Журнал обчислювальної та прикладної математики **2013, №2(112)** Journal of Computational & Applied Mathematics

УДК 519.854

АЛГОРИТМ СЛУЧАЙНОГО ПОВТОРНОГО ЛОКАЛЬНОГО ПОИСКА РЕШЕНИЯ ЗАДАЧИ О ПОКРЫТИИ МИНИМАЛЬНОЙ МОЩНОСТИ

В. П. Шило

Резюме. В статье рассмотрена задача о покрытии минимальной мощности (MCSCP), которая является *NP*-трудной и имеет многочисленные практические приложения. MCSCP — наиболее сложный подкласс задач о покрытии. Рассмотрены лучшие известные алгоритмы для решения этой задачи. Предложен и исследован новый случайный алгоритм повторного локального поиска, который использует адаптивную настройку повторности. Приведены результаты обширного вычислительного эксперимента, которые показали преимущества предложенного алгоритма над известными лучшими алгоритмами. В ходе вычислительного эксперимента предложенным алгоритмом были найдены два новых рекордных решения.

1. Введение

Задача о покрытии (SCP) является одной из старейших и исследованных задач дискретной оптимизации. Пусть задано множество $M, m = |M|$ — мощность множества M, и n его подмножеств $S_j \subseteq M, j \in N = \{1, ..., n\}$. Семейство F подмножеств S_j будем называть допустимым, если каждый элемент множества M принадлежит хотя бы одному подмножеству S_j, входящему в семейство F: $\forall i \in M\ \exists S_j : S_j \subset F, i \in S_j$. Будем считать, что каждому подмножеству S_j приписан неотрицательный вес $w_j \geq 0, j = 1, ..., n$, а вес любого семейства равен сумме весов входящих в него подмножеств. Задача о покрытии заключается в отыскании допустимого семейства подмножеств F с минимальным весом.

Математическая модель поставленной задачи SCP описывается следующей задачей булева линейного программирования:

минимизировать

$$w(x) = \sum_{j=1}^{n} w_j x_j \tag{1}$$

при ограничениях

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

$$x_j = 0 \vee 1, j = 1, 2, ..., m. \tag{3}$$

Переменная x_j равняется 1, если подмножество S_j входит в семейство F, и нулю — в противном случае. В матрице покрытия A коэффициент a_{ij} равен 1, если элемент i принадлежит подмножеству S_j и равен нулю в противном случае. Как видно, i-я строка матрицы A связана с элементом i множества M, а j-й столбец — с подмножеством S_j. Если все веса подмножеств одинаковые, то задача SCP называется задачей о покрытии минимальной мощности (задача MCSCP) и считается, что $w_j = 1, j = 1, ..., n$. Она является наиболее трудной для решения. Именно эту задачу мы будем рассматривать в данной статье.

Задача о покрытии минимальной мощности имеет множество важных практических приложений. К ним относятся задачи планирования, тестирования, построения оптимальных логических схем, поиска компьютерных вирусов, балансировки сборочной линии, поиска информации и многие другие. Кроме того, во многих практических приложениях (планирование летных экипажей, политических округов, охраны природы и т.д.) относительные изменения соответствующих весов могут быть достаточно малы, чтобы оправдать использование рассматриваемой модели. И, наконец, очень часто алгоритмы для задачи о покрытии минимальной мощности входят в состав алгоритмического обеспечения современных компьютерных технологий.

