УДК. 519.711

С.В. ЛИСТРОВОЙ, С.Е. ЛАВРИК

Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Украина

ОБЩИЙ ПОДХОД К РЕШЕНИЮ ЗАДАЧ БУЛЕВОГО ПРОГРАММИРОВАНИЯ

На основе идей рангового подхода предложен метод позволяющий решать задачи линейного булевого и нелинейного булевого программирования с единых позиций. Что достигнуто, благодаря использованию представления пространства решений, в виде симметричного графа и разработки процедуры отсечения неперспективных вариантов с использованием принципа оптимизации по направлению. В работе показано, что данный подход позволяет получить эффективные приближенные алгоритмы решения данной задачи, имеющие полиномиальную временную сложность. При этом погрешность решений, в случае равномерного закона распределения коэффициентов в функционале и ограничениях при числе ограничений больше шестидесяти не превышает 2-6%. Показано, что с увеличением размерности решаемой задачи и числа ограничений в ней погрешность в среднем у предложенных алгоритмов асимптотически уменьшается.

ранговый подход, булевое программирование, алгоритмы полиномиальной сложности

Введение

Задачи булевого линейного и нелинейного программирования являются моделями широкого класса прикладных задач в теории построения сложных систем и при этом задачи булевого линейного программирования относятся к классу NP-полных трудно решаемых задач [1], а эффективные методы решения задач нелинейного булевого программирования с произвольными нелинейностями практически отсутствуют. В настоящее время для каждого типа задачи разрабатывается свой метод решения. Представляется актуальным выработать единый подход к решению данного класса задач, обеспечивающий их решение с требуемой оперативностью и точностью. Рассмотрим подход, позволяющий решать задачи булевого линейного программирования и нелинейного программирования с произвольными нелинейностями одним и тем же алгоритмом, основанным на идеях рангового подхода [2 – 5].

Формализация и постановка задачи

Для описания всего множества булевых функций с множеством переменных $\{X_1, X_2 \ldots X_n\}$ введем понятие мета булевой функции $F(x)$ равной

$$F(x) = \sum_{j=1}^{p_1} C_{1j} S_1(C_n^1) + \sum_{j=1}^{p_2} C_{2j} S_2(C_n^2) + \ldots + \sum_{j=1}^{p_k} C_{kj} S_k(C_n^k) + \ldots + \sum_{j=1}^{p_n} C_{nj} S_n(C_n^n), \qquad (1)$$

где $Sr(C_n^r) = S_1 + S_2 + \ldots + S_{p_r}$ – сумма всех возможных сочетаний произведений переменных, содержащих в каждом произведении $Sr = X_p X_к \ldots X_m$ r различных переменных; $p_r = \dfrac{n!}{r!(n-r)!}$; C_{rj} – коэффициенты, стоящие в произведениях Sr содержащих r переменных.

Обозначим через H множество всех булевых функций, которое можно породить на основе $F(x)$, полагая равными нулю различные сочетания C_{ij} в (1). Множество H является полным в том смысле, что содержит в себе все возможные булевы функции, которые можно вообще построить на основе данного подмножества переменных $\{X_1, X_2 \ldots X_n\}$. Не трудно показать, что мощность данного множества очень велика, но конечна и равна 2^{p_Σ}, где

$$p_\Sigma = 1 + \frac{1}{2}\left[\frac{n!}{1!(n-1)!}\left(\frac{n!}{1!(n-1)!}+1\right)\right] + \frac{n!}{2!(n-2)!}\left(\frac{n!}{2!(n-2)!}+1\right) + \ldots + \frac{n!}{k!(n-k)!} \times \left(\frac{n!}{k!(n-k)!}+1\right) + \ldots + \frac{n!}{(n-1)!1!}\left(\frac{n!}{(n-1)!1!}+1\right)\right].$$

Следует отметить, что с помощью соотношения (1) может быть определен класс задач дискретной оптимизации, в которых решение определяется только сочетанием переменных и не зависит от перестановки переменных в $S_r(C_n^r)$. То есть значение C_{lj} в этих задачах зависит только от сочетания переменных в $S_r(C_n^r)$.

В общем случае задачу булевого программирования можно представить в виде

$$f(X_1, X_2, ...X_n) \Rightarrow \max$$
$$g_j(X_1, X_2, ...X_n) \le b_j; \quad j = (\overline{1,m}), \qquad (2)$$

где $f(X_1, X_2, ...X_n) \in H$, $g_j(X_1, X_2, ...X_n) \in H$.

Рассмотрим граф $G(X, E)$, рис. 1, в котором вершины X_i и X_j соединены ребром (i, j), если они могут быть объединены в клику. В графе $G(X, E)$ каждой вершине X_i соответствует переменная X_i.

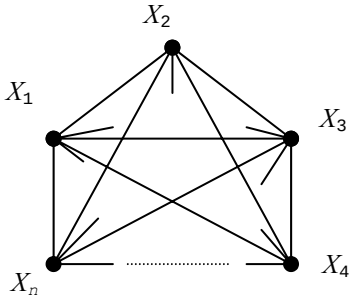


Рис. 1. Граф G

Выделим в графе G произвольную клику $Q = X_p X_r ... X_m$, состоящую из r вершин, где $r < n$, и рассмотрим ее пересечения с

$$S_r(C_n^r) \in f(X_1, X_2, ...X_n),$$

а также с $S_r(C_n^r) \in g_j(X_1, X_2, ...X_n)$. Каждое пересечение можно охарактеризовать суммами коэффициентов C_{lj}, стоящими при $S_r(C_n^r)$ в функционале $f(X_1, X_2, ...X_n)$ и ограничениях $g_j(X_1, X_2, ...X_n)$. При этом в общем случае произвольная клика Q всегда будет характеризоваться соответствующим весом по функционалу $f(X_1, X_2, ...X_n)$ и не более чем m весами по ограничениям $g_j(X_1, X_2, ...X_n)$. Таким образом, произвольная задача булевого программирования может рассматриваться как задача нахождения клики Q^* максимального веса по весам функционала, в графе G, у которой все m весов по весам ограничений не превышают соответственно b_j. Так, например, если решается задача линейного программирования:

$$f(x) = C_1X_1 + C_2X_2 + C_3X_3 + C_4X_4 \to \max; \qquad (3)$$
$$B_1X_1 + B_2X_2 + B_3X_3 + B_4X_4 \le b_1;$$
$$K_1X_1 + K_2X_2 + K_3X_3 + K_4X_4 \le b_2,$$

то граф G будет иметь вид (рис. 2):

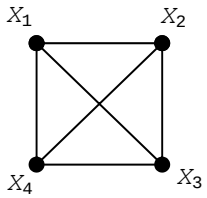


Рис. 2. Граф G для задачи линейного программирования

Таблица 1

Задача линейного программирования

X_1	C_1	B_1	K_1
X_2	C_2	B_2	K_2
X_3	C_3	B_3	K_3
X_4	C_4	B_4	K_4

Как видно из табл. 1, каждая вершина графа G характеризуется тремя весами (C_i, B_i, K_i), и для решения задачи в графе G (рис. 2) следует найти такую клику Q^* из взвешенных вершин, чтобы ее суммарный вес по весам $\{C_i\}$ был максимален, а суммарные веса по весам $\{B_i\}$ и $\{K_i\}$ не превышали соответственно величин b_1 и b_2. Следует отметить, что в задачах линейного программирования и с нелинейностью выше второй (т.е. при наличии в функции цели или ограничениях произведений состоящих из числа переменных более двух) весовые характеристики ребер графа G полагаются равными нулю.

В случае решения задач квадратичного программирования удобно вводить и весовые характеристики ребер.

