Компьютерная математика

УДК 519.854.33

П.В. ШИЛО

ПРИМЕНЕНИЕ «БЕСПОЛЕЗНЫХ» ХОДОВ ПРИ РЕШЕНИИ ЗАДАЧИ О ПОКРЫТИИ

Предложена модификация алгоритма случайного повторного локального поиска для решения задачи о покрытии с применением «бесполезных» ходов, что позволяет расширить поисковые возможности алгоритма. Эффективность разработанного алгоритма подтверждена экспериментально при решении задач большой размерности, а также сравнением полученных результатов с известными. С помощью предложенного алгоритма найдено новое рекордное решение.

Введение. Широко распространенные методы решения задач дискретной оптимизации – это методы локального типа. Один из важных моментов в процессе локального поиска – вопрос о количестве шагов, необходимых для достижения локального экстремума. В работе [1] этот вопрос исследован с позиции теории сложности. Следует отметить, что при нахождении локальных решений многих классов дискретных оптимизационных задач (например, задач о максимальном разрезе графа, о рюкзаке) часто встречаются текущие допустимые решения с одинаковыми значениями целевой функции. Ходы в алгоритмах локального поиска, не улучшающие значение целевой функции, т. е. не приносящие никакой пользы, будем называть «бесполезными». Пусть на какой-то итерации локального поиска сделан ход $x \to y$ (из решения x сделан переход в решение y) и $f(x) = f(y)$. Это и есть бесполезный ход. Когда таких ходов много, как, например, часто бывает при решении вышеназванных и других задач, возникает вопрос об их эффективном использовании. Собственно, необходимо решить, как преодолеть «плато» ландшафта целевой функции решений, порождающее это множество бесполезных ходов, и приблизиться к лучшим решениям. Этому вопросу и посвящена данная статья.

Постановка задачи. Рассмотрим задачу о покрытии минимальной мощности следующего вида: найти

$$\min \sum_{j=1}^{n} x_j \qquad (1)$$

при условиях

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \; i \in M = \{1,...,m\}, \qquad (2)$$

$$x_j = 0 \vee 1, \; j \in N = \{1,...,n\}, \qquad (3)$$

где $a_{ij} \in \{0,1\}, i \in M, j \in N$. Она имеет многочисленные приложения в дискретной математике, теории графов, планировании и др. Примером такой задачи является задача водопроводчика [2].

Задача вида (1) – (3) *NP*-трудная и наиболее сложная для решения среди задач о покрытии. Лучшие известные алгоритмы ее решения приведены в работах [3 – 9]. Настоящая работа посвящена дальнейшему развитию результатов работ [8, 9].

Алгоритмы. В работах [8, 9] предложены различные варианты алгоритма случайного повторного локального поиска, которые показали хорошие результаты. Вначале приведем схему алгоритма случайного локального поиска [9] (алгоритма I).

Схема алгоритма I

1. RandomLocalSearchMG (*F*)
2. Calc(n_control,n_cover,*F*)
3. while(*F* не является покрытием)
4. Формирование множества *Max_cover*
5. Формирование множества *BestMove*
6. $S_j \leftarrow$ RandomSelectElement(*BestMove*)
7. $F = F \cup S_j$
8. ReCalc(n_control,n_cover,*F*)
9. Формирование множества *Redundant*
10. while(*Redundant* не пусто)
11. $S_j \leftarrow$ RandomSelectElement(Redundant)
12. $F = F \setminus S_j$
13. ReCalc(n_control,n_cover,*F*)
14. Формирование множества *Redundant*
15. end while
16. Формирование множества *Cand_Redundant*
17. if (*Cand_Redundant* не пусто)
18. $S_j \leftarrow$ RandomSelectElement(Cand_Redundant)
19. goto line 6
20. end if
21. end while
22. end

В строке 2 осуществляется вызов процедуры Calc. В этой процедуре для подмножеств $S_j \cap F$ рассчитываются величины $n\_control_j$, а для подмножеств $S_j \cap F$ – величины n_cover_j. Здесь $n\_control_j$ – количество покрытых элементов из *M*, покрываемых только подмножеством S_j, а n_cover_j – число непокрытых элементов из *M*, покрываемых подмножеством S_j. В строке 4 формируется множество *Max_cover*, состоящее из подмножеств S_j с максимальным значением n_cover_j.

