ТЕОРІЯ ОПТИМАЛЬНИХ РІШЕНЬ

УДК 519.85

П.И. СТЕЦЮК, А.В. ГАСНИКОВ

NLP -ПРОГРАММЫ И r -АЛГОРИТМ В ЗАДАЧЕ ЭНТРОПИЙНО-ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

Рассматривается задача энтропийно-линейного программирования в прямой и двойственной формулировках. Для обеих задач приводятся результаты вычислительных экспериментов с программами решения задач нелинейного программирования из NEOS-солвера. Обсуждаются вычислительные эксперименты для решения двойственной задачи с помощью r -алгоритма Шора с адаптивным шагом.

Введение. Одномерная нелинейная функция $f(x) = -x\ln x + cx$ является вогнутой при $x \geq 0$. Она достигает своего максимума в единственной точке $x^* = \exp(c-1)$, такой, что $x^* > 0$ при любом значении параметра c. Этот факт позволяет эффективно учитывать требование на неотрицательность переменных в задачах энтропийно-линейного программирования (*ЭЛП*-задачи), которые имеют многочисленные приложения [1]. В этих задачах максимизируемая целевая функция – это сепарабельная вогнутая функция, построенная как сумма функций вида $f(x)$, а ограничения заданы так, как в задаче линейного программирования.

Наша цель показать, что указанные особенности *ЭЛП*-задач позволяют разрабатывать эффективные алгоритмы для решения *ЭЛП*-задач большой размерности. Сделаем это с помощью численных экспериментов для современных программ решения задач нелинейного программирования (*NLP*-программ) и r -алгоритма Шора.

Последовательность изложения материала статьи следующая. В разделе 1 для простейшей *ЭЛП*-задачи приведены формулировки прямой и двойственной задач и описаны некоторые их свойства. В разделе 2 даны результаты решения прямой и двойственной задач небольших размеров с помощью *NLP*-программ из *NEOS*-солвера. В разделе 3 приведены результаты решения двойственной *ЭЛП*-задачи больших размеров с помощью r -алгоритма с адаптивным шагом.

1. Прямая и двойственная *ЭЛП*-задачи. Пусть заданы вектор $c = \{c_i\}_{i=1}^{n}$, матрица $A = \{a_{ij}\}_{i,j=1}^{n,m}$ и вектор $b = \{b_j\}_{j=1}^{m}$, а неизвестными являются неотрицательные компоненты вектора $x = \{x_i\}_{i=1}^{n}$. Задача энтропийно-линейного программирования в простейшей постановке имеет такой вид:

$$f^* = f(x^*) = \max_{x \geq 0} \left\{ f(x) = -\sum_{i=1}^{n} x_i \ln x_i + \sum_{i=1}^{n} c_i x_i \right\} \tag{1}$$

при ограничениях

$$\sum_{i=1}^{n} a_{ij} x_i = b_j,\ j = 1,...,m. \tag{2}$$

Задачу (1) – (2) условимся называть прямой *ЭЛП*-задачей с n переменными и m ограничениями (ограничения на неотрицательность переменных учитывать не будем). Будем предполагать, что система ограничений $Ax = b$ имеет допустимые точки $\overline{x} = \{\overline{x}_i > 0\}_{i=1}^{n}$, т. е. задача (1) – (2) имеет единственное оптимальное решение $x^* = \{x_i^* > 0\}_{i=1}^{n}$, все компоненты которого положительны.

Если $u = \{u_j\}_{j=1}^{m}$ – неизвестные множители Лагранжа, соответствующие ограничениям (2), то двойственная *ЭЛП*-задача имеет следующий вид:

$$\psi^* = \psi(u^*) = \min_{u \in R^n} \left\{ \psi(u) = -\sum_{j=1}^{m} b_i u_i + \sum_{i=1}^{n} \exp\left(c_i + \sum_{j=1}^{m} a_{ij} u_j - 1 \right) \right\}. \tag{3}$$

Она является задачей безусловной минимизации гладкой выпуклой функции $\psi(u)$ от m переменных. Из теории двойственности следует, что $\psi^* = f^*$. Если известны $u^* = \{u_j^*\}_{j=1}^{m}$ – оптимальные значения множителей Лагранжа для задачи (3), то оптимальные значения переменных в задаче (1) – (2) вычисляются по формуле $x^* = \exp\left(c + A^T u^* - e\right)$, где e – вектор, состоящий из n единиц.

