УДК 519.854.33

В.П. ШИЛО

РЕШЕНИЕ ЗАДАЧИ О ПОКРЫТИИ МИНИМАЛЬНОЙ МОЩНОСТИ*

Работа посвящена решению имеющей многочисленные приложения NP-трудной задачи о покрытии минимальной мощности (MCSCP) – наиболее сложному подклассу задач о покрытии. Рассмотрены лучшие известные алгоритмы решения этой задачи. Предложен и исследован новый случайный алгоритм повторного локального поиска, использующий адаптивную настройку повторности и модифицированную целевую функцию. Приведены результаты обширных экспериментальных расчетов, которые показали преимущества предложенного алгоритма над известными лучшими алгоритмами. С помощью разработанного алгоритма найдено девятнадцать новых рекордных решений.

Введение. Распространенная экстремальная задача на множествах – это задача о покрытии (SCP). Ее математическая модель формулируется в виде задачи булева линейного программирования:

минимизировать

$$f(x) = \sum_{j=1}^{n} c_j x_j \qquad (1)$$

при ограничениях

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

$$x_j = 0 \vee 1,\ j = 1,\ldots,n. \qquad (3)$$

В случае если c_j, $j = 1,\ldots, n$, одинаковые, задача SCP называется задачей о покрытии минимальной мощности (задача MCSCP) и считается, что $c_j = 1,\ j = 1,\ldots,n$. Данная задача наиболее трудная среди задач вида (1)–(3). Решению этой задачи и посвящена настоящая работа.

К многочисленным практическим приложениям задачи о покрытии минимальной мощности следует отнести задачи планирования, тестирования, построения оптимальных логических схем, поиска компьютерных вирусов, балансировки сборочной линии, поиска информации и многие другие. Кроме того, во многих практических приложениях

* Работа выполнена при частичной финансовой поддержке Украинского научно-технологического центра (грант № 5710).

(планирование летных экипажей, полити-ческих округов, охраны природы и т. д.)

относительные изменения соответствующих весов могут быть достаточно малы, чтобы оправдать использование рассматриваемой модели. Очень часто алгоритмы для задачи о покрытии минимальной мощности входят в состав алгоритмического обеспечения современных компьютерных технологий.

Так как MCSCP является *NP*-трудной задачей, разработка приближенных алгоритмов для ее решения представляет особый интерес. В работах [1 – 5] предложены и рассмотрены различные приближенные алгоритмы для этой задачи.

Сравнительный анализ 9 разных алгоритмов, включая различные варианты жадного алгоритма, алгоритм случайного округления и алгоритм нейронных сетей, проведен в [1]. Наилучшие результаты из них показал вероятностный жадный алгоритм, но он имеет плохое качество решений. Разные варианты алгоритмов GRASP предложены и исследованы в [2]. Все они имеют среднее качество решений и требуют большого объема вычислений. Основанный на использовании метода табу алгоритм предложен в [3]. Качество решений, получаемых этим алгоритмом, выше среднего, но он требует большого объема вычислений. В работе [4] предложен случайный повторный жадный алгоритм, он превосходит предыдущие алгоритмы. Алгоритм локального поиска [5] использует приемы метода табу для избегания циклов. Он является лучшим среди перечисленных алгоритмов по качеству решений и быстродействию.

Алгоритм случайного локального поиска. В работе [6] предложен алгоритм случайного повторного локального поиска, который показал хорошие результаты. Его основная составная часть – это алгоритм случайного локального поиска, схема которого приводится далее.

```
1.    RandomLocalSearch (F)
2.        Calc(n_control,n_cover,F)
3.        while(F не является покрытием)
4.            Формирование множества Max_cover
5.            S_j ¬ RandomSelectElement(Max_cover)
6.            F = F И S_j
7.            ReCalc(n_control,n_cover,F)
8.            Формирование множества Redundant
9.            while(Redundant не пусто)
10.               S_j ¬ RandomSelectElement(Redundant)
11.               F = F \ S_j
12.               ReCalc(n_control,n_cover,F)
13.               Формирование множества Redundant
14.           end while
15.           Формирование множества Cand_Redundant
16.           if ( Cand_Redundant не пусто )
17.             S_j ¬ RandomSelectElement(Cand_Redundant)
```

