестник компьютерных и информационных технологий. 2008. № 10. С. 31–34.

2. *Соколова Е. А.* Математическая модель компрессии статичных изображений вариабельными фрагментами с учетом погрешностей. Депонированная рукопись № 748-В2007 19.07. 2007.

3. *Соколова Е. А.* Использование теоретико-множественного подхода для поиска необходимого контента по атрибутам и ключевым словам // Фундаментальные исследования. 2013. № 8–6. С. 1360–1363.

4. *Маров А. А.* 3ds max. Реальная анимация и виртуальная реальность.Электронное издание, 2010.

5. *Киан Б. Н.* Цифровые эффекты в MAYA. Создание и анимация [Электронное издание]. 2015.

6. *Кулагин Б. Ю.* 3ds max 7.5. Актуальное моделирование, визуализация и анимация. СПб.: Издательство НПО "Профессионал", 2010.

7. *Колесников В. Ю.* Новые форматы изображений: требование времени // http://www.publish.ru/articles/200102_4043046.

8. *Бутенко В. К.* Загрузка формата obj // https://sites.google.com/site/raznyeurokipoinformatiki/home/opengl-s/zagruzka-formata-obj

Сведения об авторах

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

Бережная Е. Т.,
студентка магистратуры (гр. ИВм-15-1) СКГМИ (ГТУ)
e-mail: elenaamaya@mail.ru

ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ

УДК 004.272

Мирошников А. С.,
Глотова А. В.

ЭФФЕКТИВНОСТЬ ИСПОЛЬЗОВАНИЯ ТЕХНОЛОГИИ CUDA ДЛЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧ НЕЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ БОЛЬШОГО ОБЪЕМА НА ПРИМЕРЕ АЛГОРИТМА ГРАДИЕНТНОГО СПУСКА

Описан алгоритм градиентного спуска и его параллельная реализация с использованием технологии CUDA. Технология CUDA (Compute Unified Device Architecture) – это технология от компании NVidia, позволяющая с большой скоростью выполнять решение сложных задач, работать с большими объемами данных за счет распараллеливания вычислений на десятках тысяч нитей.

Ключевые слова: *нелинейное программирование, градиент, нить, блок, NVidia, CUDA.*

1. Введение

Задачи нелинейного программирования встречаются в естественных науках, технике, экономике, математике, в сфере деловых отношений и в науке управления государством.

Нелинейное программирование связано с основной экономической задачей. Так в задаче о распределении ограниченных ресурсов максимизируют либо эффективность, либо минимизируются затраты при наличии ограничений, которые выражают условия недостатка ресурсов.

Решения задачи нелинейного программирования являются основой при принятии государственных решений. Полученный результат является рекомендуемым, поэтому необходимо исследовать предположения и точность постановки задачи нелинейного программирования перед принятием окончательного решения.

Задачи нелинейного программирования часто возникают и в других отраслях науки. Так, например, в физике целевой функцией может быть потенциальная энергия, а ограничениями – различные уравнения движения. В общественных науках и психологии возникает задача минимизации социальной напряженности, когда поведение людей ограничено определенными законами.

Определимся с понятием задачи нелинейного программирования и методами их решения.

2. Задача нелинейного программирования и метод градиентного спуска

Задачами нелинейного программирования называются задачи математического программирования, в которых нелинейны и (или) целевая функция, и (или) ограничения в виде неравенств или равенств.

Общая формулировка выглядит следующим образом:

Найти переменные $x_1, x_2, \ldots, x_n$, удовлетворяющие системе уравнений

$$\varphi_i(x_1, x_2, \ldots, x_n) = b_i, \quad i = 1,2, \ldots, m$$

и обращающие в максимум (минимум) целевую функцию

$$Z = f(x_1, x_2, \ldots, x_n)$$

Общих способов решения для них не существует. В каждом конкретном случае способ выбирается в зависимости от вида функции F(x). Многие задачи нелинейного программирования могут быть приближены к задачам линейного программирования, и таким образом найдено близкое к оптимальному решение, но в целом они относятся к трудным вычислительным задачам. При их решении часто приходится прибегать к приближенным методам оптимизации – численным методам, которые подразделяются на:

- ***методы прямого поиска*:** в этих методах для определения направления спуска не требуется вычислять производные целевой функции. Направление минимизации в данном случае полностью определяется последовательными вычислениями значений функции.
- ***градиентные методы:*** методы, использующие только значения 1-й производной целевой функции, по которой можно оценить

необходимое условие экстремума, а значит, выбрать верное направление спуска.

- ***методы второго порядка*:** методы, использующие значения 1-й и 2-й производных целевой функции. В этом случае выполняются как необходимые, так и достаточные условия экстремума. Ярким примером этих методов является метод Ньютона и его модификации [1].

Остановимся на рассмотрении градиентного спуска с дроблением шага.

Общий его алгоритм выглядит следующим образом:

Шаг1. Задается функция F(x1,x2,…,xn) и начальная точка отсчета X_o, а также указывается точность вычислений e и шаг H.

Шаг 2. Определяется градиент функции Grad.

Шаг 3. Осуществляется проверка: является ли вычисленный градиент по модулю меньше заданной точности вычислений. Если да – перейти к шагу 6, если нет – к шагу 4.

Шаг 4. Вычисляется значение функции в текущей точке F1, затем определяются новые координаты согласно формуле (1):

$$x_i^{k+1} = x_i^k - H * grad\left(x_i^k\right). \quad (1)$$

Далее идет вычисление нового значения функции F2.

Шаг 5. Проверка F1<F2. Если да – заменить старые значения (x1,x2,…,xn) на новые и перейти к шагу 2. Иначе – уменьшить шаг H в 2 раза и перейти к шагу 4.

Шаг 6. Завершение вычислений [2].

Этот алгоритм для двух переменных представлен на рис. 1.

Если количество переменных велико, то процесс вычисления точки минимума функции может затянуться на долгое время, поэтому в 2005 году студентом Киевского национального университета имени Тараса Шевченко Ниссенбаумом Геннадием был предложен параллельный алгоритм градиентного спуска, который подразумевал расчет нового значения переменной x_i отдельным потоком, а определение значения функции – (i+1) потоком[3]. Схожий алгоритм описан и в статье Атамуратова А. Ж. "Использование методик параллельного программирования при численном решении задач оптимизации методами координатного и градиентного спусков на примере задач гашения колебаний"[4] .

Однако найденное значение необязательно будет являться глобальным минимумом функции в заданной области, если она имеет несколько точек экстремума . Зачастую градиентные методы приводят к некой стационарной точке или локальному экстремуму, что не удовлетворяет интересам решающего задачу человека.