Так, рассмотрим задачу квадратичного программирования следующего вида.

$$f(x) = C_1X_1 + C_2X_2 + C_3X_3 + C_4X_4 +$$
$$+ C_{12}X_1X_2 + C_{13}X_1X_3 + C_{34}X_1X_4 + C_{23}X_2X_3 +$$
$$+ C_{24}X_2X_4 + C_{34}X_3X_4 \to \max; \quad (4)$$
$$B_1X_1 + B_2X_2 + B_3X_3 + B_4X_4 \le b_1;$$
$$K_1X_1 + K_2X_2 + K_3X_3 + K_4X_4 \le b_2.$$

Для нее можно поставить в соответствие следующий граф (рис. 3);

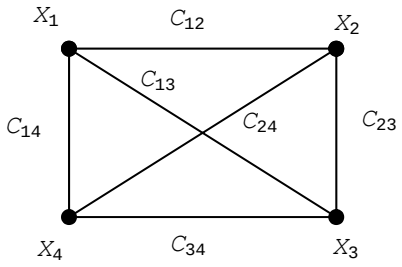


Рис. 3. Граф *G* для задачи квадратичного программирования

Таблица 2

Задача квадратичного программирования

X_1	C_1	B_1	K_1
X_2	C_2	B_2	K_2
X_3	C_3	B_3	K_3
X_4	C_4	B_4	K_4

Как видно из рис. 3 и табл. 2, вершинам графа, как и в задаче линейного программирования, соответствуют веса (C_i, B_i, K_i), а ребрам – веса $\{C_{ij}\}$. Для решения данной задачи в графе *G* требуется найти клику Q^* максимального суммарного веса по C_i и C_{ij} и при этом суммарные веса по весам $\{B_i\}$ и $\{K_i\}$ не превышали соответственно величин b_1 и b_2. В этом случае граф *G* является взвешенным и по вершинам и по ребрам, аналогично взвешенными являются и все клики данного графа. Следует иметь в виду, что в квадратичной задаче $C_{ij}=C_{ji}$.

Таким образом, для решения произвольной задачи булевого программирования необходимо построить алгоритм, позволяющий находить в заданном графе *G,* взвешенном по вершинам, или и по вершинам и по ребрам, клику Q^* максимального суммарного веса по весам функционала $f(X_1, X_2, ...X_n)$, у которой все *m* весов по весам ограничений $g_j(X_1, X_2, ...X_n)$ не превышают соответствующих b_j.

Решение задачи

Для решения задачи воспользуемся представлением исходного графа *G* в виде симметричного дерева путей, предложенного в работах [6 – 8]. Смысл такого представления заключается в следующем. Пусть все возможные состояния некоторой системы определяется графом $G(V, E)$ с *n* вершинами, где вершины соответствуют возможным состояниям системы. Перейдем к пространству с $(n–1)^2$ состояниями. Для этого каждому из *n* состояний поставим в соответствие еще $(n–1)$ состояние, характеризующее способ достижения состояния из множества $\{1, 2, ….. n\}$. При этом в качестве добавляемых состояний определим ранг пути в графе $G(V, E)$. Т.е. из вершины *s* графа $G(V, E)$ в произвольную вершину *j* можно попасть путем ранга $r=1$, используя одно ребро, путем ранга $r=2$, используя 2 ребра и т. д. путем ранга $r=n–1$, используя $n–1$ ребро. Такое пространство состояний можно представить в виде стянутого дерева путей *D*, графически оно может быть изображено так, как это показано на рис. 4.

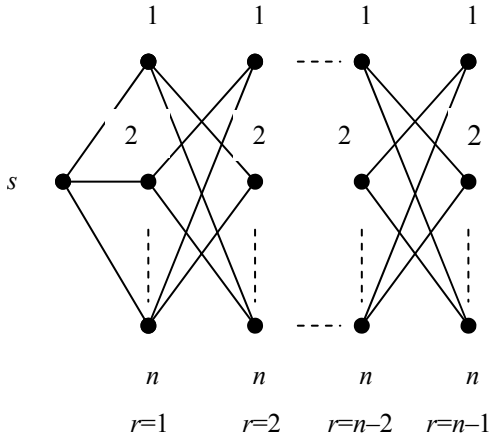


Рис. 4. Стянутое дерево всех путей D графа G(V, E)

Дерево всех путей D содержит $(n–1)$ горизонтальную линейку и $(n–1)$ ярус. Для прочтения путей на каждой горизонтальной линейке можно бывать только один раз. Исходя из стянутого дерева путей, для произвольной вершины *j* множество путей, ведущих в эту вершину из некоторой вершины *s* можно представить в следующем виде:

$$m_s(j) = m_{sj}^{r=1} \cup m_{sj}^{r=2} \cup .... \cup m_{sj}^{r=n-1};\ j = (\overline{1, n-1}),\quad (5)$$

где $m_{sj}^r = \{\mu_{sj}^r\}$ подмножества путей из произвольной вершины s в некоторую вершину j графа $G(V, E)$, ранга r. Следует отметить, что дерево всех путей D может строиться и от конкретной вершины i графа, в этом случае вершина $s=i$ и i-я горизонтальная линейка исключается в D. Например, при $i=2$ дерево D будет иметь вид рис. 5. В дальнейшем стянутое дерево путей, приведенное на рис. 4, будет использоваться для построения однопроходных алгоритмов решения задачи (2), а дерево D (рис. 5) – для построения n-проходных алгоритмов, суть построения которых будет пояснена ниже.

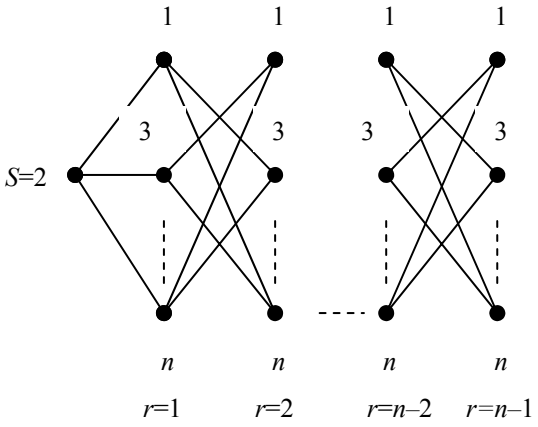


Рис. 5. Стянутое дерево всех путей D графа от вершины $s=2$ $G(V, E)$

Таким образом, используя граф D и введя правила формирования путей следующего ранга мы можем из произвольной вершины s поэтапно строить пути $\{\mu_{sj}^r\}$ произвольного ранга в плоть до ранга $r=n-1$. В нашей задаче под состоянием системы мы будем подразумевать различные способы объединения вершин графа D в клики. Тогда каждому пути $\{\mu_{sj}^r\}$ ранга r в графе D, проходящем через вершины (v_h, v_k… v_p) в исходном графе G решаемой задачи соответствует клика из $r-1$ вершины($X_h X_k ... Xp$), характеризующаяся соответствующим весом по функционалу $f(X_1, X_2, ...X_n)$ и не более чем m весами по ограничениям $g_j(X_1, X_2, ...X_n)$. Весовые характеристики $\{d_{sj}^{r-1}\}$ произвольной клики $Q^{r-1}(j)$, состоящей из $(r-1)$ вершины и определяемой одним из путей $\mu_{sj}^r \in m_{sj}^r$ ранга r в графе D вычисляются по весам функционала. Вычисление производится путем суммирования коэффициентов подмножества $L_f=\{C_{rj}\}$, стоящих при $S_{r-1}(C_n^{r-1}) \in P_f$, где P_f все подмножества $\{S_{r-1}(C_n^{r-1})\}_f$, удовлетворяющие условию $S_{r-1}(C_n^{r-1})_f \cap Q^{r-1}(j) \neq \varnothing$, а $S_{r-1}(C_n^{r-1})_f$ определяется функционалом $f(X_1, X_2, ...X_n)$. Аналогично определяются весовые характеристики по весам ограничений путем суммирования коэффициентов подмножества $L_B=\{C_{rj}\}$, стоящих при $S_{r-1}(C_n^{r-1}) \in P_B$, где P_B – все подмножества $\{S_{r-1}(C_n^{r-1})\}_B$, удовлетворяющие условию $S_{r-1}(C_n^{r-1})_B \cap Q^{r-1}(j) \neq \varnothing$, а $S_{r-1}(C_n^{r-1})$ определяется ограничениями $g_j(X_1, X_2, ...X_n)$; $j = (\overline{1, m})$. Таким образом, весовые характеристики клик $Q^{r-1}(j)$, характеризующихся множеством путей m_{sj}^r по весам функционала и ограничений определяются соответственно равенствами