Множество *BestMove*, которое формируется в строке 5, состоит из подмножеств S_j, входящих в множество *Max_cover* с максимальным значением *gmod*. В строке 6 случайно выбирается подмножество S_j из множества *Max_cover* с одинаковой вероятностью. Процедура ReCalc вызывается в строках 8, 13. В ней для подмножеств $S_j \cap F$ пересчитываются величины $n\_control_j$, а для подмножеств $S_j \cap F$ – величины n_cover_j. Строки 9 и 14 служат для формирования множества *Redundant*, состоящего из подмножеств $S_j \cap F$, для которых $n\_control_j = 0$. В строке 16 формируется множество *Cand_Redundant*, которое состоит из подмножеств $S_j \cap F$, таких, что $n\_cover_j > 0$, и введение S_j в F приведет к появлению непустого множества *Redundant*. Если множество *Cand_Redundant* не пусто, то из него с одинаковой вероятностью случайно выбирается подмножество S_j (строка 18) и осуществляется переход на строку 6. Блок, состоящий из строк 16 – 20, позволяет делать ходы, отличные от ходов жадного алгоритма.

Алгоритм I базируется на использовании повторного жадного алгоритма [6], который на каждом ходе (строки 4 – 7) пытается максимизировать число покрытых элементов. В работе [9] предложено выбирать ходы, максимизирующие следующую функцию:

$$gmod(F) = L_0 \cdot numbcover + \sum_{k=0}^{L_{\max}} L_k \cdot numbcontrol_k,$$

где $numbcover$ – число покрытых элементов, а $numbcontrol_k$ – число подмножеств $S_j \cap F$, которые контролируют ровно k элементов, $L_{\max} = |F|$, $|F|$ – мощность множества *F*. Значения L_k, $k = L_{\max},..,1$ выбираются из тех же соображений, что и в [10]:

$$L_k = \begin{cases} 1 + L_{k+1} + |F| \cdot (L_{k+1} - L_{k+2}) \text{ если, } k \le |F|, \\ 0, \text{ если } k > |F|. \end{cases}$$

Поскольку числа L_k могут быть очень большими, что неудобно при вычислениях, была использована «усеченная» модифицированная функция:

$$L_k = \begin{cases} |F|^{L_{\max} - k} & \text{при } k \le L_{\max},\ L_{\max} \le |F|. \\ 0 & \text{при } k > L_{\max}. \end{cases}$$

Далее приведем схему алгоритма случайного повторного локального поиска IteratedRandomLocalSearchMG [9] (алгоритма II), в которой используется метод дихотомии.

Схема алгоритма II.

```
1.  IteratedRandomLocalSearchMG (F)
2.      BestF = {1,2,...,n}
3.      for nat = 1 to maxnat
4.          F = Ж
5.          RandomLocalSearchMG(F)
6.          MinF = F; nbad = 0; ln_delete = 1; un_delete =| MinF|
7.          for niter = 1 to maxniter
8.              F= MinF
9.              Случайное удаление n_delete подмножеств из F
10.             RandomLocalSearchMG(F)
11.             if (|F| ≤ |MinF|) then
12.                 MinF = F
13.                 if (|F| < |BestF|) then BestF = F
14.             end if
15.             else nbad = nbad+1
16.             if (niter кратна величине ntune)
17.                 if (nbad > ubad) then
18.                     un_delete = n_delete
19.                     n_delete = (ln_delete + un_delete )/2
20.                 end if
21.                 if (nbad < lbad) then
22.                     ln_delete = n_delete
23.                     n_delete = (ln_delete + un_delete )/2
24.                 end if
25.                 nbad = 0
26.             end if
27.         end for
28.     end for
29. end
```

Блок, состоящий из строк 16 – 26, служит для настройки параметра n_delete. Каждые ntune итераций величина nbad, равная количеству итераций, на которых найдено решение F, худшее, чем $MinF$, используется для коррекции значения n_delete. Отметим, что схема повторности в алгоритме II отличается от предложенных в [6, 8].

В предлагаемой модификации алгоритма [9] используются бесполезные ходы. Рассмотрим схему процедуры RandomLocalSearchMG (*F*). Если после формирования множества *Cand_Redundant* (строка 16) оно оказалось пустым, то допускается включение в него подмножеств $S_j \Pi F$, таких, что $n\_cover_j = 0$. Очевидно, что это приводит к использованию бесполезных ходов, что, однако, позволяет расширить поисковые возможности алгоритма. Применение таких ходов может приводить к зацикливанию алгоритма. Чтобы избежать этого, предусмотрено запоминание выполненных бесполезных ходов. Это позволяет препятствовать их повторению и, в конечном счете, избежать зацикливания. Модификация алгоритма оказалась эффективной, что подтверждают приведенные далее результаты вычислительного эксперимента.