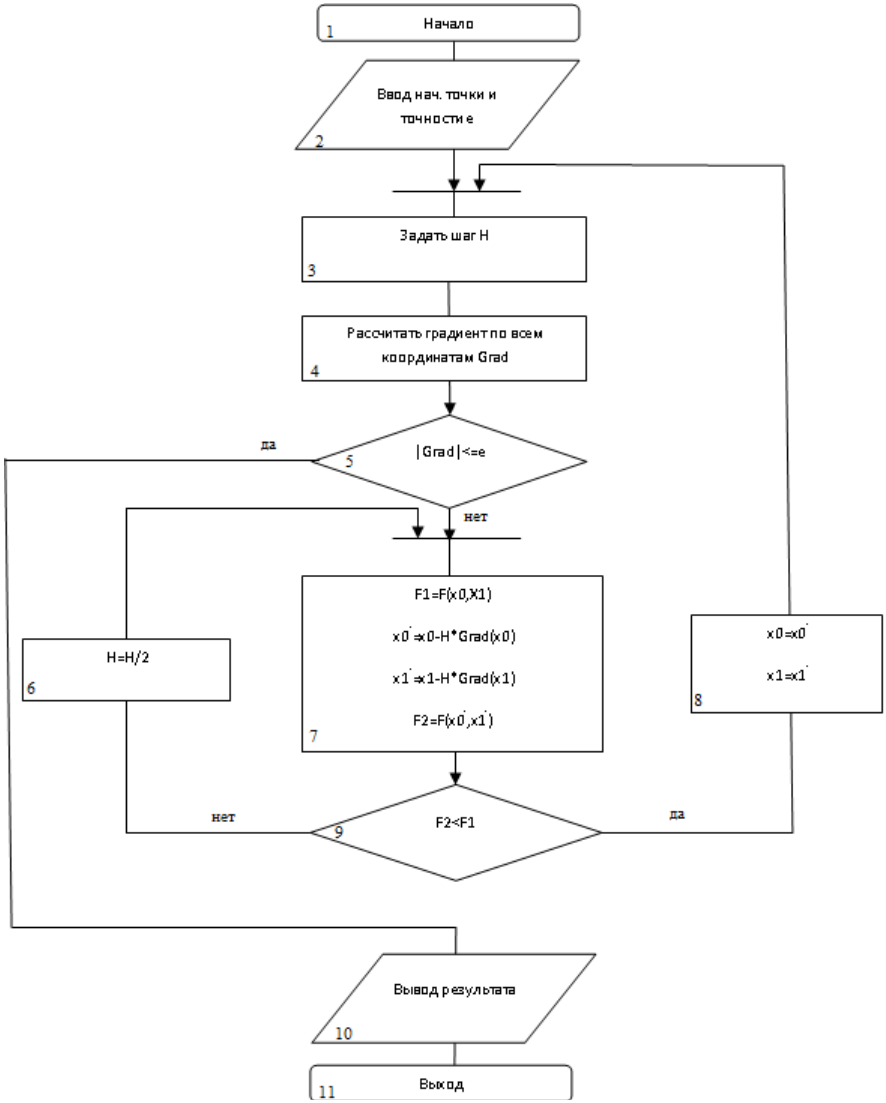


Рис. 1. Блок-схема градиентного спуска с дроблением шага

Проблему нахождения глобального экстремума при наличии нескольких точек минимума можно обойти с помощью разбиения области поиска на части – наложения на неё сетки, – и вычисления экстремума градиентным спуском из каждого узла сетки (рис. 2).

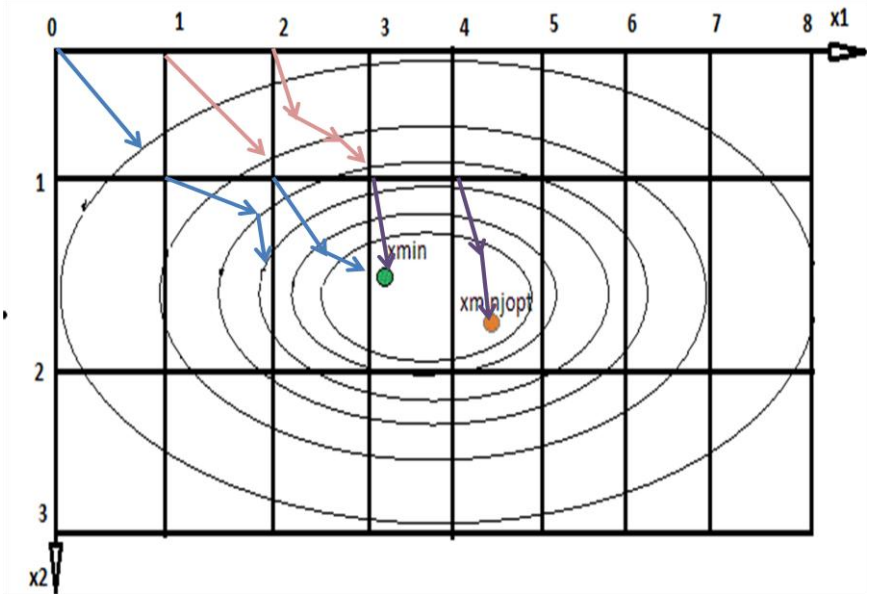


Рис. 2. Спуск из каждого узла на примере трех градиентных спусков

Чем мельче берется сетка, тем выше вероятность попадания в глобальный экстремум. Однако стоит обратить внимание на то, что в этом случае поиск глобального экстремума обращается в трудоемкую задачу, требующую длительного времени для нахождения её решения и больших вычислительных ресурсов для рассмотрения каждого выделенного участка области.

Рассмотрим параллельный алгоритм градиентного спуска, созданный для определения глобального экстремума.

Шаг 1. Ввод исходных данных: количество переменных k, функция F, граничные значения для каждой переменной (a_{imin},a_{imax}), i=1…k, количество частей , на которое делится каждая переменная m, начальный шаг H и точность вычислений e.

Шаг 2. Разбиение области определения на m^k частей.

Шаг 3. Передача потоку данных для расчета – граничных значений переменных (b_{imin},b_{imax}), i=1…k в той части области, в которой этот поток будет осуществлять поиск экстремума.

Шаг 4. Поиск экстремума каждым потоком согласно вышеприведенному алгоритму нахождения экстремума градиентным методом с дроблением шага.

Шаг 5. Передача найденной точки минимума по завершению работы потока с экстремумами, обнаруженными другими потоками в заданных областях.

Шаг 6. Вывод значения минимальной целевой функции и координат точки, в котором оно достигается.

Описанный алгоритм приведен в виде блоксхемы ниже на рис. 3:

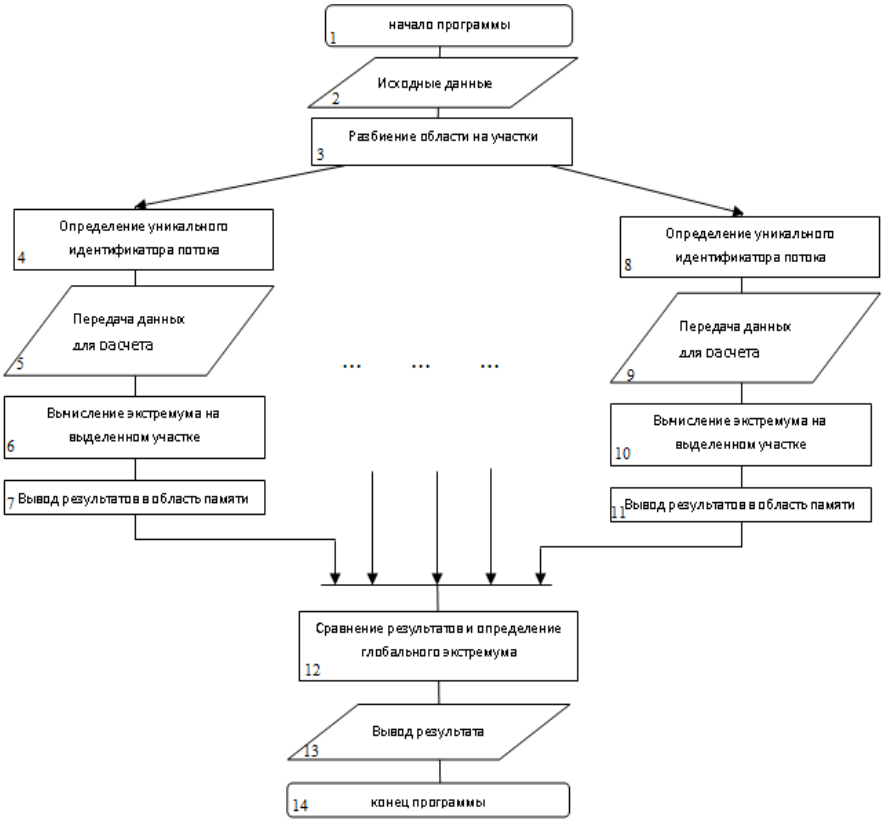


Рис. 3. Параллельный алгоритм градиентного спуска

Для программной реализации описанного алгоритма можно задействовать как возможности центрального процессора компьютера, используя технологию распараллеливания OpenMPI или библиотеку Threads, так и возможности графических процессоров видеокарты с помощью технологии CUDA от компании NVIDIA.

OpenMP (OpenMulti-Processing) — это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью (SMP-системах) [5].

Threads – библиотека поддержки потоков, взаимных исключений (мьютексов), условных переменных и фьючеров [6].

CUDA (Compute Unified Device Architecture) – это архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную производительность благодаря использованию GPU (графических процессоров).

Основными её понятиями являются:

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).[7]

3. Математическая модель

Для оптимизации использования графического процессора необходимо в первую очередь оптимизировать количество нитей x, используемых в параллельных алгоритмах, не превышающее максимального числа активных нитей, поддерживаемых GPU-P. С возрастанием их количества производительность обработки должна возрастать, но в то же время с возрастанием количества нитей будет неизбежно возрастать и время на организацию их работы. Принимая во внимание эти факторы, мною была выбрана математическая модель, целевой функцией которой является минимизация общего времени выполнения поиска глобального экстремума. Оно представляет собой сумму трех времен:

– среднего времени на обработку введенных с формы данных и разбиение области поиска экстремума на части – c, c = const;

– среднего времени на организацию работы нитей, представляющего собой сумму произведений времени на организацию одной нити

f на количество нитей *x*, т. к. передача данных нитям происходит последовательно;

– времени нахождения экстремума, представляющего также сумму частных от деления времени поиска экстремума одной нитью – d на количество нитей, задействованных для расчета, т. к. процесс происходит параллельно, и время нахождения точки минимума уменьшается с ростом числа нитей.

В общем виде целевую функцию можно записать как (2):

$$T = c + \sum_{i=1}^{\infty} f_i \cdot x^i + \sum_{i=1}^{\infty} \frac{d_i}{x^i} \to \min \tag{2}$$

В первом приближении при i=1 получим целевую функцию (3):

$$T = c + f \cdot x + \frac{d}{x} \to \min \tag{3}$$

Изменяя количество нитей для вычисления экстремума функции

$$F(x, y)=3\cdot x^2 - x^3 - y^4 ,$$

найдем время выполнения вычислений для трех значений x: 10, 50, 100. Составим систему из трех линейных уравнений (4) и определим значения коэффициентов c, f и d методом Крамера:

$$\begin{cases} c + 10f + \frac{d}{10} = 1112.57 \\ c + 50f + \frac{d}{50} = 323.67 \\ c + 100f + \frac{d}{100} = 225.39 \end{cases} \tag{4}$$

$$\Delta = \begin{vmatrix} 1 & 10 & 1/10 \\ 1 & 50 & 1/50 \\ 1 & 100 & 1/100 \end{vmatrix} = 3{,}6 \Rightarrow \Delta c = \begin{vmatrix} 1112{,}57 & 10 & 1/10 \\ 323{,}67 & 50 & 1/50 \\ 225{,}39 & 100 & 1/100 \end{vmatrix} = 453{,}606 \Rightarrow$$

$$\Delta f = \begin{vmatrix} 1 & 1112{,}5 & 1/10 \\ 1 & 323{,}67 & 1/50 \\ 1 & 225{,}39 & 1/100 \end{vmatrix} = 0{,}0266 \Rightarrow \Delta d = \begin{vmatrix} 1 & 10 & 1112{,}5 \\ 1 & 50 & 323{,}67 \\ 1 & 100 & 225{,}39 \end{vmatrix} = 35513{,}8$$

$$c = \frac{453{,}606}{3{,}6} = 126{,}002 \quad f = \frac{0{,}0266}{3{,}6} = 0{,}00738 \quad d = \frac{35513{,}8}{3{,}6} = 9864{,}94$$

Таким образом, наша целевая функция принимает вид (5):

$$T = 126 + 0{,}0074 \cdot x + \frac{9865}{x} \to \min \tag{5}$$

Для проверки адекватности полученной целевой функции и корректности найденных коэффициентов проведем эксперимент, в котором будем менять количество нитей, используемых для вычисления, и замерять время нахождения экстремума. После чего сравним времена T_k, полученные аналитическим путем через подстановку количества нитей в целевую функцию, с временами t_k, полученными экспериментально с помощью замера времени работы кода программы. Результаты эксперимента представлены в табл. 1:

Таблица 1

Время вычисления экстремума в зависимости от количества нитей

Количество потоков	Аналитическое время, T, мс	Экспериментальное время, t, мс
1	9991,0074	9996,0086
10	1112,57	1122,6468
20	619,398	625,08
30	455,055	462,678
40	372,921	379,896
50	323,67	324,745
60	290,86	290,96
70	267,446	268,578
80	249,9045	247,683
90	236,277	232,365
100	225,39	218,782

Рис. 4. График зависимости времени нахождения экстремума от количества нитей

Как можно видеть из рис. 4, расхождение между аналитическим временем и экспериментальным не превышает 0,1 секунды, из чего делаем вывод, что допущение (3) верно и коэффициенты вычислены правильно. В итоге получим следующую математическую модель:

$$\begin{cases} T = c + f \cdot x + \dfrac{d}{x} \to \min \\ 1 \le x \le P \end{cases} \qquad (6)$$

Решив систему (6), определяем оптимальное количество нитей, необходимое для того, чтобы время вычисления было минимальным.

Для использования GPU, найденное количество нитей необходимо разбить на блоки таким образом, чтобы нагрузка на видеоускоритель была максимальной. Для подсчета этих параметров на сайте *http://www.nvidia.ru/object/cuda-parallel-computing-ru.html* компании NVIDIA есть алгоритм расчета количества нитей на блок, математическая модель (7) которого представлена ниже:

$$\begin{cases} N_a = \min(N_{\max}, B_a N) \to \max \\ B_a = \min(B_{\max}, \left\lfloor \dfrac{W_{\max}}{W} \right\rfloor) \to \max \\ N_{\max} = W_{\max} N_W \\ W = \left\lceil \dfrac{N}{N_W} \right\rceil \\ 1 \le N \le N_{B\max}, \text{ целое} \end{cases} \quad (7)$$

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.
$\lfloor\ \rfloor$ – округление в меньшую сторону до ближайшего целого,
$\lceil\ \rceil$ – округление в большую сторону до ближайшего целого.