$$d_{sj}{}^{f(r-1)} = \sum_{C_{rj} \in L_f} C_{rj};\quad d_{sj}{}^{B(r-1)} = \sum_{C_{rj} \in L_B} C_{rj}. \quad (6)$$

Следует отметить, что в графе D каждый путь имеет в общем случае $m+1$ длину, одну по весам функционала и m по весам ограничений, и для решения поставленной задачи нам в графе D нужно построить путь максимальной длины по весам функционала от вершин 1, 2, … n ко всем остальным вершинам графа, при этом его длины по весам ограничений не должны превышать соответствующей величины b_j. Если на основе подмножеств путей $m_{sj}^{r=1}$ в графе D строить подмножества $m_{sj}^{r=2}$ и так далее до $m_{sj}^{r=n-1}$, то мы вынуждены будем построить $(n-1)!$ путей, поэтому для формирования

путей вводится процедура A, позволяющая отсекать неперспективные пути. Для отсечения неперспективных вариантов в процедуре A предлагается использовать принцип оптимизации по направлению к произвольной вершине p, при формировании путей следующего ранга m_{sp}^{r+1} на основе путей предыдущего ранга m_{sj}^{r}, который для рассматриваемой задачи определяется следующим рекуррентным соотношением

$$\mu_{sp}^{r+1} = \max_{j}\{\{\mu_{sj}^{r}\} \cup (j,p)\};\ j=\overline{(1,n)};\ p=\overline{(1,n)};\ j \neq p, \quad (7)$$

где (j, p) – ребро графа D; n – число различных вершин в графе D.

Перед началом работы процедуры A_1 переменной i:=1. Рассмотрим возможность построения n-проходных и однопроходных процедур решения задачи (2) соответственно на стянутых деревьях, приведенных на рис. 4, 5.

Перед началом работы процедуры A_1 переменной i:=1.

Процедура A_1 с n проходами.

Шаг 1. Переменной s:=i и из вершины s строятся все возможные пути ранга r=1 ко всем вершинам графа D (рис. 2), удовлетворяющие ограничениям $g_j(X_1, X_2, ...X_n)$; $j=\overline{(1,m)}$, при этом длины по весам функционала и ограничениями вычисляются в соответствии с соотношениями (6).

Шаг 2. Используя пути текущего ранга r, строятся все возможные пути ранга r:=r+1 удовлетворяющие ограничениям $g_j(X_1, X_2, ...X_n)$; $j=\overline{(1,m)}$ с использованием рекуррентного соотношения (7). При этом проверка ограничений и выбор пути максимального по весам функционала осуществляется на основе вычислений длин путей по весам функционала и ограничений в соответствии с соотношениями (6). (Следует отметить, что если в процессе применения рекуррентного соотношения (7) возникают несколько путей одинаковой длины, то необходимо их все продлевать на следующем ранге).

Шаг 3. Проверяем $m_{sj}^{r+1}=\varnothing$, если истина, то путь μ_{sj}^{*r} максимальной длины, полученный на ранге r, является локальным экстремумом решаемой задачи, иначе выполняем следующий шаг.

Шаг 4. Проверяем i=n–1, если ложь, то i:=i+1 и переходим к шагу 1, иначе процедура A_1 заканчивает работу, при этом из множества локальных экстремумов $\{\mu_{sj}^{*r}\}$ выбирается глобальный μ_{sj}^{**r}, соответствующий оптимальному решению задачи (2).

Для снижения временной сложности работы алгоритма возможно использовать однопроходный вариант реализации данной процедуры на основе стянутого древа путей, приведенного на рис. 1. При этом однопроходная процедура A_2 имеет вид

Процедура A_2 с 1 проходом.

Шаг 1. Из вершины s строятся все возможные пути ранга r=1 ко всем вершинам графа D (рис. 2), удовлетворяющие ограничениям $g_j(X_1, X_2, ...X_n)$; $j=\overline{(1,m)}$, при этом длины по весам функционала и ограничениям вычисляются в соответствии с соотношениями (6).

Шаг 2. Используя пути текущего ранга r, строятся все возможные пути ранга r:=r+1, удовлетворяющие ограничениям $g_j(X_1, X_2, ...X_n)$; $j=\overline{(1,m)}$ с использованием рекуррентного соотношения (7). При этом проверка ограничений и выбор пути максимального по весам функционала осуществляется на основе вычислений длин путей по весам функционала и ограничений в соответствии с соотношениями (6). (Следует отметить, что если в процессе применения рекуррентного соотношения (7) возникают несколько путей одинаковой длины, то необходимо их все продлевать на следующем ранге).

Шаг 3. Проверяем $m_{sj}^{r+1}=\varnothing$, если истина, то μ_{sj}^{*r} – путь максимальной длины, полученный на ранге r, является локальным экстремумом решаемой задачи, иначе выполняем следующий шаг.

Шаг 4. Проверяем ранг $r = n$, если ложь, то переходим к выполнению шага 2, иначе процедура A_2 заканчивает работу, и путь μ_{sj}^{*r} максимальной длины, полученный на ранге $r = n$ соответствует оптимальному решению задачи (2).

Еще одним вариантом уменьшения временной сложности алгоритмов на основе процедур A_1 и A_2 могут являться процедуры $A^{/}$ и $A^{//}$, отличающиеся от A_1 и A_2 тем, что на каждом ярусе формирования путей на основе процедур A_1 и A_2 локальные экстремумы будут выделяются не в каждом множестве, а выделяются глобальные экстремумы на ярусе и на основе пути соответствующего глобального экстремума на ярусе формируются пути следующего яруса, удовлетворяющие ограничениям. При этом процедуры $A^{/}$ и $A^{//}$ будут иметь следующий вид.

Процедура $A^{/}$. Перед началом работы процедуры $A^{/}$ переменной i:=1.

Шаг 1. Переменной s:=i и из вершины s строятся все возможные пути ранга r=1 ко всем вершинам графа D (рис. 2), удовлетворяющие ограничениям $g_j(X_1, X_2, ...X_n)$; $j = \overline{(1,m)}$, при этом длины по весам функционала и ограничениями вычисляются в соответствии с соотношениями (6). Далее выделяется самый длинный путь на ярусе.

Шаг 2. Используя самый длинный путь текущего ранга r, построенный на предыдущем шаге, строятся все возможные пути ранга r:=r+1 удовлетворяющие ограничениям $g_j(X_1, X_2, ...X_n)$; $j = \overline{(1,m)}$ с использованием рекуррентного соотношения (7). При этом проверка ограничений и выбор пути максимального по весам функционала осуществляется на основе вычислений длин путей по весам функционала и ограничений в соответствии с соотношениями (6).