Главная особенность *ЭЛП*-задач – это выполнение условия $n \gg m$. Поэтому быстрое решение задачи (1) – (2) кроется в эффективном решении задачи (3), количество переменных в которой намного меньше, чем количество переменных в прямой *ЭЛП*-задаче. Так, если в прямой *ЭЛП*-задаче количество переменных n будет порядка сотен тысяч, а количество ограничений m – порядка нескольких тысяч, то решение задачи (3) можно осуществить с помощью стандартных *NLP*-программ. Однако, более эффективными будут специальные двойственные алгоритмы, которые учитывают специфику прямой и двойственной *ЭЛП*-задач. Двойственный алгоритм будет определяться выбранным градиентным алгоритмом для минимизации гладкой функции $\psi(u)$. Изложенное подтвердим далее, где задачу (3) будем решать с помощью r-алгоритма.

2. *NLP*-программы из *NEOS*-солвера. Как задача (1) – (2), так и задача (3) являются задачами нелинейного программирования, для решения которых существуют эффективные *NLP*-программы. Их можно использовать через *NEOS*-солвер [2], который является частью системы NEOS (Network-Enhanced Optimization System), содержащей библиотеку оптимизационного программного обеспечения, оптимизационный сервер (http://www-neos.mcs.anl.gov/) для использования этой библиотеки в сети Интернет, а также необходимую информацию о программном обеспечении. *NLP*-программы находятся в секции «Nonlinearly Constrained Optimization» *NEOS*-солвера. Девять из них поддерживают работу с описанием задач нелинейного программирования на языке моделирования AMPL [3]. Это известные программы: CONOPT 3.15C, filter, Ipopt 3.10.3, KNITRO 9.0.0, LANCELOT, LOQO 7.00, MINOS 5.51, MOSEK и SNOPT 7.2-10.

Для этих программ проведен вычислительный эксперимент, который состоял в одновременном решении десяти прямых и десяти двойственных *ЭЛП*-задач при $n = 4000$ и $m = 400$. Данные для *ЭЛП*-задачи генерировались с помощью *AMPL*-го датчика **Uniform(0,1)**, который обеспечивает равномерное распределение случайных чисел на (0,1). Компоненты вектора c выбирались равными $10\xi_i + 1$, $i = 1,...,n$, а компоненты матрицы A – равными $5\xi_{ij}$, $i = 1,..., n$, $j = 1,..., m$. Вектор b вычислялся по формуле $b = Ax_0$, где компоненты вектора x_0 выбирались равными $3\xi_i + 1$, $i = 1,..., n$.

Успешно решили все задачи те пять программ, для которых в табл. 1 приведены t_p и t_d – времена (в секундах), затраченные на решение прямой и двойственной *ЭЛП*-задач. В последней строке даны средние времена для десяти задач.

Четыре программы не справились с решением всех задач. Программа LOQO не смогла вычислить значения функции и ограничений в начальной точке, а программы filter, LANCELOT и SNOPT были сняты системой из-за превышения максимально допустимого времени. При этом filter и SNOPT не решили ни одной задачи, а LANCELOT успела решить четыре задачи из десяти, затратив на решение прямых задач 3607, 6759, 6942 и 6790 секунд, а на решение двойственных задач 85, 86, 34 и 77 секунд соответственно.

Из табл. 1 видим, что наилучшими по времени оказались программы KNITRO и Ipopt. С их помощью решались *ЭЛП*-задачи, размеры которых были увеличены. Если $n = 10000$ и $m = 600$, то средние времена для KNITRO оказались такими: $\overline{t}_p = 222.7$ и $\overline{t}_d = 91.5$. Из-за нехватки оперативной памяти программа Ipopt не решила прямые задачи, а среднее время решения двойственных задач для нее составило $\overline{t}_d = 85.2$. Если $n = 40000$ и $m = 600$, то ни одной из программ не хватило памяти для решения прямых задач, а среднее время решения двойственных задач составило $\overline{t}_d = 283.7$ для программы KNITRO и $\overline{t}_d = 324.6$ для программы Ipopt.