Пример расчета оптимального количества нитей

Рассчитаем количество нитей, необходимое для минимального времени вычисления экстремума функции $F(x,y) = 3 \cdot x^2 - x^3 - y^4$ с точностью вычисления eps = 0,1; начальным шагом h = 0,5. Область определения по каждой переменной мы разобьем на m = 10 частей.

Области определения координат:

	x	y
Начало интервала	0	0
Конец интервала	1000	1000

Программно определим:
время нахождения экстремума одной нитью d = 9865 мс;

среднее время на организацию работы одной нити f = 0,0074 мс;
среднее время на организацию вычислений c = 126 мс.

При расчете будет использоваться видеокарта Nvidia Geforce GT 720 со следующими параметрами:

Версия CUDA 6.5
Количество варпов Wmax = 64
Количество нитей в варпе Nw = 32
Количество блоков Bmax = 32
Количество мультипроцессоров SM = 2

Для начала определим: сколько активных нитей нам доступно:

$$N_{\max} = W_{\max} \cdot N_W = 64 \cdot 32 = 2048;$$

$$N = \frac{N_{\max}}{B_{\max}} = \frac{2048}{32} = 64;$$

$$W = \left\lceil \frac{N}{N_W} \right\rceil = \left\lceil \frac{64}{32} \right\rceil = 2.$$

Следовательно, максимальное количество нитей на GPU составит P = 2048 · 2 = 4096.

Целевая функция имеет вид:

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

Найдем первую производную от целевой функции:

$$T'(x) = 0 + f + d \cdot (-\frac{1}{x^2}) = 0, \text{ следовательно } x' = \sqrt{\frac{d}{f}},$$

$$x = \sqrt{\frac{9865}{0{,}0074}} = 1154{,}67$$

Найдена точка экстремума. Убедимся, что она является минимумом, найдя вторую производную:

$$T''(x) = \frac{2d}{x^3} = \frac{2 \cdot 9865}{1154.67^3} = 0{,}0000128 > 0.$$

Т. к. вторая производная больше нуля, x является точкой минимума.

Мы получили дробное значение x. Воспользовавшись системой (8), определим точное количество нитей, необходимых нам для вычисления экстремума:

$$x = \begin{cases} 1, x' < 1; \\ p, x' > p; \\ \lfloor x \rfloor, T(\lfloor x \rfloor) < T(\lceil x \rceil); \\ \lceil x \rceil, \ T(\lceil x \rceil) \geq T(\lceil x \rceil). \end{cases} \quad (8)$$

$$T(1154) = 126 + 0{,}0074 \cdot 1154 + \frac{9865}{1154} = 143{,}088;$$

$$T(1155) = 126 + 0{,}0074 \cdot 1155 + \frac{9865}{1155} = 143{,}087.$$

Исходя из формулы (8), получим:

$x = \lfloor x \rfloor = 1155$; время работы $T(x) = 143{,}087$ мс.

Разобьем полученное число нитей на блоки:

$$B = \left\lceil \frac{x}{N} \right\rceil = \left\lceil \frac{1155}{64} \right\rceil = 18.$$

Таким образом, для данной видеокарты получены следующие оптимальные параметры:

– число активных нитей x = 1155;

– количество нитей в блоке N = 64;

– число блоков B = 18.

Следовательно, эти параметры нужно использовать в программе для минимизации времени решения задачи на данной видеокарте.

4. Интерфейс программы и исследование эффективности параллельного алгоритма градиентного спуска, реализованного с помощью технологии CUDA

Интерфейс разработанного программного комплекса имеет следующий вид:

1. Функция и начальные данные задаются на форме, представленной на рис. 5:

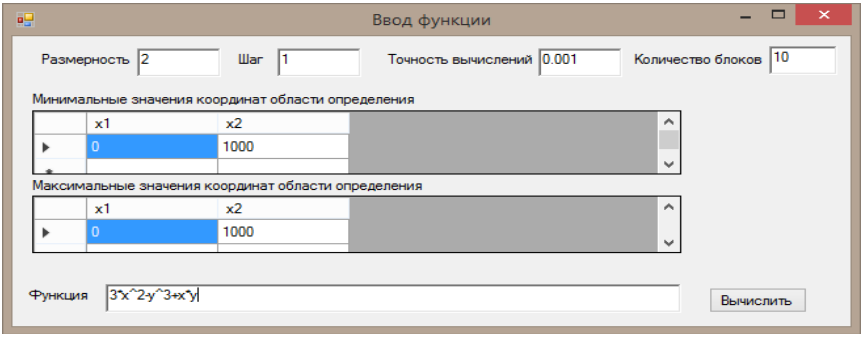


Рис. 5. Ввод вычисляемой функции и параметров вычисления

2. Решение задачи нелинейного программирования параллельным алгоритмом градиентного спуска представляется в отдельных окнах, как показано на рис. 6:

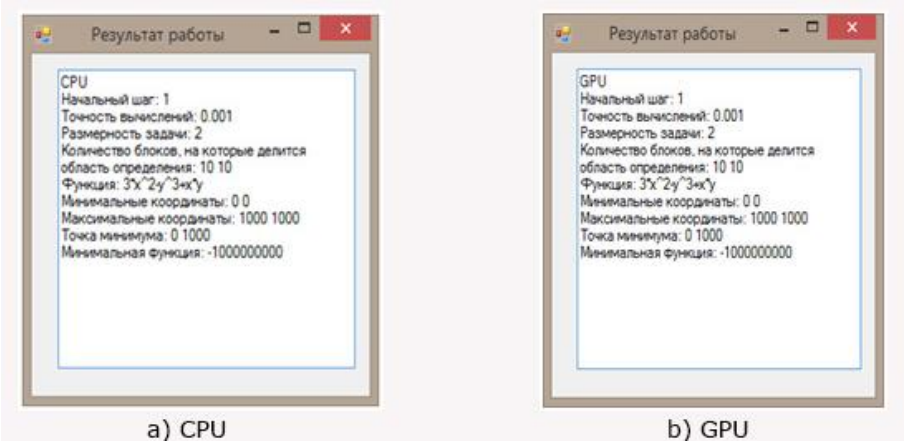


Рис. 6. Результаты поиска экстремума параллельным алгоритмом градиентного спуска

Для проверки эффективности использования технологии CUDA проведем эксперимент на определение зависимости времени выполнения параллельного алгоритма градиентного спуска на CPU и GPU от размерности задачи.

Эксперимент проводился на ПК, имеющем следующие характеристики:

Intel Core i5-3337U, 1,80GHz,ОЗУ-8Гб, ОС- Windows 8.1– 64–bit

Видеокарта Nvidia Geforce GT 720, версия CUDA 6.5

Решалась задача нахождения минимума функции: $3 \cdot x^2 - y^3 + x \cdot y$.