Шаг 3. Проверяем $m_{sj}^{r+1} = \varnothing$, если истина, то путь μ_{sj}^{*r} максимальной длины, полученный на ранге r, является локальным экстремумом решаемой задачи, иначе выполняем следующий шаг.

Шаг 4. Проверяем i=n–1, если ложь, то i:=i+1 и переходим к выполнению шага 1, иначе процедура $A^{/}$ заканчивает работу, при этом из множества локальных экстремумов(полученных за один проход) $\{\mu_{sj}^{*r}\}$ выбирается глобальный μ_{sj}^{**r}, соответствующий оптимальному решению задачи (2).

Процедура $A^{//}$.

Шаг 1. Из вершины s строятся все возможные пути ранга r=1 ко всем вершинам графа D (рис. 2), удовлетворяющие ограничениям $g_j(X_1, X_2, ...X_n)$; $j = \overline{(1,m)}$, при этом длины по весам функционала и ограничениями вычисляются в соответствии с соотношениями (6). Далее выделяется самый длинный путь на ярусе.

Шаг 2. Используя самый длинный путь текущего ранга r, построенный на предыдущем шаге, строятся все возможные пути ранга r:=r+1, удовлетворяющие ограничениям $g_j(X_1, X_2, ...X_n)$; $j = \overline{(1,m)}$ с использованием рекуррентного соотношения (7). При этом проверка ограничений и выбор пути максимального по весам функционала осуществляется на основе вычислений длин путей по весам функционала и ограничений в соответствии с соотношениями (6).

Шаг 3. Проверяем $m_{sj}^{r+1} = \varnothing$, если истина, то путь μ_{sj}^{*r} максимальной длины, полученный на ранге r, является локальным экстремумом решаемой задачи, иначе выполняем следующий шаг.

Шаг 4. Проверяем ранг $r = n$, если ложь, то переходим к выполнению шага 2, иначе процедура $A^{//}$ заканчивает работу, и путь μ_{sj}^{*r} максимальной длины, полученный на ранге $r = n$ соответствует оптимальному решению задачи (2).

Экспериментальное исследование алгоритмов на основе разработанных процедур. Исследовались следующие алгоритмы: алгоритм A_5 на основе многопроходной процедуры A_1, алгоритм A_4 на основе однопроходной процедуры A_2 и алгоритм A_3 на

основе однопроходной процедуры $A^{//}$. При исследовании коэффициенты в функционале и ограничениях генерировались по равномерному закону распределения в функционале в диапазоне от 0 до 20, а в ограничениях от 0 до 10. На каждую точку при оценке временной сложности алгоритмов в среднем и погрешности алгоритмов решалось не менее 50 тестовых задач, результаты получены с доверительной вероятностью 0,95. В качестве точного алгоритма использовался разработанный авторами алгоритм для задачи квадратичного и линейного программирования, скомбинированного на основе использования для прогнозирования идеи рангового подхода, а для отсева неперспективных путей – метод ветвей и границ. Данный комбинированный алгоритм позволил снять погрешности для задач до размерности не превышающей n=70. Графики зависимости погрешности от размерности (n) решаемых задач и от числа ограничений (m) в задаче (2) приведены на рис. 6 – 8. Из рисунков видно, что погрешность алгоритмов с увеличением числа ограничений m асимптотически уменьшается, с увеличением n возрастает, а при $m \geq 50$ погрешность алгоритмов стабилизируется. Также видно, что погрешность для задач линейного программирования не превышает 2%, а для задач квадратичного 5 – 10%. Решение тестовых задач показало, что увеличение диапазона изменения коэффициентов в функционале и ограничениях приводит к резкому снижению погрешностей алгоритмов, переход от нелинейностей одного порядка к нелинейностям более высокого порядка может приводить к незначительному возрастанию погрешностей при небольшом числе ограничений, но с возрастанием числа ограничений возрастание погрешности очень быстро компенсируется.

Оценка сложности процедур {*A*}. Поскольку число путей, строящихся на произвольном ранге r, не может превысить (n–1)* (n–1), максимальный ранг r произвольного пути не превышает (n–1), а число циклов, выполняемое процедурой A_1, равно n, то после n циклов число путей, которое построит процедура A_1, не может превзойти $(n-1)*(n-1)*(n-1)*n \approx n^4$, а число обработанных векторов n^5. С учетом число слагаемых (k) в функционале и числа ограничений (m) временная сложность алгоритма не превысит в худшем случае $O(n^5 k(m+1))$. В случае, когда решение задачи осуществляется за один проход процедуры A_2 или за один проход процедуры $A^{//}$, но с выделением наиболее длинного пути на ярусе сложность процедур A_2 и $A^{//}$ не превысят соответственно $O(n^4k(m+1))$ и $O(n^3k(m+1))$. Таким образом, алгоритмы A_5, A_4, A_3 имеют соответственно временную сложность, не превышающую в худшем случае $O(n^5k(m+1))$, $O(n^4k(m+1))$ и $O(n^3k(m+1))$. Экспериментальное исследование временной сложности показало (рис. 9), что число обрабатываемых векторов не зависит от числа ограничений и в среднем для алгоритмов A_5, A_4, A_3 временная сложность не превышает соответственно $O(0, 1n^{4,9})$, $O(0, 3n^{3,7})$ и $O(0, 4n^{2,8})$.

Учет перестановок переменных. Мы рассматривали задачи булевого программирования, в которых коэффициенты C_{lj} в (1) не зависят от порядка следования переменных, а только от их сочетаний. Покажем на примере задачи о кратчайшем гамильтоновом пути, что на основе предложенной процедур {A} могут эффективно решаться и задачи булевого программирования, в которых длина пути в графе G зависит не только от сочетания переменных, но и от перестановок переменных.

Процедура А для определения кратчайшего гамильтонового пути в произвольном графе. Из вершины s строятся все возможные пути ранга r=1 ко всем вершинам графа D, далее, используя пути ранга r=1, строятся все возможные пути ранга r=2 и на их основе формируются пути следующего ранга, с использованием рекуррентного соотношения (8), и т. д. до построения путей ранга $r=n-1$.

$$\mu_{sp}^{r+1} = \min_{j}\{\{\mu_{sj}^{r}\} \cup (j,p)\};\ j = \overline{(1,n)};\ p = \overline{(1,n)};\ j \neq p \quad (8)$$

(следует отметить, что если в процессе применения рекуррентного соотношения (3) возникают несколько кратчайших путей одинаковой длины, то необходимо их все строить на следующем ранге).

Можно выделить следующие особенности работы процедуры *A*. В процессе ее работы может возникать две ситуации. Первая – когда процедура *A* на каждом шаге построила пути в множества m_{sj}^{r}, т.е. принцип оптимальности работы процедуры не нарушался, и вторая, когда к одной из вершин ни одного пути построить нельзя. Последнее обстоятельство возможно в двух случаях:

а) если анализируемый граф неполный и к вершине *p* не существует пути некоторого ранга *r*=*k*;

б) когда некоторая вершина *p* войдет во все пути предыдущего ранга.

Нетрудно видеть, что в первой ситуации процедура *A* не теряет оптимальное решение, а во второй ситуации (случай б) оптимальное решение может быть потеряно, поскольку принцип оптимальности работы процедуры нарушается. Это означает, что на основе процедуры *A* мы можем построить только приближенный алгоритм решения задачи. Однако после нарушения принципа оптимальности последующее продление путей с использованием рекуррентного соотношения (8) позволит минимально отклоняться от оптимального решения.

Алгоритм A_1 решения задачи определения кратчайших гамильтоновых путей в произвольных графах.