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

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

2. Алгоритм случайного локального поиска

Для разработки нового алгоритма нами был взят за основу алгоритм, предложенный в работе [4]. Это объясняется, с одной стороны, его высокой эффективностью и, с другой, определенными сомнениями в целесообразности и продуманности некоторых его составляющих. Предложена следующая схема алгоритма случайного локального поиска.

```
 1: RandomLocalSearch (F)
 2: Calc(n_control,n_cover,F)
 3: while (F не является покрытием) do
 4:    Формирование множества Max_cover
 5:    S_j ←RandomSelectElement(Max_cover)
 6:    F = F ∪ S_j
 7:    Calc(n_control,n_cover,F)
 8:    Формирование множества Redundant
 9:    while (Redundant не пусто) do
10:       S_j ← RandomSelectElement(Redundant)
11:       F = F\S_j
12:       Calc(n_control,n_cover,F)
13:       Формирование множества Redundant
14:    end while
15:    Формирование множества Cand_Redundant
16:    if ( Cand_Redundant не пусто ) then
17:       S_j ←RandomSelectElement(Cand_Redundant)
18:       goto line 6
19:    end if
20: end while
21: end
```

В строках 2, 7, 12 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, покрываемых только подмножеством S_j, а величина n_cover_j — числу 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. Подмножество S_j из множества *Max_cover* с одинаковой вероятностью случайно выбирается в строке 5. 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.

Рассмотрим отличия предложенного нами алгоритма RandomLocalSearch (RLS) от алгоритма EG [4]. Во-первых, не используются оценочные функции при формировании множеств *Redundant* и *Cand_Redundant.* Их применение всегда увеличивает объем вычислений и не всегда приводит к повышению качества решений. Более того, возможное уменьшение мощности этих множеств может привести к снижению качества решений. Во-вторых, не используется процедура optimize(F), объем вычислений которой порядка $O(mn)$. Вместо этого введен блок (строки 15–19), позволяющий выбирать вместо жадного хода ход, после выполнения которого возникает непустое множество *Redundant.*

3. Алгоритм случайного повторного локального поиска

Ниже приведена схема предлагаемого случайного повторного алгоритма локального поиска IteratedRandomLocalSearch (IRLS).

```
 1: IteratedRandomLocalSearch (F)
 2: n_delete=initn_delete; BestF = {1, 2, ..., n}
 3: for  nat=1 to maxnat do
 4:    F = ∅
 5:    RandomLocalSearch(F)
 6:    MinF=F; nbad=0
 7:    for  niter=1 to maxniter do
 8:       F= MinF
 9:       Случайное удаление n_delete подмножеств из F
10:       RandomLocalSearch(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) then
17:          if (nbad>ubad) then n_delete= n_delete-1
18:          if (nbad<lbad) then n_delete= n_delete+1
19:          nbad=0
20:       end if
21:    end for
22: end for
23: end
```

Блок (строки 16–20) 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 нами выбрана схема повторности, отличная от предложенной в [4]. Там каждое подмножество удаляется с одинаковой вероятностью, поэтому количество удаленных подмножеств является случайной величиной. К тому же, в IteratedRandomLocalSearch осуществлена адаптивная настройка n_delete, что, по нашему мнению, является преимуществом.

4. Результаты вычислительного эксперимента

Для сравнительного исследования эффективности разработанного и существующих алгоритмов были проведены экспериментальные расчёты по решению 70 случайно сгенерированных тестовых задач 4—6, A—E и NRE—NRH большой размерности. Все эти задачи доступны в OR-library по адресу (http://people.brunel.ac.uk/ mastjjb/jeb/orlib/scpinfo.html). Параметры алгоритма IRLS указаны в табл. 1. Заметим, что в отличие от алгоритмов [4, 5], которые использовали настройку на задачи, все параметры IRLS не изменялись при проведении вычислительных экспериментов.

Таблица 1. Параметры алгоритма IteratedRandomLocalSearch

Параметр	maxnat	maxniter	initn_delete	ntune	lbad	ubad
Значение	100	3000	8	27	18	24

В табл. 2, 3 содержатся сводные результаты по решению тестовых задач.