ТАБЛИЦА 1. Результаты для *NLP*-программ из *NEOS*-солвера (22.01.2015)

№	**CONOPT**		**Ipopt**		**KNITRO**		**MINOS**		**MOSEK**	
	t_p	t_d	t_p	t_d	t_p	t_d	t_p	t_d	t_p	t_d
1	290.2	20.3	45.6	22.9	6.1	22.1	1580.5	290.2	35.1	114.8
2	292.9	13.3	48.0	14.5	8.5	18.5	1732.2	252.0	36.1	153.4
3	301.6	14.7	45.8	14.3	8.7	13.4	1556.5	299.9	50.6	136.8
4	276.9	17.6	48.2	14.3	8.7	13.4	1583.3	289.6	39.4	114.0
5	234.2	16.0	47.7	14.3	8.6	13.4	1483.5	281.5	38.2	119.1
6	204.1	13.3	48.0	14.5	8.3	13.4	1418.7	254.9	29.9	142.2
7	275.3	13.9	47.7	14.3	14.0	15.8	1554.0	257.2	35.3	130.7
8	206.8	13.1	47.8	14.3	9.1	21.5	1545.9	265.8	28.3	132.7
9	242.8	16.2	47.9	14.5	20.1	15.8	1447.0	252.6	38.4	137.5
10	274.6	18.7	47.9	14.3	9.7	21.9	1505.1	305.3	25.5	143.4
$\overline{t}$	**259.9**	**15.7**	**47.5**	**15.2**	**10.2**	**16.9**	**1540.7**	**274.9**	**35.7**	**132.4**

3. *r*-алгоритм Шора [4]. Для того, чтобы оценить время решения задачи (2) с помощью r-алгоритма, была реализована программа **delp** на МАТЛАБ-подобном некоммерческом языке GNU Octave. В ее основу положен код octave-программы **ralgb5** [5, стр. 384 – 385], для которой значения функции $\psi(u)$ и ее градиента $g_\psi(u)$ вычислялись по такому правилу:

$$x(u) = \exp\left(c + A^T u - e\right), \quad \psi(u) = -b^T u + e^T x(u), \quad g_\psi(u) = -b + Ax(u), \qquad (4)$$

где e – вектор, состоящий из n единиц. Программа **ralgb5** реализует $r(\alpha)$-алгоритм с постоянным коэффициентом растяжения пространства α и адаптивным шагом, который настраивается с помощью параметров h_0, q_1, q_2 и n_h.

Вычисления проводились на компьютере Pentium 3GHz в системе Windows7/32 с использованием GNU Octave версии 3.6.4. Тестовые *ЭЛП*-задачи были такими же, как и для *NLP*-программ, но случайные числа с равномерным распределением на (0,1) генерировались октавовским датчиком **rand("seed", 2015)**. Программа **delp** показала очень незначительные затраты по времени (см. табл. 2) для *ЭЛП*-задач, имеющих размеры, близкие к используемым выше.

В табл. 2 дано время (в секундах) $t_1,...,t_7$ для решения серии из семи *ЭЛП*-задач одного размера, в последнем столбце приведено среднее время решения задач этой серии. Для расчетов параметры r-алгоритма выбирались такими: $\alpha = 2$, $h_0 = 1.0$, $q_1 = 0.95$, $q_2 = 1.1$ и $n_h = 3$. Программа останавливалась, если для точки u_r выполнялось условие $\left\|u_r - u^*\right\| \le 10^{-7}$, где $\|\circ\|$ – эвклидова норма. Это гарантировало, что норма относительных невязок для ограничений в форме равенств (2) не превышала $10^{-8} - 10^{-7}$.

ТАБЛИЦА 2. Времена работы программы **delp** для небольших ЭЛП-задач

№	n	m	t_1	t_2	t_3	t_4	t_5	t_6	t_7	$\overline{t}$
1	4000	400	1.27	1.25	1.23	1.23	1.23	1.22	1.25	1.24
2	10000	400	2.14	2.09	2.10	2.10	2.13	2.12	2.11	2.11
3	40000	400	9.82	9.92	9.92	10.02	9.77	9.79	9.75	9.86
4	4000	600	2.20	2.19	2.20	2.20	2.20	2.23	2.20	2.20
5	10000	600	3.47	3.50	3.45	3.45	3.43	3.48	3.50	3.47
6	40000	600	11.54	10.98	11.25	11.63	11.36	11.42	11.59	11.40
7	4000	800	3.58	3.58	3.58	3.59	3.57	3.57	3.57	3.58
8	10000	800	5.16	5.16	5.18	5.16	5.19	5.17	5.17	5.17
9	40000	800	12.85	13.13	13.00	13.01	13.01	12.97	13.02	13.00

Скорость работы программы **delp** во многом зависит от настройки трех параметров $r(\alpha)$-алгоритма: α, h_0 и q_1. Их надо выбирать с учетом того, что минимизируется гладкая выпуклая функция, т. е при заданном $\alpha \in [2,4]$ эффективную работу будет обеспечивать такое значение параметра $q_1 < 1$ (уменьшающее величину шага на итерации), при котором точность спуска по направлению увеличивается по ходу итераций. Это означает, что на одну итерацию $r(\alpha)$-алгоритма должно приходиться в среднем около $1.2 \div 1.5$ вычислений значений функции и градиента. Для задач из табл. 2 такой настройки не требовалось, так как улучшать время решения на несколько секунд особого смысла не имеет.