Результаты экспериментальных расчетов. Проведен вычислительный эксперимент по использованию предложенного алгоритма для решения 70 случайно сгенерированных тестовых задач scp4-scp6, scpa-scpe и scpnre-scpnrh вида (1) – (3) большой размерности, доступных в OR-library (http://mscmga.ms.ic.ac.uk/jeb/orlib/scpinfo.html). Также проведено сравнительное исследование рассмотренного и известных алгоритмов [3 – 9].

Предложенный алгоритм реализован на языке C++, вычислительный эксперимент проводился с использованием PC с Intel® Core QUAD CPU Q9550 2.83GHz и 8.0GB оперативной памяти. Каждая тестовая задача решалась 100 раз. Параметры алгоритма указаны в табл. 1. В отличие от алгоритмов [6, 7], где использовалась настройка на задачи, все параметры предложенного алгоритма не изменялись при проведении вычислительного эксперимента.

ТАБЛИЦА 1. Параметры алгоритма

Параметр	maxnat	maxniter	ntune	lbad	ubad	$L_{\max}$
Значение	100	3000	27	18	24	4

Результаты экспериментальных расчетов приведены в табл. 2, 3. В них ρ – плотность матрицы $\left\| a_{ij} \right\|_{m \times n}$ (отношение числа единичных элементов к общему числу элементов матрицы). В пятой–девятой колонках таблиц содержатся рекорды best, полученные соответственно в работах [3 – 7]; в десятой колонке – наилучшие известные рекорды, 19 из которых получены в работах [8, 9]. В одиннадцатой колонке приведены рекорды, найденные предложенным алгоритмом. В двенадцатой колонке содержится количество вызовов ncalc процедуры RandomLocalSearchMG при ста решениях соответствующей задачи предложенным алгоритмом, в тринадцатой колонке – количество nbest найденных этим алгоритмом рекордов (из 100). В последней колонке приведено наименьшее время mintime (в сек.) поиска решения одной из сотни задач разработанным алгоритмом.

ТАБЛИЦА 2. Результаты расчетов для задач scp4-scp6, scpa, scpb

Задача	*m*	*n*	ρ (%)	[1]	[2]	[3]	[4]	[5]	BKS	Our best	ncalc	nbest	min time
1	2	3	4	5	6	7	8	9	10	11	12	13	14
41	200	1000	2	41	38	38	38	38	38	38	25732	100	0
42	200	1000	2	38	37	37	37	37	37	37	2820	100	0
43	200	1000	2	40	38	38	38	38	38	38	2524	100	0
44	200	1000	2	41	39	38	39	38	38	38	3087972	93	0,02
45	200	1000	2	40	38	38	38	38	38	38	33176	100	0
46	200	1000	2	40	38	37	37	37	37	37	300020	100	0
47	200	1000	2	41	38	38	38	38	38	38	90860	100	0
48	200	1000	2	40	38	38	37	37	37	37	468549	100	0
49	200	1000	2	40	38	38	38	38	38	38	35330	100	0
410	200	1000	2	41	38	38	38	38	38	38	411025	100	0,02
51	200	2000	2	35	35	35	34	34	34	34	996676	100	0
52	200	2000	2	35	34	35	34	34	34	34	128673	100	0
53	200	2000	2	36	35	34	34	34	34	34	29065	100	0
54	200	2000	2	36	34	34	34	34	34	34	12003	100	0
55	200	2000	2	36	34	34	34	34	34	34	24682	100	0
56	200	2000	2	36	34	34	34	34	34	34	84676	100	0
57	200	2000	2	35	34	34	34	34	34	34	13990	100	0
58	200	2000	2	37	35	34	34	34	34	34	234982	100	0
59	200	2000	2	36	36	35	35	35	35	35	17787	100	0
510	200	2000	2	36	35	34	34	34	34	34	160455	100	0
61	200	1000	5	21	21	21	21	21	21	21	4778	100	0
62	200	1000	5	21	20	21	20	20	20	20	358084	100	0
63	200	1000	5	21	21	21	21	21	21	21	2607	100	0
64	200	1000	5	22	21	21	21	20	20	20	4209249	87	0
65	200	1000	5	22	21	21	21	21	21	21	13835	100	0
a1	300	3000	2	40	39	39	39	39	39	39	39564	100	0
a2	300	3000	2	41	39	39	39	39	39	38	9040670	19	0,8
a3	300	3000	2	40	39	39	39	39	39	38	9747231	4	1,51
a4	300	3000	2	40	38	38	38	37	37	37	9062181	20	1,72
a5	300	3000	2	40	39	38	38	38	38	38	299480	100	0,06
b1	300	3000	5	23	22	22	22	22	22	22	16829	100	0,01
b2	300	3000	5	22	22	22	22	22	22	22	6945	100	0
b3	300	3000	5	22	22	22	22	22	22	22	37263	100	0
b4	300	3000	5	23	22	22	22	22	22	22	100383	100	0,02
b5	300	3000	5	23	22	22	22	22	22	22	37001	100	0,02