Шаг 1. Присваиваем значение *S*:=1 переменной *S* и в графе *D*, используя процедуру *A*, определяем кратчайшие пути ранга *r*=*n*–1от вершины *S* ко всем остальным вершинам графа и увеличиваем значение *S*:= *S*+1.

Шаг 2. Проверяем *S*=*n*, если нет, то переходим к выполнению шага 1, иначе выполняем следующий шаг.

Шаг 3. Среди всех построенных кратчайших путей ранга *r*=*n*–1 на шагах 1 и 2 выбираем самый короткий, и алгоритм заканчивает работу.

Для решения задачи за один проход определим точку входа, начиная с которой начнет работу алгоритм. Для этого введем понятие минимально удаленной вершины *i* в произвольном графе *G*(*V*, *E*).

Пусть задан граф *G*(*V*, *E*) с *n* вершинами и множеством ребер *E*, каждому из которых присвоен вес в виде произвольного положительного числа β_i. Поставим каждой вершине *s* графа *G*(*V*, *E*) в соответствие вектор X_s:

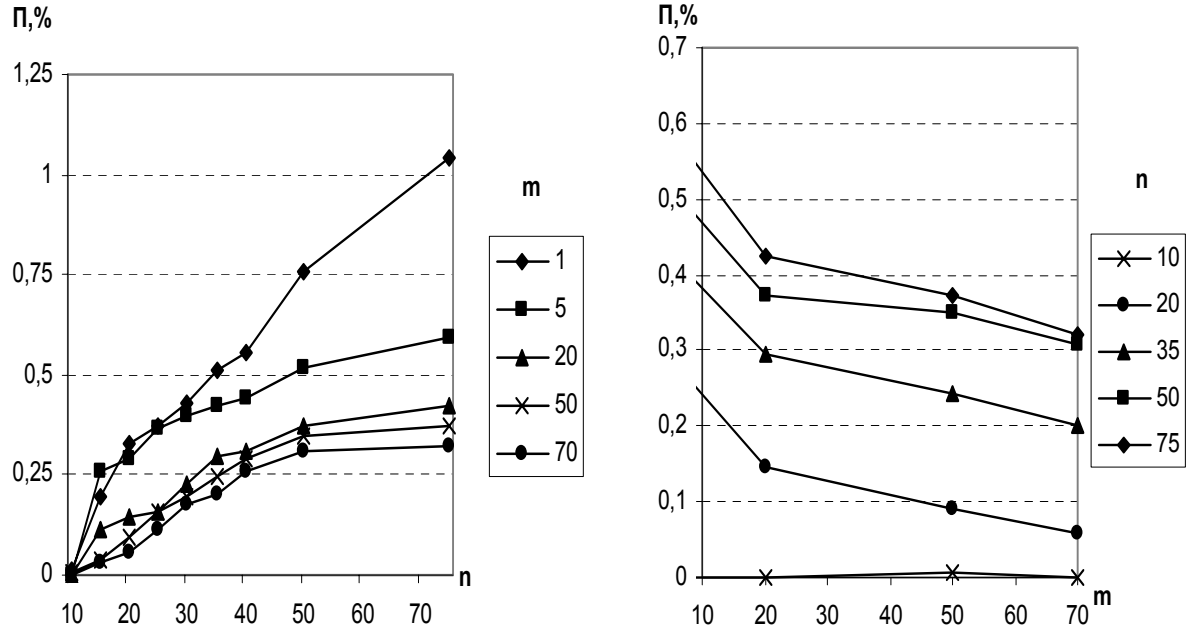
$$X_s = \{\beta_1, \beta_2, \ldots, \beta_s, \wp, \beta_{s+1}, \ldots, \beta_n\}; \quad s = \overline{(1,n)},$$

в котором на *s*-й позиции стоит символ $\wp$, означающий, что элемент β_s в X_s отсутствует. Множество $\{\beta_i\}$ представляет собой множество положительных чисел, к которым будем относить и 0**. Будем говорить, что некоторая вершина *i* минимально удалена от вершины *s*, определяемой вектором X_s, если расстояние *d*(*s*, *i*) между вершинами *s* и *I* определяется соотношением**

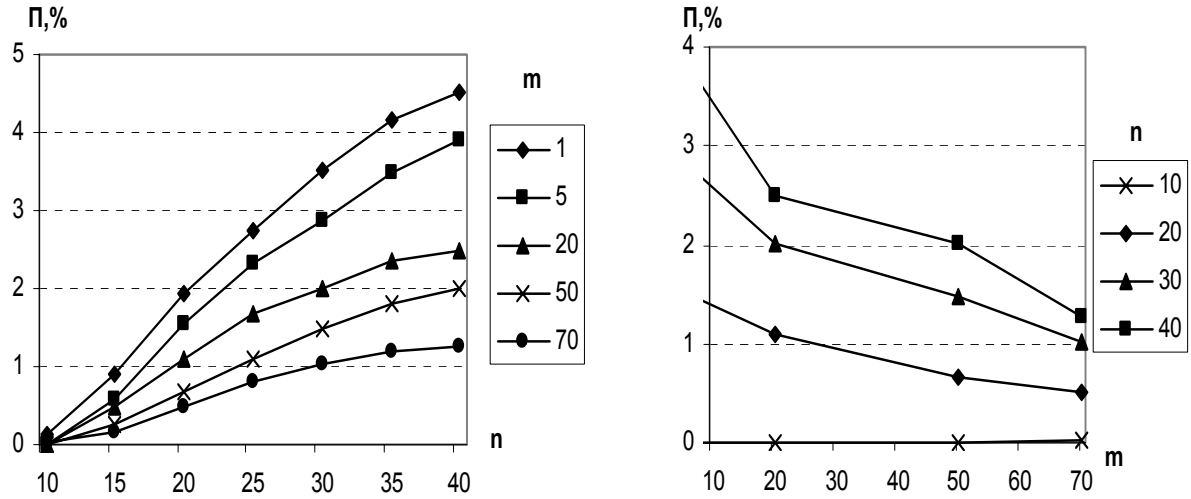
$$d(s,i) = \min_i \{\beta_i \in X_s\}. \qquad (9)$$

Если существует *Q* вершин *i*, удовлетворяющих соотношению (9), то это означает, что существует *Q* минимально равно удаленных вершин от вершины *s*, определяемой вектором X_s. Определив вершину *i* входа для графа можно решать задачу определения кратчайшего гамильтонового пути за один проход процедуры A_1, при этом погрешность по сравнению с *n*-проходной процедурой увеличивается не более чем на один процент, а сложность алгоритма понижается в *n*–1 раз. Алгоритм с использованием минимально удаленной точки обозначим A_2.

Экспериментальное исследование алгоритмов решения задачи определения кратчайших гамильтоновых путей. При экспериментальном сравнении разработанных алгоритмов с известными, веса ребер графа генерировались по равномерному закону распределения в диапазоне 0-50. Для получения среднего значения каждой точки графиков всех анализируемых характеристик решалось по 200 тестовых задач, все результаты статистического анализа получены с доверительной вероятностью 0,95.