Однако, если размеры *ЭЛП*-задачи большие, то согласованная настройка параметров α и q_1 позволяет добиться значительного эффекта. Насколько за счет этого для нашего тестового примера можно ускорить программу **delp** легко видеть из табл. 3, где приведены результаты расчета для трех *ЭЛП*-задач больших размеров. Их размеры выбирались такими, чтобы количество элементов в матрице ограничений равнялось 40.000.000. В таблице дано количество итераций (itn) и количество вычислений функции (nfg) в зависимости от параметра $q_1 \in \{0.9, 0.95, 0.99, 1.0\}$. Все остальные параметры в расчетах были такими же, как при решении небольших задач. В табл. 3 приведено также и время решения задач (в секундах). Минимальные из них получены при $\alpha = 2$ и $q_1 = 0.9$ и составляют для первой задачи 33 секунды, для второй – 15 секунд, а для третьей – 55 секунд. При $q_1 = 1.0$ для двух задач было превышено максимальное количество итераций (maxitn = 5000).

ТАБЛИЦА 3. Работа программы **delp** для больших задач при разных q_1

№	n	m	q_1	Itn	Nfg	t	q_1	Itn	nfg	t
1	1000000	40	0.90	185	249	33	0.95	313	411	54
			0.99	614	774	103	1.00	898	1135	150
2	100000	400	0.90	230	313	15	0.95	537	760	38
			0.99	1828	2380	118	1.00	5000	6007	299
3	10000	4000	0.90	247	343	55	0.95	314	320	65
			0.99	1567	1572	326	1.00	5000	5003	1040

Заключение. Приведенные вычислительные эксперименты для r-алгоритма Шора с адаптивной регулировкой шага подтверждают, что для решения задач энтропийно-линейного программирования больших размеров целесообразно разрабатывать «быстрые» специализированные алгоритмы. Для задач небольших размеров они будут эффективнее, чем современные оптимизационные программы общего назначения. Если количество переменных будет порядка сотен тысяч, а количество ограничений – порядка нескольких тысяч, то решение *ЭЛП*-задач можно получить за несколько минут на современных персональных компьютерах. Если матрица ограничений разрежена либо имеет блочную структуру, то учет специфики матрицы только усилит выигрыш по времени для специализированных алгоритмов.

Работа выполнена при поддержке НАНУ (проект № 0114U001055) и РФФИ (гранты 13-01-12007-офи_м, 14-01-00722-а).

П.І. Стецюк, О.В. Гасніков

NLP-ПРОГРАМИ ТА r-АЛГОРИТМ У ЗАДАЧІ ЕНТРОПІЙНО-ЛІНІЙНОГО ПРОГРАМУВАННЯ

Розглядається задача ентропійно-лінійного програмування у прямому та двоїстому формулюваннях. Для обох задач наведено результати обчислювальних експериментів з програмами розв'язання задач нелінійного програмування із NEOS-солвера. Обговорюються обчислювальні експерименти для розв'язання двоїстої задачі за допомогою r-алгоритма Шора з адаптивним кроком.

P.I. Stetsyuk, A.V. Gasnikov

NLP-PROGRAMS AND r-ALGORITHM FOR ENTROPY LINEAR PROGRAMMING PROBLEM

The maximum entropy linear programming problem in primal and dual formulations is considered. The results of computational experiments with the NEOS-solvers for nonlinear programming problems are reported for both formulations. The computational experiments with solving the dual problem using Shor's r-algorithm with an adaptive step are discussed.

1. *Fang S.-C., Rajasekera J.R., Tsao H.-S.J.* Entropy optimization and mathematical programming. – Kluwer's International Series, 1997. – 343 p.
2. *NEOS Solver* [Электронный ресурс]: http://www.neos-server.org/neos/solvers/ – Режим доступа: свободный.
3. *Fourer R., Gay D., Kernighan B.* AMPL: A Modeling Language for Mathematical Programming. – Belmont: Duxburry Press, 2003. – 517 p.
4. *Шор Н.З.* Методы минимизации недифференцируемых функций и их приложения. – Киев: Наук. думка, 1979. – 200 с.
5. *Стецюк П.И.* Методы эллипсоидов и r-алгоритмы. – Кишинэу: Эврика, 2014. – 488 с.

Получено 02.02.2015