ТАБЛИЦА 3. Результаты расчетов для задач scpc-scpe, scpnre- scpnrh

Задача	*m*	*n*	ρ (%)	[1]	[2]	[3]	[4]	[5]	BKS	Our best	ncalc	nbest	mintime
1	2	3	4	5	6	7	8	9	10	11	12	13	14
c1	400	4000	2	45	44	43	43	43	43	43	158673	100	0,01
c2	400	4000	2	45	44	44	43	43	43	43	179561	100	0,08
c3	400	4000	2	45	44	43	43	43	43	43	223861	100	0
c4	400	4000	2	46	44	43	43	43	43	43	134231	100	0,03
c5	400	4000	2	45	44	44	43	43	43	43	444509	100	0,11
d1	400	4000	5	26	25	25	25	25	25	24	721583	100	0,22
d2	400	4000	5	25	25	25	25	25	25	24	9321576	13	15,63
d3	400	4000	5	25	25	25	25	24	24	24	9694321	8	15,02
d4	400	4000	5	26	25	25	25	25	25	24	9743099	6	47,09
d5	400	4000	5	26	25	25	25	25	25	24	9910753	3	65,66
e1	50	500	20	5	5	5	5	5	5	5	273	100	0
e2	50	500	20	5	5	5	5	5	5	5	108	100	0
e3	50	500	20	5	5	5	5	5	5	5	100	100	0
e4	50	500	20	5	5	5	5	5	5	5	161	100	0
e5	50	500	20	5	5	5	5	5	5	5	100	100	0
nre1	500	5000	10	17	17	18	17	17	17	16	7807968	47	0,83
nre2	500	5000	10	17	17	18	17	17	17	16	8503965	30	10,26
nre3	500	5000	10	17	17	18	17	17	17	16	8681740	22	10,28
nre4	500	5000	10	17	17	17	17	17	17	16	8975564	22	8,7
nre5	500	5000	10	17	17	17	17	17	17	17	3670	100	0
nrf1	500	5000	20	10	10	11	10	10	10	10	11328	100	0
nrf2	500	5000	20	11	10	11	10	10	10	10	11417	100	0,26
nrf3	500	5000	20	11	10	11	10	10	10	10	11666	100	0,27
nrf4	500	5000	20	11	10	11	10	10	10	10	12179	100	0,17
nrf5	500	5000	20	11	10	10	10	10	10	10	11894	100	0,25
nrg1	1000	10000	2			63	62	61	61	60	9800242	3	5,86
nrg2	1000	10000	2			61	62	62	61	60	9887404	3	126,3
nrg3	1000	10000	2			62	62	62	62	61	6209178	60	2,69
nrg4	1000	10000	2			63	62	62	62	61	8087238	37	4,86
nrg5	1000	10000	2			63	62	62	62	61	8252734	34	5
nrh1	1000	10000	5			35	34	34	34	33	9849871	4	249,63
nrh2	1000	10000	5			36	34	34	34	33	9802399	4	264,17
nrh3	1000	10000	5			36	34	34	34	33	9930265	2	314,53
nrh4	1000	10000	5			35	34	34	34	33	9926608	2	344,06
nrh5	1000	10000	5			36	34	34	34	33	9821161	4	23,66

Заключение. Анализ предложенного алгоритма и результаты вычислительного эксперимента позволяют сделать такие выводы. Для задачи scpnrh5 найден новый рекорд. Получены известные рекорды для всех остальных тестовых задач, 19 из которых найдены ранее только с помощью алгоритма [9]. Что касается данных последних двух колонок табл. 2, 3, то они отличаются от результатов работы [9] в основном для некоторых задач, причем незначительно, и колеблются в ту или иную сторону. В работе [11] отмечалось, что в связи с развитием многопроцессорных комплексов при многократном решении задачи целесообразно обращать внимание не на средние показатели алгоритма, а на частоту нахождения лучших решений и минимальное время решения задачи. Из приведенных в последних двух колонках данных табл. 2, 3 следует, что разработанный алгоритм удовлетворяет современным требованиям.