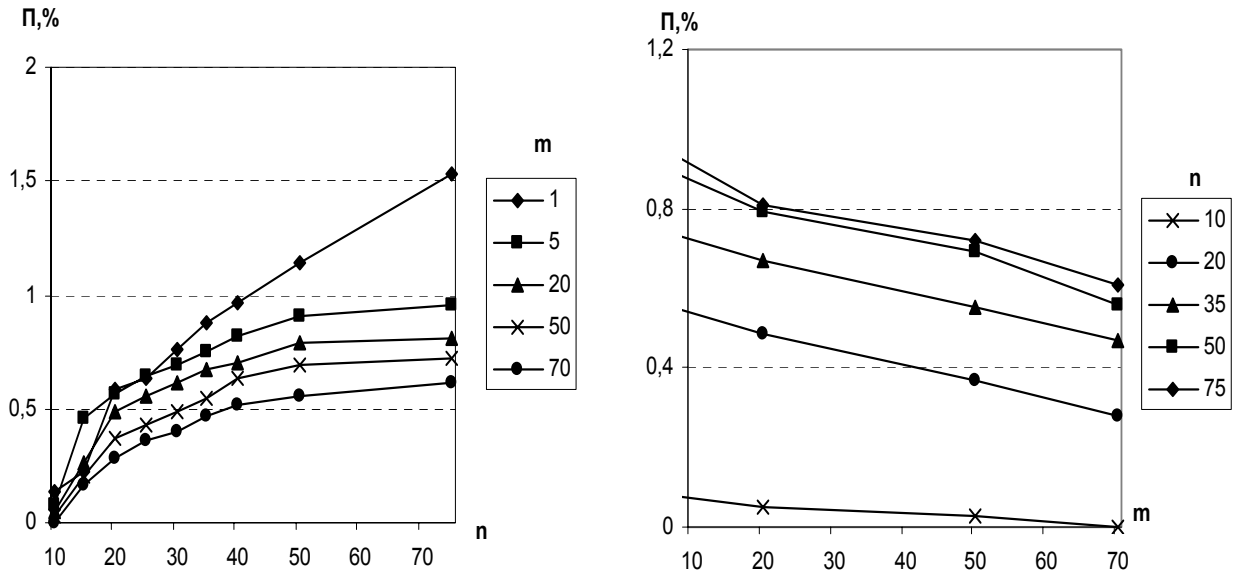


а

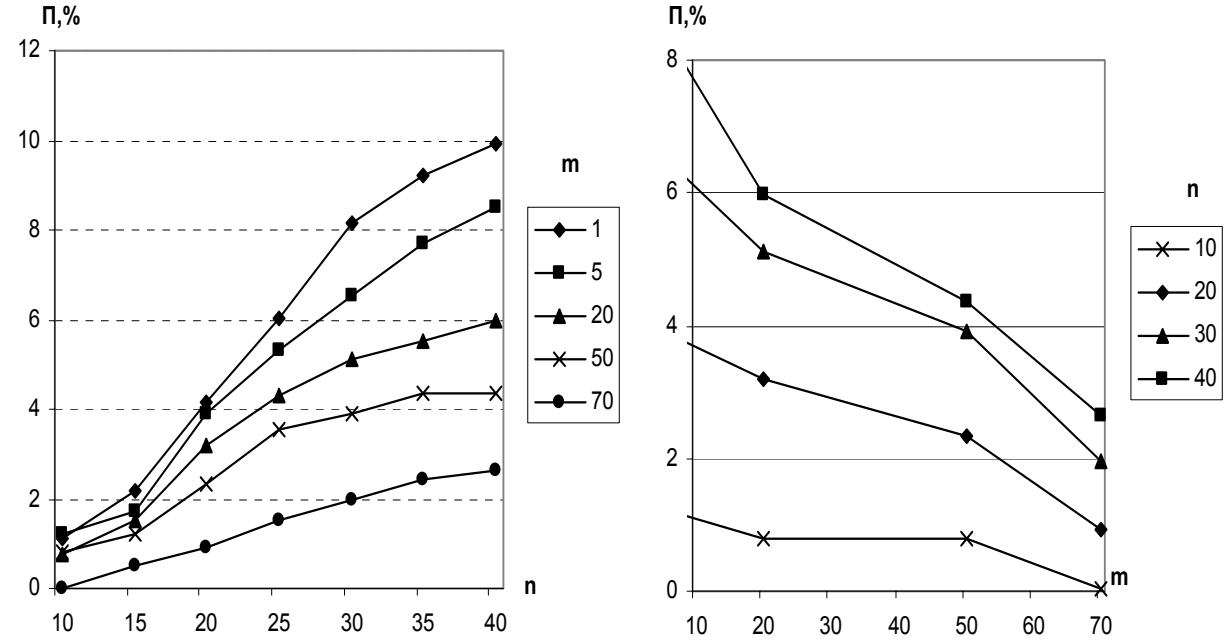


б

Рис. 6. Зависимость погрешности алгоритма A_5:
а – от размерности решаемой задачи линейного булевого программирования (n)
при различном числе ограничений (m);
б – от размерности решаемой задачи квадратичного булевого программирования (n)
при различном числе ограничений (m)