Точность вычислений: 0,0001

Начальный шаг: 1.

При изменении количества частей, на которые разбивалась область определения, были получены следующие результаты:

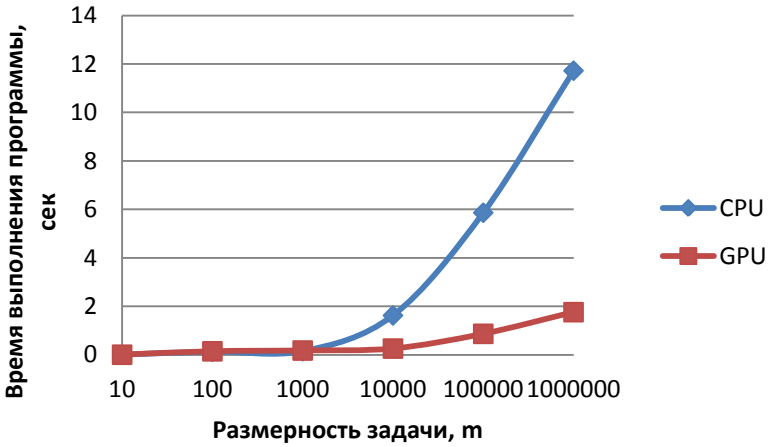


Рис. 7. График зависимости времени вычисления экстремума методом градиентного спуска от размерности задачи на CPU и на GPU

Как видно из эксперимента, время, затрачиваемое на выполнение поиска экстремума с помощью технологии CUDA не имеет преимущества перед реализацией на CPU при малых размерностях решаемой задачи, но значительно меньше времени, которое требуется на решение этой же задачи CPU при больших размерностях.

5. Заключение

Результаты эксперимента, полученные в ходе тестирования программы, наглядно доказывают эффективность использования технологии CUDA от компании NVIDIA при реализации параллельного алгоритма градиентного спуска для задач большой размерности. В сравне-

нии с итогами программы, где распараллеливание происходило на уровне переменных, приведенная в статье реализация выигрывает, т. к. не требует затрат времени на многократное создание потоков и передачу данных от них, потому что изначально определяет количество создаваемых потоков и получает в качестве результата не значение одной переменной, требующей дальнейшего вычисления функции, а непосредственно экстремум в заданной области.

Реализация на CUDA позволяет задействовать больше потоков для вычисления экстремума, чем CPU, а также имеет более высокую пропускную способность, что выгодно при работе с огромными объемами данных, поэтому является выигрышной технологией при выполнении независимых параллельных вычислений, часто применяемых при решении задач нелинейного программирования. Также использование возможностей видеокарты для математических вычислений наиболее выгодно с экономической точки зрения, т. к. её стоимость значительно меньше стоимости многопроцессорного компьютера, способного соперничать по скорости и объемам работы с видеокартой.

Литература

1. *Трифонов А. Г.* Постановка задачи оптимизации и численные методы ее решения. М.: Изд-во Высшая школа, 2007, 645 с.
2. *Пантелеев А. В.* Методы оптимизации в примерах и задачах. М.: Изд-во «Высшая школа», 2009. 544 с.
3. *Ниссенбаум Г.* Разработка алгоритмов параллельных вычислений для решения задач моделирования сложных систем. Киев, 2005.
4. *Атамуратов А. Ж.* Использование методик параллельного программирования при численном решении задач оптимизации методами координатного и градиентного спусков на примере задач гашения колебаний // Молодой учёный. № 1(60) январь 2014. Казань, 2014.
5. *Левин М. П.* Параллельное программирование с использованием OpenMP. М.: Изд-во "БИНОМ Лаборатория знаний ". 2012. 121 с.
6. *Герб Саттер.* Исключительный стиль C++. М.: Изд-во "Вильямс", 2015. 400 с.
7. *Дж. Сандерс, Э. Кэндрот*. Технология CUDA в примерах. Введение в программирование графических процессоров. М.: Изд-во "ДМК Пресс", 2011. 232 с.
8. *Мирошников А. С.* Система управления параллельной обработки данных в локальных вычислительных сетях. Дисс. … канд. техн. наук. Владикавказ: СКГМИ (ГТУ). 2000.

Сведения об авторах

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

Глотова А. В.,
студента магистратуры (гр. ИВм-15-1)
СКГМИ (ГТУ)
e-mail: flameforyou@bk.ru

УДК 004. 272

Мирошников А. С.,
Датиев А. А.

ЭФФЕКТИВНОСТЬ ИСПОЛЬЗОВАНИЯ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ШИФРОВАНИЯ BLOWFISH В ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С ОБЩЕЙ ПАМЯТЬЮ ПОД УПРАВЛЕНИЕМ ОПЕРАЦИОННЫХ СИСТЕМ WINDOWS И LINUX

В статье описана параллельная реализация алгоритма шифрования Blowfish. Производится сравнительный анализ работы алгоритма шифрования в вычислительных системах с общей памятью, под управлением операционных систем Windows и Linux.

Ключевые слова: *шифрование, дешифрование, Blowfish, поток, операционная система, Windows, Linux.*

1. Введение

Алгоритм Blowfish был разработан Брюсом Шнайером в 1994 г. Автор алгоритма предложил его в качестве замены стандарту DES. Несомненно, в тот момент замена DES новым стандартом шифрования была уже актуальна из-за короткого ключа DES, который к тому времени было возможно найти путем полного перебора. Брюс Шнайер предположил, что других реальных кандидатов на замену DES нет, в частности, по следующим причинам (описаны Шнайером в спецификации алгоритма Blowfish):

- многие известные и криптографически стойкие (или считавшиеся таковыми на момент разработки алгоритма Blowfish) алгоритмы, например, IDEA или REDOC-II, запатентованы, что ограничивает их использование;
- спецификация алгоритма ГОСТ 28147-89 не содержит значений таблиц замен, т. е., по мнению Шнайера, алгоритм описан не полностью.

Цель данной статьи:

- разработка и реализация параллельного алгоритма шифрования Blowfish.
- проверить, есть ли выигрыш по скорости шифрования данных параллельного алгоритма шифрования Blowfish, реализованного для операционных систем Windows и Linux.

2. Описание многопоточности в Windows и Linux

В среде MicrosoftWindows процесс, – это контейнер для потоков (именно этими словами о процессах говорит Джефри Рихтер в своей классической книге «Программирование приложений для Microsoft Windows») [1]. Процесс-контейнер содержит как минимум один поток. Если потоков в процессе несколько, приложение (процесс) становится многопоточным.

В мире Linux все выглядит иначе. В Linux каждый поток является процессом, и для того чтобы создать новый поток, нужно создать новый процесс. Преимущество многопоточности Linux перед многопроцессностью Windows заключается в том, что в многопоточных приложениях Linux для создания дополнительных потоков используются процессы особого типа.

Эти процессы представляют собой обычные дочерние процессы главного процесса, но они разделяют с главным процессом:

1) адресное пространство;
2) файловые дескрипторы;
3) обработчики сигналов.

Для обозначения процессов этого типа применяется специальный термин – легкие процессы (light weight processes).