П.В. Шило

ЗАСТОСУВАННЯ «ДАРЕМНИХ» ХОДІВ ПРИ РОЗВ'ЯЗАННІ ЗАДАЧІ ПРО ПОКРИТТЯ

Запропонована модифікація алгоритму випадкового повторного локального пошуку для розв'язання задачі про покриття із застосуванням «даремних» ходів, що дозволяє розширити пошукові можливості алгоритму. Ефективність розробленого алгоритму підтверджена експериментально при розв'язанні задач великої розмірності, а також порівнянням отриманих результатів із відомими. За допомогою запропонованого алгоритму знайдено новий рекордний розв'язок.

P.V. Shylo

APPLICATION OF “USELESS” MOVES TO MINIMUM CARDINALITY SET COVERING PROBLEM

In this paper, the modification of a new algorithm based on the iterated random local search for Minimum Cardinality Set Covering Problem (MCSCP) with “useless” moves is proposed that makes it possible to increase its search capabilities. The efficiency of the algorithm is confirmed experimentally by solving problems of high dimension and comparing the results with the known ones. The proposed algorithm improves the new record solution for 1 benchmark instance widely used in the literature.

Предложена модификация алгоритма случайного повторного локального поиска для решения задачи о покрытии с применением «бесполезных» ходов, что позволя-ет расширить поисковые во-зможности алгоритма. Эффек-тивность разработанного алго-ритма подтверждена экспериментально при решении задач большой размерности, а также сравнением полученных результа-тов с известными. С помощью предложенного алгоритма найде-но новое рекордное решение.

1. *Johnson D.S., Papadimitriou C.H., Yannakakis M.* How easy is local search? // Comput. Sys. Sci. – 1988. – **37**. – P. 79 – 100.
2. *Кофман А., Анри-Лабордер А*. Методы и модели исследования операций. Целочисленное программирование. – М.: Мир, 1977. – 432 с.
3. *Grossman T., Wool A.* Computational experience with approximation algorithms for the set covering problem // European Journal of Operational Research. – 1997. – **101**, N 1. – P. 81 – 92.
4. *Bautista J.* A GRASP algorithm to solve the unicost set covering problem // Computers & Operations Research. – 2007. – **34(10)**. – P. 3162 – 3173.
5. *Kinney G.W., Barnes J.W., Colletti B.W.* A Reactive Tabu Search algorithm with variable clustering for the Unicost Set Covering Problem // International Journal of Operational Research. – 2007. – **2**, N 2. – P. 156 – 172.
6. *Marchiori E., Steenbeek A.G.* An iterated heuristic algorithm for the set covering problem // In proceeding of: Algorithm Engineering, 2nd International Workshop (WAE '98, Saarbrücken, Germany, August 20 – 22, 1998). – 1998. – P. 155 – 166.
7. *Musliu N.* Local search algorithm for unicost set covering problem // In Advances in Applied Artificial Intelligence: Lecture Notes in Artificial Intelligence. – Berlin, Heidelberg: Springer. – 2006. – **4031**. – P. 302 – 311.
8. *Шило В.П.* Алгоритм случайного повторного локального поиска решения задачи о покрытии минимальной мощности // Журнал обчислювальної та прикладної математики. – 2013. – № 2(112). – С. 53 – 59.
9. *Шило В.П.* Решение задачи о покрытии минимальной мощности // Компьютерная математика. – 2013. – Вып. 2. – С. 152 – 160.
10. *Alimonti P.* New local search approximation techniques for maximum generalized satisfiability problems // Inform. Proc. Lett. – 1996. – **57**. – P. 151–158.
11. *Шило В.П., Шило О.В.* Решение задачи булева квадратичного программирования без ограничений методом глобального равновесного поиска // Кибернетика и системный анализ. – 2011. – № 6. – С. 68 – 78.

Получено 13.01.2014

Об авторах:

Шило Петр Владимирович,
инженер-программист Института кибернетики имени В.М. Глушкова НАН Украины.
E-mail: petershylo@gmail.com