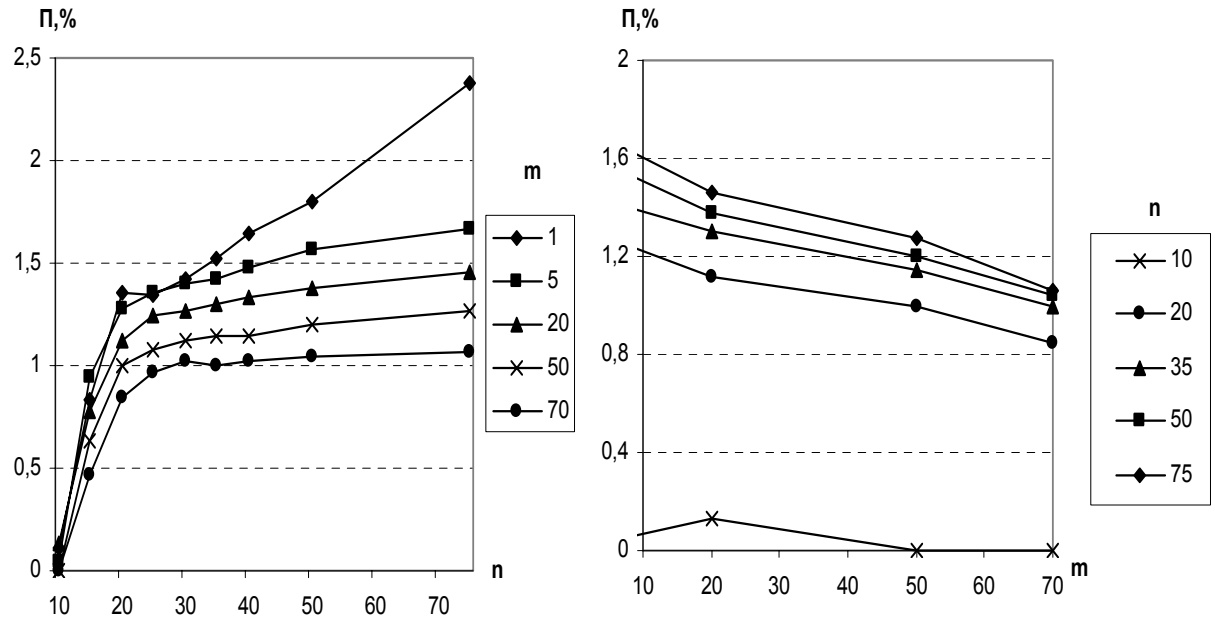


а

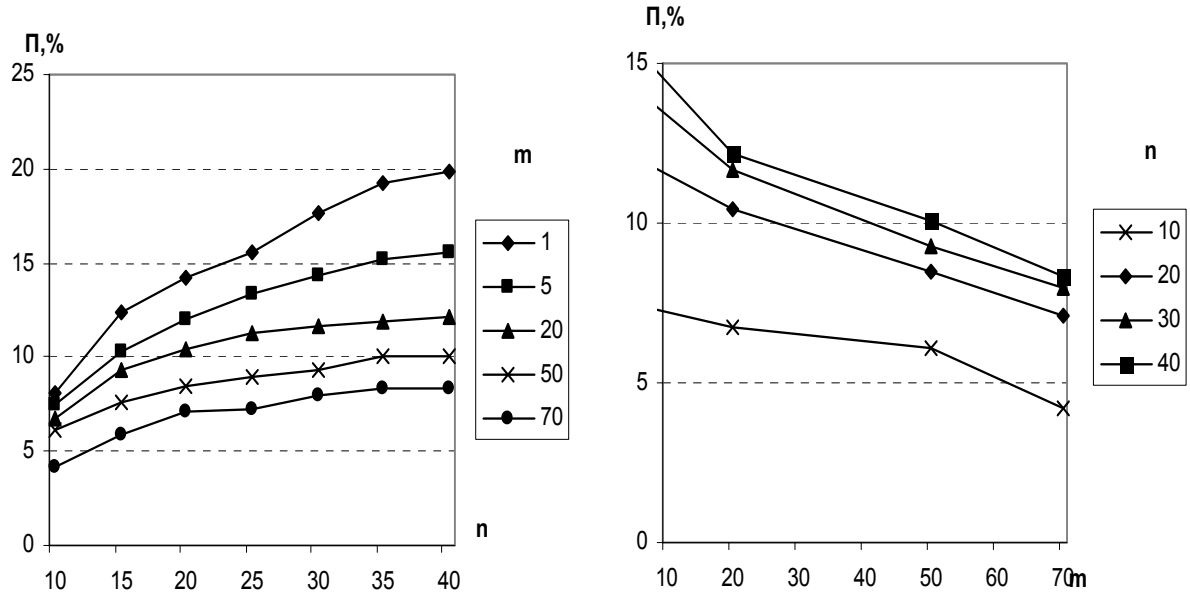


б

Рис. 7. Зависимость погрешности алгоритма A_4:
а – от размерности решаемой задачи линейного булевого программирования (n)
при различном числе ограничений (m);
б – от размерности решаемой задачи квадратичного булевого программирования (n)
при различном числе ограничений (m)



а



б

Рис. 8. Зависимость погрешности алгоритма A_3:
а – от размерности решаемой задачи квадратичного булевого программирования (n)
при различном числе ограничений(m);
б – от размерности решаемой задачи линейного булевого программирования (n)
при различном числе ограничений (m)

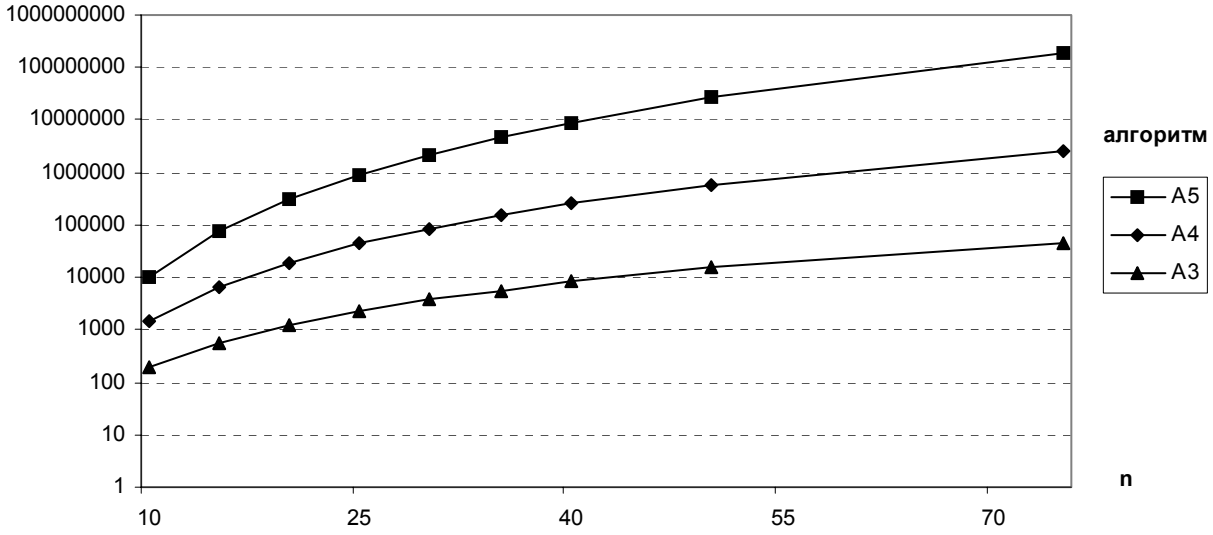


Рис. 9. Зависимость числа обрабатываемых векторов от размерности решаемой задачи линейного и квадратичного квадратично программирования для алгоритмов A_5, A_4, A_3

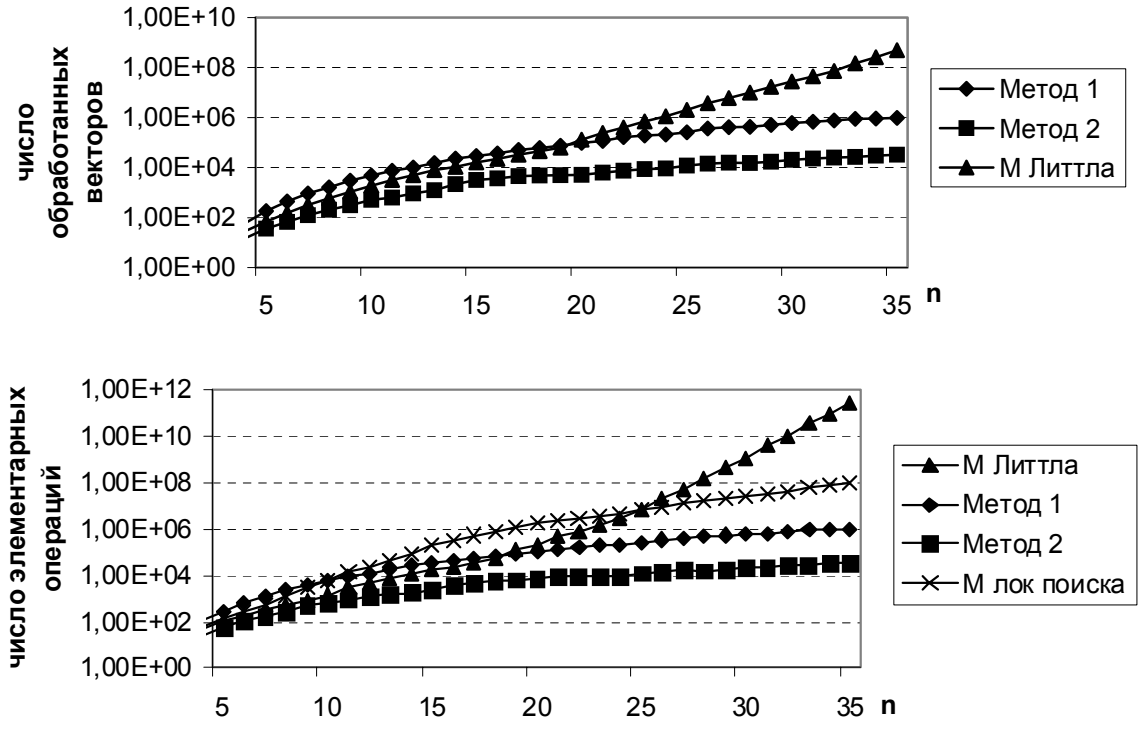


Рис. 10. Зависимости числа обрабатываемых векторов и элементарных операций от размерности решаемой задачи