Таблица 2. Сводные результататы 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	401356	0,00
scp42	38	37	37	37	37	37	37	100	41356	0,00
scp43	40	38	38	38	38	38	38	100	83460	0,00
scp44	41	39	38	39	38	38	38	88	13004676	0,03
scp45	40	38	38	38	38	38	38	100	338228	0,00
scp46	40	38	37	37	37	37	37	100	3698068	0,05
scp47	41	38	38	38	38	38	38	100	1505461	0,00
scp48	40	38	38	37	37	37	37	100	3979361	0,05
scp49	40	38	38	38	38	38	38	100	419429	0,00
scp410	41	38	38	38	38	38	38	100	1763259	0,06
scp51	35	35	35	34	34	34	34	99	4531224	0,02
scp52	35	34	35	34	34	34	34	100	977696	0,05
scp53	36	35	34	34	34	34	34	100	337507	0,01
scp54	36	34	34	34	34	34	34	100	414099	0,02
scp55	36	34	34	34	34	34	34	100	499814	0,01
scp56	36	34	34	34	34	34	34	100	973126	0,03
scp57	35	34	34	34	34	34	34	100	182193	0,00
scp58	37	35	34	34	34	34	34	100	2863216	0,05
scp59	36	36	35	35	35	35	35	100	290155	0,00
scp510	36	35	34	34	34	34	34	100	3068748	0,14
scp61	21	21	21	21	21	21	21	100	45580	0,00
scp62	21	20	21	20	20	20	20	100	599835	0,00
scp63	21	21	21	21	21	21	21	100	24441	0,00
scp64	22	21	21	21	20	20	20	97	8185640	0,00
scp65	22	21	21	21	21	21	21	100	104370	0,00
scpa1	40	39	39	39	39	39	39	100	420737	0,02
scpa2	41	39	39	39	39	39	**38**	9	37472276	6,64
scpa3	40	39	39	39	39	39	39	100	374466	0,02
scpa4	40	38	38	38	37	37	37	37	30717609	1,33
scpa5	40	39	38	38	38	38	38	100	2244857	0,14
scpb1	23	22	22	22	22	22	22	100	121484	0,02
scpb2	22	22	22	22	22	22	22	100	57673	0,00
scpb3	22	22	22	22	22	22	22	100	156793	0,00
scpb4	23	22	22	22	22	22	22	100	439091	0,06
scpb5	23	22	22	22	22	22	22	100	219039	0,02

Каждая тестовая задача решалась 100 раз. Во второй — шестой колонках таблиц приведены рекорды соответственно из работ [1–5]. В седьмой колонке содержатся известные рекорды. В восьмой приведены рекорды, полученные предложенным алгоритмом

IteratedRandomLocalSearch, в девятой колонке — количество найденных этим алгоритмом рекордов (из 100).

Таблица 3. Сводные результататы 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	2531955	0,11
scpc2	45	44	44	43	43	43	43	100	2663363	0,25
scpc3	45	44	43	43	43	43	43	100	3704540	0,61
scpc4	46	44	43	43	43	43	43	100	2358900	0,20
scpc5	45	44	44	43	43	43	43	99	6258633	0,95
scpd1	26	25	25	25	25	25	**24**	98	8956646	0,69
scpd2	25	25	25	25	25	25	25	100	120212	0,01
scpd3	25	25	25	25	24	24	24	4	35591521	156,78
scpd4	26	25	25	25	25	25	25	100	341016	0,00
scpd5	26	25	25	25	25	25	25	100	364259	0,01
scpe1	5	5	5	5	5	5	5	100	186	0,00
scpe2	5	5	5	5	5	5	5	100	133	0,00
scpe3	5	5	5	5	5	5	5	100	100	0,00
scpe4	5	5	5	5	5	5	5	100	181	0,00
scpe5	5	5	5	5	5	5	5	100	100	0,00
scpnre1	17	17	18	17	17	17	17	100	90243	0,00
scpnre2	17	17	18	17	17	17	17	100	53808	0,00
scpnre3	17	17	18	17	17	17	17	100	75672	0,00
scpnre4	17	17	17	17	17	17	17	100	50345	0,00
scpnre5	17	17	17	17	17	17	17	100	95983	0,00
scpnrf1	10	10	11	10	10	10	10	100	5519	0,00
scpnrf2	11	10	11	10	10	10	10	100	94740	0,02
scpnrf3	11	10	11	10	10	10	10	100	77209	0,02
scpnrf4	11	10	11	10	10	10	10	100	73939	0,02
scpnrf5	11	10	10	10	10	10	10	100	104399	0,02
scpnrg1			63	62	61	61	61	59	26018437	5,42
scpnrg2			61	62	62	61	61	71	23908902	13,52
scpnrg3			62	62	62	62	62	100	2258028	0,56
scpnrg4			63	62	62	62	62	100	2889478	0,76
scpnrg5			63	62	62	62	62	100	2958199	0,89
scpnrh1			35	34	34	34	34	100	990892	0,42
scpnrh2			36	34	34	34	34	100	756843	0,27
scpnrh3			36	34	34	34	34	100	694191	0,81
scpnrh4			35	34	34	34	34	100	775551	0,53
scpnrh5			36	34	34	34	34	100	751407	0,00