18. goto line 6
19. end if
20. end while
21. end

В строке 2 вызывается процедура Calc. В ней для подмножеств $S_j \in F$ рассчитываются величины $n\_control_j$, а для подмножеств $S_j \notin F$ – величины n_cover_j. Величина $n\_control_j$ равна количеству покрытых элементов из $M = \{1, ..., m\}$, покрываемых только подмножеством S_j, а величина n_cover_j – числу непокрытых элементов из *M*, покрываемых подмножеством S_j. Множество *Max_cover*, состоящее из подмножеств S_j с максимальным значением n_cover_j, формируется в строке 4. В строке 5 случайно выбирается подмножество S_j из множества *Max_cover* с одинаковой вероятностью. В строках 8, 13 вызывается процедура ReCalc. В ней для подмножеств $S_j \in F$ пересчитываются величины $n\_control_j$, а для подмножеств $S_j \notin F$ – величины n_cover_j. В строках 8 и 13 формируется множество *Redundant*, состоящее из подмножеств $S_j \in F$, для которых $n\_control_j = 0$. В строке 15 формируется множество *Cand_Redundant*, состоящее из подмножеств $S_j \notin F$, 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 17) и осуществляется переход на строку 6.

Блок (строки 15 – 19) позволяет делать ходы, отличные от ходов жадного алгоритма. Для того чтобы усилить его роль, можно проводить определенную селекцию ходов, выполняемых в строках 4 – 6. Суть данной селекции описана далее.

Модифицированная целевая функция. Жадный алгоритм при каждом ходе (строки 4 – 6) пытается максимально уменьшить число непокрытых элементов или, что тоже, максимизировать число покрытых элементов. Мы предлагаем 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 $S_j \in F$, 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 [7]:

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

Так как числа L_k могут быть очень большими, что неудобно при вычислениях, мы применяли «усеченную» модифицированную функцию:

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

Строки 4, 5 процедуры RandomLocalSearch (*F*) заменяются в процедуре RandomLocalSearchMG(*F*) такими строками:

4. Формирование множества *Max_cover*.
5. Формирование множества *BestMove*.
6. S_j ¬ RandomSelectElement(*BestMove*).

В строке 5 формируется множество *BestMove*. Оно состоит из подмножеств S_j, входящих в множество *Max_cover*, с максимальным значением *gmod*.

Алгоритм случайного повторного локального поиска. Далее приведена схема предлагаемого случайного повторного алгоритма локального поиска IteratedRandomLocalSearchMG. В отличие от предложенной в [6] в этой схеме используется метод дихотомии.

```
1.    IteratedRandomLocalSearchMG (F)
2.        BestF = {1,...,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| J |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= (un_delete + un_delete )/2
20.                   end if
21.                   if (nbad<lbad) then
22.                       ln_delete= n_delete
23.                       n_delete= (un_delete + un_delete )/2
```

```
24.                         end if
25.                         nbad=0
26.                     end if
27.                 end for
28.             end for
29.     end
```

Блок (строки 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 здесь нами выбрана схема повторности, отличная от предложенных в [4, 6].

Результаты вычислительного эксперимента. Эффективность разработанного и существующих алгоритмов исследована при решении 70 случайно сгенерированных тестовых задач 4 – 6, A-E и NRE-NRH большой размерности. Данные задачи доступны в OR-library (http://mscmga.ms.ic.ac.uk/jeb/orlib/scpinfo.html). Их характеристики приведены в табл. 1, где ρ – плотность матрицы $[a_{ij}]_{m\times n}$ (под плотностью понимаем отношение числа единичных элементов к общему числу элементов матрицы).

ТАБЛИЦА 1. Характеристики тестовых задач

Задача	*m*	*n*	ρ (%)
4	200	1000	2
5	200	2000	2
6	200	1000	5
A	300	3000	2
B	300	3000	5
C	400	4000	2
D	400	4000	5
E	50	500	20
NRE	500	5000	10
NRF	500	5000	20
NRG	1000	10000	2
NRH	1000	10000	5

Алгоритм IteratedRandomLocalSearchMG реализован на языке C++, все вычислительные эксперименты проводились с использованием PC с Intel® Core QUAD CPU Q9550 2.83GHz и 8.0GB оперативной памяти. Каждая тестовая задача решалась 100 раз. Параметры алгоритма приведены в табл. 2. Заметим, что в отличие от алгоритмов [4, 5], использующие настройку на задачи, все параметры IteratedRandomLocalSearchMG не изменялись при проведении вычислительных экспериментов.

ТАБЛИЦА 2. Параметры алгоритма

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

В табл. 3, 4 приведены результаты решения тестовых задач. Во второй – шестой колонках таблиц содержатся рекорды, полученные соответственно в работах [1 – 5], в седьмой колонке – наилучшие известные рекорды. В восьмой колонке приведены рекорды, полученные предложенным алгоритмом IteratedRandomLocalSearchMG, в девятой колонке – количество nbest найденных данным алгоритмом рекордов (из 100). В десятой колонке содержится количество вызовов ncalc процедуры RandomLocalSearchMG при ста решениях соответствующей задачи алгоритмом IteratedRandomLocalSearchMG. В последней колонке приведено наименьшее время mintime (в сек.) поиска решения одной из сотни задач предложенным алгоритмом.