Прилагательное «легкий» в названии процессов-потоков вполне оправдано. Поскольку этим процессам не нужно создавать собственную копию адресного пространства (и других ресурсов) своего процесса-родителя, создание нового легкого процесса требует значительно меньших затрат, чем создание полновесного дочернего процесса, что наглядно показано на рис.1.

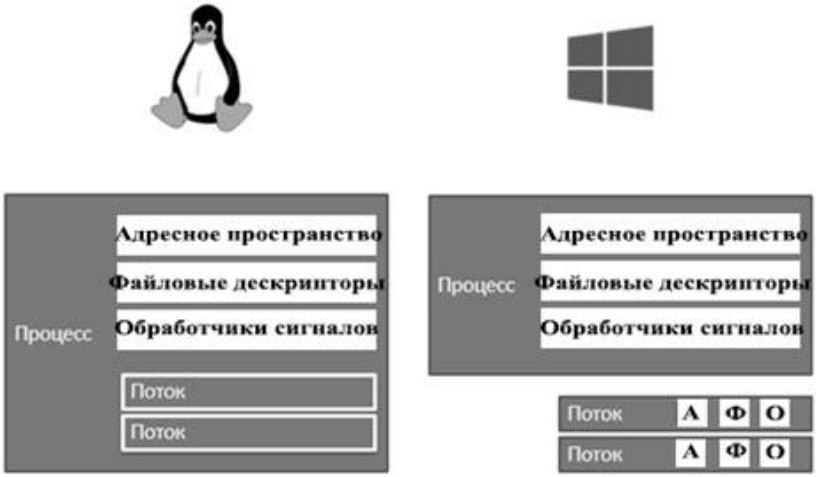


Рис. 1. Организация многопоточности в операционных системах Linux и Windows

3. **Описание алгоритма шифрования Blowfish**

Blowfish – криптографический алгоритм, реализующий блочное симметричное шифрование с переменной длиной ключа. Разработан Брюсом Шнайером в 1993 году [2]. Представляет собой сеть Фейстеля. Выполнен на простых и быстрых операциях: XOR, подстановка, сложение. Является незапатентованным и свободно распространяемым.

До появления Blowfish существовавшие алгоритмы были либо запатентованными, либо ненадёжными, а некоторые и вовсе держались в секрете (например, Skipjack). Алгоритм был разработан в 1993 году Брюсом Шнайером в качестве быстрой и свободной альтернативы устаревшему DES и запатентованному IDEA. По заявлению автора, критериями проектирования Blowfish были:

– скорость (шифрование на 32-битных процессорах происходит за 26 тактов);

– простота (за счёт использования простых операций, уменьшающих вероятность ошибки реализации алгоритма);

– компактность (возможность работать в менее, чем 5 Кбайт памяти);

– настраиваемая безопасность (изменяемая длина ключа).

Алгоритм состоит из двух частей: расширение ключа и шифрование данных. На этапе расширения ключа исходный ключ (длиной до 448 бит) преобразуется в 18 32-битовых подключей и в 4 32-битных S-блока, содержащих 256 элементов. Общий объём полученных ключей равен $(18 + 256 \cdot 4) \cdot = 33344$ бит или 4168 байт.

Параметры:

секретный ключ K (от 32 до 448 бит)

32-битные ключи шифрования $P_1 – P_{18}$

32-битные таблицы замен $S_1 – S_4$:

$S_1[0]$, $S_1[1]$, … $S_1[255]$

$S_2[0]$, $S_2[1]$, … $S_2[255]$

$S_3[0]$, $S_{,3}[1]$, … $S_3[255]$

$S_4[0]$, $S_4[1]$, … $S_4[255]$

4. Функция F(x)

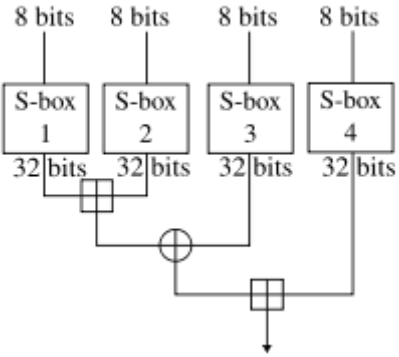


Рис. 2. Функция F(x) в Blowfish

Функция F(x) принимает на вход блок размером в 32 бита и проделывает с ним следующие операции:

– 32-битный блок делится на четыре 8-битных блока (X_1, X_2, X_3, X_4); каждый из которых является индексом массива таблицы замен S_1– S_4;

– значения $S_1[X_1]$ и $S_2[X_2]$ складываются по модулю 2^{32}, после складываются по модулю 2с $S_3[X_3]$ и, наконец, складываются с $S_4[X_4]$ по модулю 2^{32}.

Результат этих операций – значение F(x).

$$F(X_1, X_2, X_3, X_4) = ((((S_1[X_1] \text{ и } S_2[X_2]) \bmod 2^{32}) \oplus S_3[X_3]) + S_4[X_4]) \bmod 2^{32}.$$

Алгоритм шифрования 64-битного блока с известным массивом P и F(x)

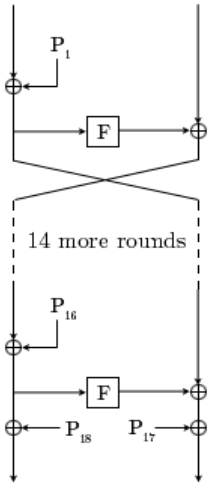


Рис. 3. Сеть Фейстеля при зашифровании

Blowfish представляет собой сеть Фейстеля [3], состоящую из 16 раундов. Алгоритм шифрования блока X размером 64 бит выглядит следующим образом [4]:

1. Разделение входного блока X на 2 32-битных блока L_0, R_0.
2. Для i = 1 …16:

$R_i = L_{i-1} \oplus P_{i+1}$;

$L_i = R_{i-1} \oplus F(R_i)$.

3. После 16 раунда L_{16}, R_{16} меняются местами:

$R_{16} \leftrightarrows L_{16}$

$L_{16} \leftarrow R_{16}$

4. К получившимся блокам прибавляются P_{17} и P_{18}

$L_{17} \leftarrow L_{16} \oplus P_{18}$

$R_{17} \leftarrow R_{16} \oplus P_{17}$

5. Выходной блок Y равен конкатенации (объединению) L_{17} и R_{17}.

Алгоритм Blowfish разделён на 2 этапа:

1. Подготовительный – формирование ключей шифрования по секретному ключу.

1.1. Инициализация массивов P и S при помощи секретного ключа K

1.1.1. Инициализация P_1–P_{18} фиксированной строкой, состоящей из шестнадцатеричных цифр мантиссы числа пи.

1.1.2. Производится операция XOR над P_1 с первыми 32 битами ключа K, над P_2 со вторыми 32-битами и так далее.

Если ключ K короче, то он накладывается циклически.

1.2. Шифрование ключей и таблиц замен

1.2.1. Алгоритм шифрования 64-битного блока, используя инициализированные ключи и таблицу замен S_1–S_4, шифрует 64 битную нулевую (0x0000000000000000) строку. Результат записывается в P_1, P_2.

1.2.2. P_1 и P_2 шифруются изменёнными значениями ключей и таблиц замен. Результат записывается в P_3 и P_4.

1.2.3. Шифрование продолжается до изменения всех ключей P_1–P_{18} и таблиц замен S_1–S_4.