В десятой колонке содержится количество вызовов RLS за 100 решений алгоритмом IteratedRandomLocalSearch соответствующей задачи. В последней колонке приводится наименьшее время (в сек.) поиска решения одной из 100 задач предложенным алгоритмом. Алгоритм IteratedRandomLocalSearch реализован на языке C++, все вычислительные эксперименты проводились с использованием PC Intel®Core QUAD CPU Q9550 2.83GHz и 8.0GB оперативной памяти.

5. Заключение

Для двух задач scpa2 и scpd1 найдены новые рекорды. Для всех остальных задач повторены прежние рекорды с меньшими затратами времени. Все это говорит о высокой робастности, потенции алгоритма IteratedRandomLocalSearch к получению решений рекордного качества. О скорости работы алгоритма можно судить по последней колонке табл. 4, 5. Как отмечалось в работе [6], в связи с развитием 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. Данные, приведенные в колонках 9, 11 табл. 4, 5 говорят о том, что разработанный алгоритм удовлетворяет современным требованиям. Все тестовые задачи он бы решил на 100-процессорном комплексе за 196,95 секунд, используя распараллеливание копий алгоритма.

Литература

1. Grossman T. Computational experience with approximation algorithms for the set covering problem / T. Grossman, A. Wool // European Journal of Operational Research. — 1997. — 101. — № 1. — P. 81–92.
2. Bautista J. A GRASP algorithm to solve the unicost set covering problem / J. Bautista, J. Pereira // Computers & Operations Research. — 2007. — 34. — P. 3162–3173.
3. Kinney G. W. A Reactive Tabu Search algorithm with variable clustering for the Unicost Set Covering Problem / G. W. Kinney, J. W. Barnes, B. W. Colletti // International Journal of Operational Research. — 2007. — 2. — № 2. — P. 156–172.
4. Marchiori E. An iterated heuristic algorithm for the set covering problem / E. Marchiori, A. G. Steenbeek // In proceeding of: Algorithm Engineering, 2nd International Workshop (WAE'98, Saarbrucken, Germany, August 20–22, 1998). — 1998. — P. 155–166.
5. Musliu N. Local search algorithm for unicost set covering problem / N. Musliu // In Advances in Applied Artificial Intelligence: Lecture Notes in Artificial Intelligence. — Berlin, Heidelberg: Springer. — 2006. — 4031. — P. 302–311.
6. Шило В. П. Решение задачи булева квадратичного программирования без ограничений методом глобального равновесного поиска / В. П. Шило, О. В. Шило // Кибернетика и системный анализ. — 2011. — № 6. — С. 68–78.

Институт кибернетики им. В.М. Глушкова НАН Украины, проспект Академика Глушкова, 40, Киев–187, ГСП, 03680, Украина.

Поступила 21.05.2013