ТАБЛИЦА 3. Результаты решения задач 4 – 6, A – B

Задача	Best[1]	Best[2]	Best[3]	Best[4]	Best[5]	BKS	Our best	nbest	ncalc	mintime
1	2	3	4	5	6	7	8	9	10	11
scp41	41	38	38	38	38	38	38	100	28183	0
scp42	38	37	37	37	37	37	37	100	3188	0
scp43	40	38	38	38	38	38	38	100	2725	0
scp44	41	39	38	39	38	38	38	95	3514305	0,02
scp45	40	38	38	38	38	38	38	100	36035	0
scp46	40	38	37	37	37	37	37	100	532298	0
scp47	41	38	38	38	38	38	38	100	110930	0
scp48	40	38	38	37	37	37	37	100	710963	0,03
scp49	40	38	38	38	38	38	38	100	43800	0
scp410	41	38	38	38	38	38	38	100	535295	0,03
scp51	35	35	35	34	34	34	34	100	966889	0
scp52	35	34	35	34	34	34	34	100	144135	0
scp53	36	35	34	34	34	34	34	100	26352	0
scp54	36	34	34	34	34	34	34	100	17556	0
scp55	36	34	34	34	34	34	34	100	42432	0
scp56	36	34	34	34	34	34	34	100	90570	0,01
scp57	35	34	34	34	34	34	34	100	17970	0
scp58	37	35	34	34	34	34	34	100	289661	0
scp59	36	36	35	35	35	35	35	100	17596	0
scp510	36	35	34	34	34	34	34	100	156533	0,02
scp61	21	21	21	21	21	21	21	100	4251	0
scp62	21	20	21	20	20	20	20	100	325570	0,02
scp63	21	21	21	21	21	21	21	100	2941	0
scp64	22	21	21	21	20	20	20	93	3903382	0
scp65	22	21	21	21	21	21	21	100	13101	0

Окончание табл. 3

1	2	3	4	5	6	7	8	9	10	11
scpa1	40	39	39	39	39	39	39	100	41931	0
scpa2	41	39	39	39	39	39	38	15	9247665	1,28
scpa3	40	39	39	39	39	39	38	2	9919192	5,16
scpa4	40	38	38	38	37	37	37	21	8716559	0,67
scpa5	40	39	38	38	38	38	38	100	341262	0,02
scpb1	23	22	22	22	22	22	22	100	15421	0
scpb2	22	22	22	22	22	22	22	100	6674	0
scpb3	22	22	22	22	22	22	22	100	32150	0
scpb4	23	22	22	22	22	22	22	100	97884	0,02
scpb5	23	22	22	22	22	22	22	100	32467	0

ТАБЛИЦА 4. Результаты решения задач C – E, NRE – -NRH

Задача	Best[1]	Best[2]	Best[3]	Best[4]	Best[5]	BFS	Our best	nbest	ncalc	mintime
1	2	3	4	5	6	7	8	9	10	11
scpc1	45	44	43	43	43	43	43	100	213076	0,02
scpc2	45	44	44	43	43	43	43	100	174729	0,05
scpc3	45	44	43	43	43	43	43	100	303836	0,02
scpc4	46	44	43	43	43	43	43	100	144943	0,05
scpc5	45	44	44	43	43	43	43	100	467337	0,05
scpd1	26	25	25	25	25	25	24	100	900350	0,56
scpd2	25	25	25	25	25	25	24	13	9373997	12,95
scpd3	25	25	25	25	24	24	24	6	9716148	20,13
scpd4	26	25	25	25	25	25	24	15	9491241	11,31
scpd5	26	25	25	25	25	25	24	8	9635568	11,67
scpe1	5	5	5	5	5	5	5	100	345	0
scpe2	5	5	5	5	5	5	5	100	111	0
scpe3	5	5	5	5	5	5	5	100	100	0
scpe4	5	5	5	5	5	5	5	100	146	0
scpe5	5	5	5	5	5	5	5	100	100	0
scpnre1	17	17	18	17	17	17	16	54	6998688	18,25
scpnre2	17	17	18	17	17	17	16	27	8565428	34,95
scpnre3	17	17	18	17	17	17	16	31	8527300	8
scpnre4	17	17	17	17	17	17	16	29	8409487	10,81
scpnre5	17	17	17	17	17	17	17	100	5135	0
scpnrf1	10	10	11	10	10	10	10	100	11411	0
scpnrf2	11	10	11	10	10	10	10	100	13123	0,03
scpnrf3	11	10	11	10	10	10	10	100	11771	0,03
scpnrf4	11	10	11	10	10	10	10	100	12190	0,03
scpnrf5	11	10	10	10	10	10	10	100	11626	0,03
scpnrg1			63	62	61	61	60	4	9777462	9,73
scpnrg2			61	62	62	61	60	9	9571109	63,64
scpnrg3			62	62	62	62	61	62	6320993	0,86
scpnrg4			63	62	62	62	61	34	8225109	15,64