2. Шифрование текста полученными ключами и F(x) с предварительным разбиением на блоки по 64 бита. Если невозможно разбить начальный текст точно на блоки по 64 бита, используются различные режимы шифрования для построения сообщения, состоящего из целого числа блоков. Суммарная требуемая память 4168 байт.

Дешифрование происходит аналогично, только P_1–P_{18} применяются в обратном порядке.

Выбор начального значения P-массива и таблицы замен

Нет ничего особенного в цифрах числа π. Данный выбор заключается в инициализации последовательности, не связанной с алгоритмом, которая могла бы быть сохранена как часть алгоритма или полу-

чена при необходимости. Как указывает[5] Шнайер: «Подойдёт любая строка из случайных битов – цифры числа e, RAND-таблицы или биты с выхода генератора случайных чисел».

Так как над блоками данных в алгоритме шифрования Blowfish производятся однотипные операции, то целесообразно использовать распараллеливание, что позволит значительно сократить время шифрования массива данных.

На рис. 4 представлен параллельный алгоритм шифрования Blowfish, реализованный на основе параллельного блочного алгоритма шифрования [6].

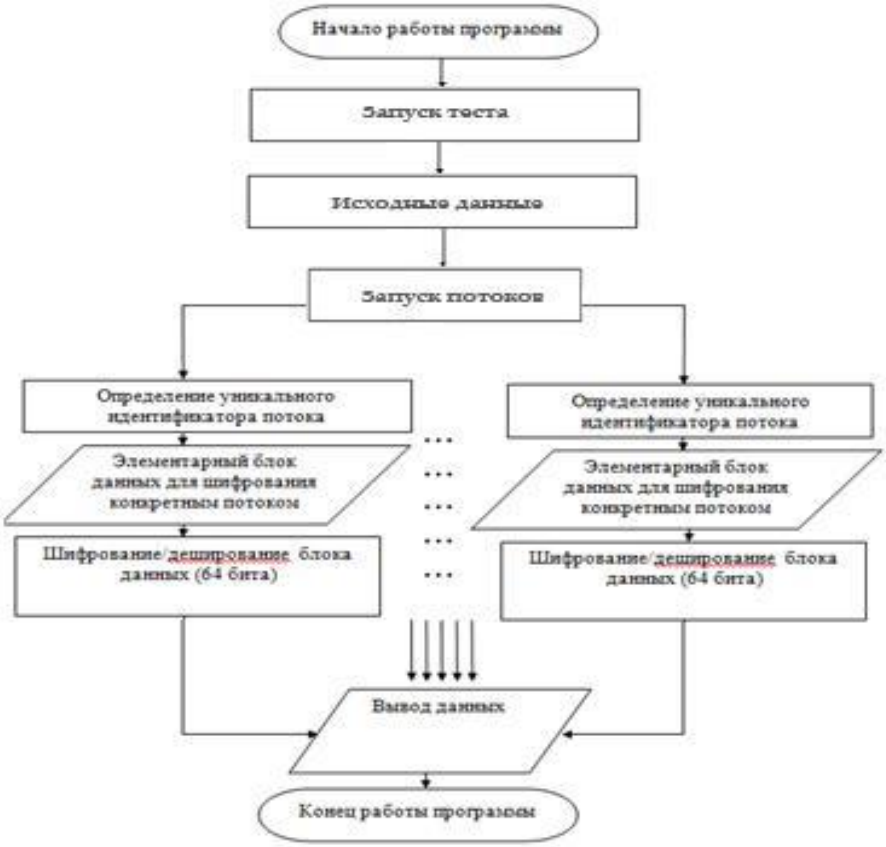


Рис. 4. Параллельный алгоритм шифрования Blowfish

5. Математическая модель

Для эффективного использования центрального процессора на каждой из операционных систем необходимо оптимизировать количество потоков, используемых при реализации параллельного алгоритма шифрования. С возрастанием количества доступных потоков – P производительность обработки данных должна возрастать, но в то же время с возрастанием количества потоков будет возрастать время на организацию – f их работы. Принимая во внимание эти факторы, мною была выбрана следующая математическая модель[7, 8], в которой также учтены среднее время на организацию вычислений на выбранной операционной системе, c – $\boldsymbol{c}$; количество задействованных потоков $-x_i$,среднее время шифрования одного элемента массива данных одним потоком, c – d и размер массива i-го файла, Мб – n_i**:**

В общем виде целевую функцию можно записать как:

$$T = c + \sum_{i=1}^{\infty} f_i * \boldsymbol{n}_i * x^i + \sum_{i=1}^{\infty} \frac{d_i \boldsymbol{n}_i}{x^i} \to \min \qquad (1)$$

В первом приближении при I = 1 получим:

$$T = c + f \cdot x \cdot n + \frac{d \cdot n}{x}. \qquad (2)$$

Изменяя количество потоков для шифрования файла размером 26 Мб, найдем время выполнения обработки для трех значений x: 1, 2, 4. Составим систему из трех линейных уравнений и определим значения коэффициентов c, f и d методом Крамера:

$$\begin{cases} c + f \cdot 26 + 26 \cdot d = 2.97 \\ c + 2 \cdot f \cdot 26 + 13 \cdot d = 1.78 \\ c + 4 \cdot f \cdot 26 + 6.5 \cdot d = 1.42 \end{cases}$$

$$\Delta = \begin{vmatrix} 1 & 26 & 26 \\ 1 & 52 & 13 \\ 1 & 104 & 6{,}5 \end{vmatrix} = 507 \Rightarrow \Delta c = \begin{vmatrix} 2{,}97 & 26 & 26 \\ 1{,}78 & 52 & 13 \\ 1{,}42 & 104 & 6{,}5 \end{vmatrix} = 60{,}8 \Rightarrow \Delta f =$$

$$= \begin{vmatrix} 1 & 2{,}97 & 26 \\ 1 & 1{,}78 & 13 \\ 1 & 1{,}42 & 6{,}5 \end{vmatrix} = 3{,}055;\ \Delta d = \begin{vmatrix} 1 & 26 & 2{,}97 \\ 1 & 52 & 1{,}78 \\ 1 & 104 & 1{,}42 \end{vmatrix} = 52{,}52$$

$$c = \frac{60{,}84}{507} = 0{,}12 f = \frac{3{,}055}{507} = 0{,}00603 d = \frac{52{,}52}{507} = 0{,}1.$$

Таким образом, наша целевая функция принимает следующий вид (3):

$$T = 0{,}12 + 0{,}00603 \cdot 26 \cdot x + \frac{2{,}6}{x} \to \min . \quad (3)$$

Для проверки адекватности полученной целевой функции и корректности найденных коэффициентов проведем эксперимент, в котором будем менять количество потоков, используемых для шифрования, и замерять время обработки данных. После чего сравним значения, полученные аналитическим путем, со значениями, полученными в ходе эксперимента. Результаты представлены в табл. 1.