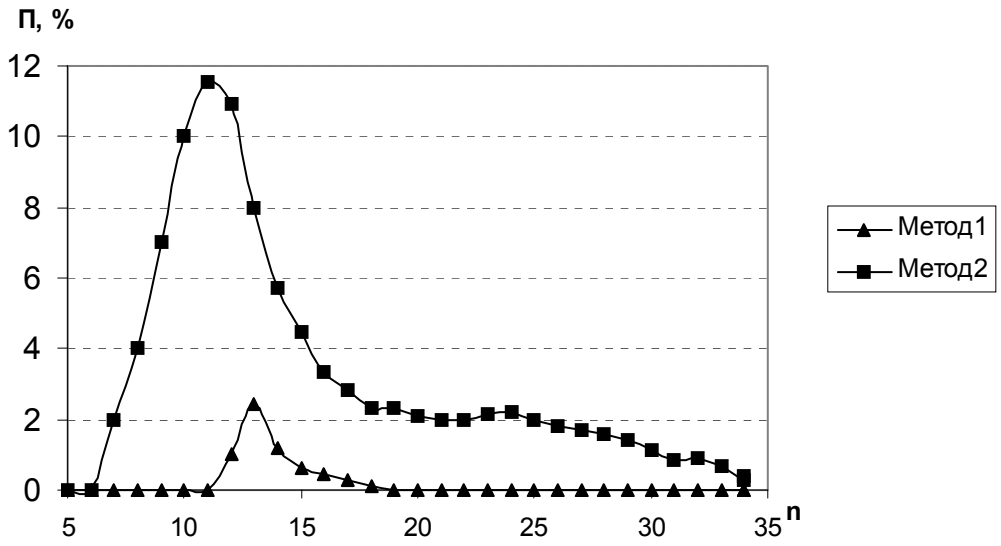


Рис. 11. Зависимость относительной погрешности от размерности решаемой задачи

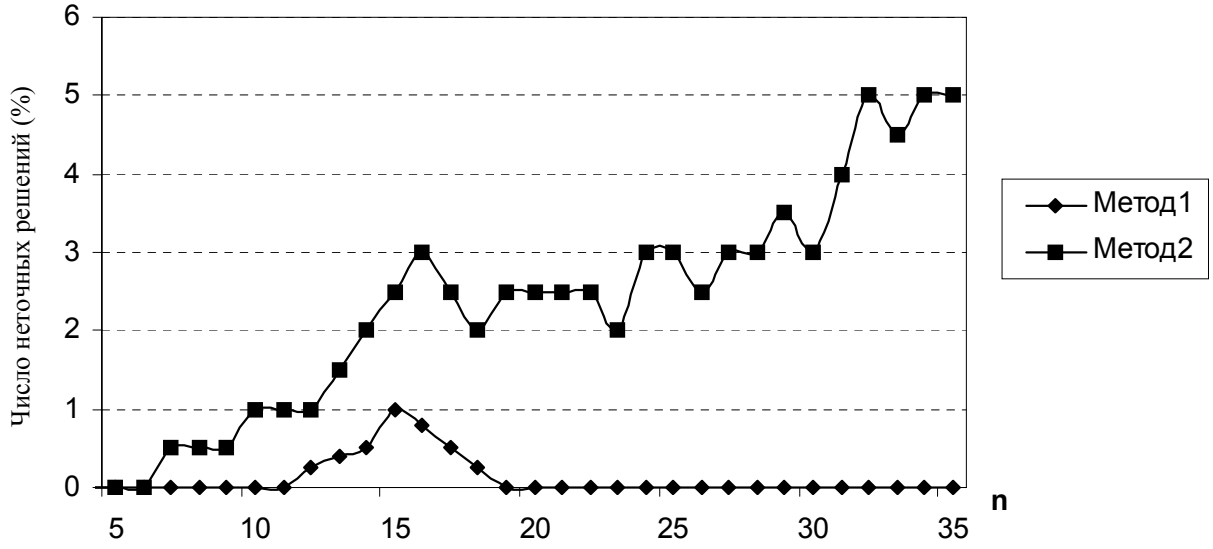


Рис. 12. Зависимость числа неточных решений в % от размерности решаемой задачи

На всех рисунках (рис. 6 – 12) разработанные алгоритмы A_1 и A_2 обозначены как метод-1 и метод-2.

Как видно из графиков, приведенных на рис. 10, при n≥27 алгоритмы Литтла и локального поиска имеют существенно более высокую временную сложность по сравнению с разработанными. При этом, как показано в [6], алгоритмы локального поиска имеют погрешность, лежащую в диапазоне от 7% до 29%, и возрастающую с увеличением размерности решаемых задач. В разработанных алгоритмах на основе идей рангового подхода погрешность с увеличением размерности задачи стабилизируется и не превышает 2% (рис. 11). Из графиков, приведенных на рис. 11, видно, что процент неточных решений для алгоритма A_1 при $n \geq 27$ очень низок. Экс-

периментальное исследование показало, что при n ≥ 27 на 2000 тестовых задач в среднем только 3 давали приближенное решение и при этом погрешность не превышала 1–2%. Использование эвристического правила в алгоритме A_2 позволило, с одной стороны, существенно уменьшить временную сложность этого алгоритма по сравнению с A_1, но при этом возрастает погрешность и число неточных решений в нем с увеличением размерности решаемой задачи растет (рис. 12).

Заключение

Таким образом, предложенный подход решения произвольных задач булевого программирования позволяет на основе предложенных алгоритмов решать с единых позиций любые задачи линейного и нелинейного программирования за полиномиальное время с требуемой точностью. Если в соотношении (1) ввести обозначение $f_k(x) = \sum_{j=1}^{p_k} C_{kj} S_k(C_n^k)$, то оно примет вид $F(x) = \sum_k f_k(x)$. В общем случае, если функционал и ограничения представляют собой произвольную нелинейную функцию от $f_k(x)$, то и такая задача нелинейного программирования также может быть эффективно решена с помощью предложенного подхода.

Литература

1. Танаев В.С. Декомпозиция и агрегирование в задачах математического программирования. – Мн.: Наука и техника, 1987. – 183 с.

2. Листровой С.В., Голубничий Д.Ю., Листровая Е.С. Метод решения задач целочисленного линейного программирования с булевыми переменными на основе рангового подхода // Электрон. моделирование. – 1998. – Т. 20, №6. – С. 14-32.

3. Листровой С.В., Третьяк В.Ф., Листровая А.С. Параллельный алгоритм оптимизации вычислительного процесса для задач булевого программирования // Электрон. моделирование. – 1998. – №5. – С. 23-33.

4. Listrovoy S.V., Golubnichiy D.Yu., Listrovaya E.S. Solution method on the basis of rank approach for integer linear problems with boolean variables // Engineering Simulation. – 1999. – Vol. 16. – P. 707-725.

5. Listrovoy S.V., Tretjak V.F., Listrovaya A.S. Parallel algorithms of calculation process optimization for the boolean programming problems // Engineering Simulation. – 1999. – Vol. 16. – P. 569-579.

6. Листровой С.В., Хрин В.Н. О возможности решения задач оптимального распределения ресурсов при управлении сложными системами в реальном масштабе времени // Изв. АН России. Техническая кибернетика. – 1992. – №4. – С. 125-133.

7. Листровой С.В. Параллельный алгоритм для задачи о кратчайших маршрутах на графе // Изв. АН СССР. Техн. кибернетика. – 1990. – № 4. – С. 67-74.

8. Листровой С.В., Хрин В.Н. Параллельный алгоритм определения путей с максимальной пропускной способностью // Кибернетика и системный анализ. – 1998. – № 2. – С. 125-134.

Поступила в редакцию 4.03.2008

Рецензент: д-р физ.-мат наук, проф. С.В. Смеляков, Харьковский национальный университет радиоэлектроники, Харьков.