Окончание табл. 4

1	2	3	4	5	6	7	8	9	10	11
scpnrg5			63	62	62	62	61	23	8557499	1,1
scpnrh1			35	34	34	34	33	4	9896235	406,75
scpnrh2			36	34	34	34	33	1	9965981	322,19
scpnrh3			36	34	34	34	33	5	9763561	180,81
scpnrh4			35	34	34	34	33	2	9928542	286,73
scpnrh5			36	34	34	34	34	100	347194	0

Заключение. С помощью предложенного в данной работе алгоритма для девятнадцати задач − scpa2-scpa3, scpd1-scpd2, scpd4-scpd5, scpnre1-scpnre4, scpnrg1-scpnrg5 и scpnrh1-scpnrh4 найдены новые рекорды. Для всех остальных задач получены известные рекорды. Это говорит о высокой робастности, потенциале алгоритма IteratedRandomLocalSearchMG относительно получения решений рекордного качества. О скорости работы алгоритма можно судить с 11 колонки табл. 3, 4. Как отмечалось в [8], учитывая развитие многопроцессорных комплексов при многократном решении задачи, следует обращать внимание не на средние показатели алгоритма, а на минимальное время решения задачи и частоту нахождения лучших решений. Приведенные в колонках 9, 11 табл. 3, 4 данные говорят о том, что разработанный алгоритм удовлетворяет современным требованиям. Все тестовые задачи он бы решил на 100-процессорном комплексе за 1423,67 секунд, используя распараллеливание копий алгоритма.

В.П. Шило

РОЗВ'ЯЗАННЯ ЗАДАЧІ ПРО ПОКРИТТЯ МІНІМАЛЬНОЇ ПОТУЖНОСТІ

Робота присвячена розв'язанню *NP*-важкої задачі про покриття мінімальної потужності (MCSCP), що має численні застосування, – найбільш складному підкласу задач про покриття. Розглянуто кращі відомі алгоритми розв'язання цієї задачі. Запропоновано та досліджено новий випадковий алгоритм повторного локального пошуку, що використовує адаптивне настроювання повторності та модифіковану цільову функцію. Наведено результати експериментальних розрахунків, які показали переваги запропонованого алгоритму над відомими кращими алгоритмами. За допомогою розробленого алгоритму знайдено дев'ятнадцять нових рекордних розв'язків.

V.P. Shylo

RANDOM ITERATED LOCAL SEARCH ALGORITHM FOR MINIMUM CARDINALITY SET COVERING PROBLEM USING MODIFIED OBJECTIVE FUNCTION

In this paper the Minimum Cardinality Set Covering Problem (MCSCP) is considered. The MCSCP is a *NP*-hard problem with a wide range of practical applications. MCSCP is the most complex subclass of the Set Covering Problem. We propose and investigate a new random algorithm with an adaptive iterative tuning based on the iterated local search. Our approach introduces a modifying objective function, which significantly improves the characteristics of the algorithm. The results of

extensive computational experiments reveal a superior performance when compared with the-state-of-the-art algorithms. The proposed approach improves the best existing solutions for 19 benchmark instances widely used in the literature.

1. *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.
2. *Bautista J., Pereira J.* A GRASP algorithm to solve the unicost set covering problem // Computers & Operations Research. – 2007. – **34**(10).– P. 3162 – 3173.
3. *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.
4. *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.
5. *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.
6. *Шило В.П.* Алгоритм случайного повторного локального поиска решения задачи о покрытии минимальной мощности // Журнал обчислювальної та прикладної математики. – 2013. – № 2 (112). – С. 52 – 59.
7. *Alimonti P.* New local search approximation techniques for maximum generalized satisfiability problems // Inform. Proc. Lett. – 1996. – **57**. – P. 151–158.
8. *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 23.05.2013

Об авторе:

Шило Владимир Петрович,
доктор физико-математических наук, ведущий научный сотрудник
Института кибернетики имени В.М. Глушкова НАН Украины.
E-mail: v.shylo@gmail.com