Таблица 1

Время шифрования

Количество потоков	Аналитическое время, t, мс	Экспериментальное время, t, мс
1	2,97	3,02
2	1,78	1,93
4	1,42	1,51
6	1,49	1,67
8	1,7	1,74
10	1,95	2,03
12	2,2	2,41

Как можно видеть из табл. 1, расхождение между аналитическим временем и экспериментальным не превышает 0,3 секунды, из чего делаем вывод, что целевая функция адекватна и коэффициенты вычислены правильно. В итоге получим следующую математическую модель:

$$\begin{cases} T = \sum_{i=1}^{q} (c + fn_i x_i + \frac{dn_i}{x_i}) \to \min \\ \forall i : 1 \le x_i \le P, целое \\ i = \overline{1, q} \end{cases} \quad (4)$$

В следующем примере решим поставленную задачу для операционной системы Windows, определив оптимальное количество потоков x_i, необходимое для того чтобы время работы программного комплекса на операционной системой Windows было минимальным. Количе-

ство доступных потоков P = 4, время обработки массива одним потоком d = 0,106, среднее время на организацию потока f = $6{,}8 \cdot 10^{-3}$ и среднее время на организацию вычислений c = 0,112 были получены экспериментальным путем посредством запуска теста подсистемы, приведенного в блок-схеме работы параллельного алгоритма на рис. 3.

На вход для шифрования программного комплекса подается массив данных, равный 26 мб.

Решение.

Целевая функция (4) принимает вид:

$$T = c + f \cdot n_i \cdot x_i + \frac{d \cdot n_i}{x_i}$$

$$T' = f \cdot n_i - \frac{d \cdot n_i}{{x_i}^2}$$

$$x_i = \sqrt{\frac{d}{f}}$$

$$x_i = \sqrt{\frac{0.106}{6.8 \times 10^{-3}}} = 3.9$$

Определим, является ли экстремум функции ее минимумом или максимумом. Найдем вторую производную функции, если $T'' > 0$, то функция имеет минимум, если $T'' < 0$, то – максимум:

$$T'' = \frac{2dn}{x^3} = 0.086 > 0$$

Следовательно, функция T имеет минимум в точке x_i.

$$T(4) = 0{,}112 + 6{,}8 \times 10^{-3} \cdot 26 \cdot 4 + \frac{0{,}106 \cdot 26}{4} = 1{,}51 \text{ с.}$$

Время работы: $T(4) = 1{,}51$ с.

Таким образом, для данного процессора получено следующее оптимальное количество потоков x = 4.

Следовательно, для обработки данного файла на операционной системе Windows следует использовать полученное количество потоков.

6. Исследование производительности

Программный комплекс представляет собой консольную программу, интерфейс которой приведен на рис. 5.

В начале работы на вход программы подается файл с данными, который необходимо зашифровать, затем производится тест производительности, в ходе которого программа выдает значения переменных c, f, d. После чего происходит шифрование.

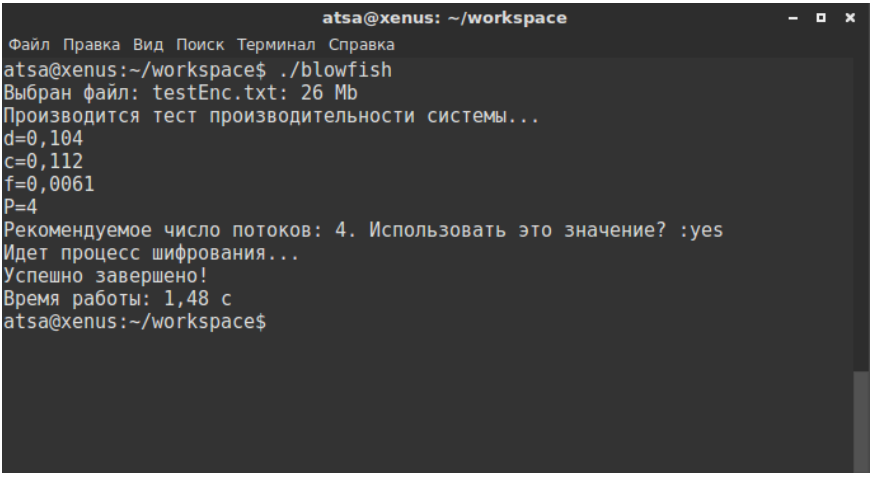


Рис. 5. Работа программы

Ниже приведены результаты экспериментов, проведенных на операционных системах Windows и Linux на одной и той же вычислительной системе с оптимальным числом потоков равным 4.

7. Результаты экспериментов

Таблица 2

ОС/Размер файла (мб)	0,512	1,024	2,048	4,096	5,12	6,34	7,956	10,512
Windows	0,175	0,1844	0,2022	0,238	0,2564	0,3103	0,4213	0,6023
Linux	0,15	0,1675	0,1801	0,2153	0,2383	0,2753	0,3411	0,46

8. Заключение

В ходе данной работы разработан программный комплекс, в котором был реализован параллельный алгоритм шифрования Blowfish..

Проведено исследование производительности реализации параллельного алгоритма шифрования в вычислительных системах под

управлением Windows и Linux. Исследование показало значительный прирост производительности реализованного алгоритма в системе под управлением операционной системы Linux. Это обусловлено особенностями архитектуры указанной операционной системы.

Литература

1. *Джеффри Рихтер*. Программирование на платформе Microsoft .NET Framework. Питер, Изд-ва: Русская Редакция, 2005.
2. Bruce Schneuer. Description of a New Variable-Length-Key, 64-Bit Block Cipher (Blowfish), in Anderson, Ross (Ed.): *Fast Soft-ware Encription – Cambridge Security Workshop, Proceedings*, Springer Verlag, Berlin, Heidelberg, New, York, 1994.
3. *Баричев С. Г., Серов Р. Е.* Основы современной криптографии. М.: Изд-во Горячая линия-Телеком, 2002. 122 с.
4. *Menezes A. J., Oorschot P. v., Vanstone S. A.* Handbook of Applied Cryptography, 1996. 816 p.
5. *Шнайер Б.* Прикладная криптография. М.: Изд-во Триумф, 2002. 816 с.
6. *Мирошников А. С., Кожиев В. С.* Эффективность использования технологии cuda для параллельного алгоритма шифрования ГОСТ 28147-89. Владикавказ, 2016.
7. *Мирошников А. С.* Система управления параллельной обработки данных в локальных вычислительных сетях. Дисс…. канд. техн. наук. Владикавказ, 2000.
8. *Groppen V. O.* Smart computing. 2004. 103 p.

Сведения об авторах

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

Датиев А.А.,
студент магистратуры СКГМИ (ГТУ)
e-mail: atsamaz.datieff@yandex,ru

Содержание

Научное издание

IT-ТЕХНОЛОГИИ: ТЕОРИЯ И ПРАКТИКА

Материалы семинара

Подписано в печать 15.06.2017. Формат 60x84 $^{1}/_{16}$. Бумага "Снегурочка". Гарнитура «Таймс». Печать на ризографе. Усл. п.л. 6,16. Уч.-изд. л. 3,3. Тираж 35 экз. Заказ № .

Федеральное государственное бюджетное образовательное учреждение высшего образования "Северо-Кавказский горно-металлургический институт (государственный технологический университет)"

Отпечатано в отделе оперативной полиграфии СКГМИ (ГТУ).
362021, Владикавказ, ул. Николаева, 44