Improve the 3-flip Neighborhood Local Search by Random Flat Move for the Set Covering Problem

Chao Gao, Thomas Weise and Jinlong Li

School of Computer Science and Technology,
University of Science and Technology of China (USTC),
Hefei, China
chao.gao.ustc@gmail.com
{tweise,jlli}@ustc.edu.cn

Abstract. The 3-flip neighborhood local search (3FNLS) is an excellent heuristic algorithm for the set covering problem which has dominating performance on the most challenging crew scheduling instances from Italy railways. We introduce a method to further improve the effectiveness of 3FNLS by incorporating random flat move to its search process. Empirical studies show that this can obviously improve the solution qualities of 3FNLS on the benchmark instances. Moreover, it updates two best known solutions within reasonable time.

Keywords: Set Covering Problem, 3-flip Local Search, Random flat move

1 Introduction

The set covering problem (SCP) is a prominent combinatorial optimization task which asks to find a collection of subsets to cover all the elements at the minimal cost. Formally, it is defined as: Given a universal set E which contains m elements, n subsets which are $S_1 \cup S_2 \cup ... \cup S_n = E$ and each subset has a cost, find a set of subsets F at the minimal total cost but still cover all elements in X, i.e., $\bigcup_{s \in F} s = E$. In literature, the SCP is generally described as integer linear programming form, as follows:

$$\min \sum_{j=1}^{n} c_j \cdot x_j \tag{1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} \cdot x_j \geq 1, \quad i \in M, \text{ where } M = \{1, 2, \ldots, m\},$$

$$x_j \in \{0,1\}, \quad j \in N, \text{ where } N = \{1,2,\ldots,n\}$$

The zero-one matrix $A = \{a_{ij}\}_{m\times n}$ represents the problem instance, and $a_{ij} = 1$ means subset S_j is able to cover the element i. For each variable $x_j = 1$ indicates that S_j is selected, and 0 otherwise. In literature, the SCP is also viewed as to find a set of columns to cover all the rows at the minimal total cost, where N is the set of all columns that each has a cost, M is the set of rows that need to be covered.

The SCP is NP-hard in the strong sense [2], thus no complete algorithms with polynomial time complexity are known for SCP. It has many real-world applications, such as crew scheduling from bus, railway and airline systems[3–5]. A number of algorithms have been proposed for SCPs, among them the exact algorithms are shown not suitable to tackle large-scale problems because of their untolerable time consumption [6]. Therefore, approximation and heuristic algorithms are also widely studied by researchers both from the operations research and artificial intelligence communities. A variety of approximation or heuristic algorithms have been proposed, including the greedy algorithm [7], randomized greedy procedures [8, 9], simulated annealing [10], genetic algorithm [11], ant colony optimization algorithm [12, 13], artificial bee colony algorithm [14] and the Meta-RaPS approach by Lan et al. [15].

However, among these heuristics, only a few of them are able to tackle the very large-scale instances from the Italy railways [4, 16], which contain to up millions of columns and thousands of rows. These instances, firstly distributed by a FASTER competition in 1994, are generally referred as the most challenging SCP instances from the OR-Library [17]. Caprara et al. proposed a Lagrangian-based heuristic (CFT) and a greedy procedure and obtained very impressive results on this set of instances and other random generated benchmark instances [16]. They won the first prize of the competition. Later, the 3-flip neighborhood local search (3FNLS) proposed by Yagiura et al. [18] is able to surpass CFT on these crew scheduling instances. We emphasize that CFT is essential to the success of 3FNLS, for 3FNLS also uses the same subgradient method implementation proposed in CFT to solve the Lagrangian relaxation of SCP.

In this paper, we introduce a search strategy named random flat move to further improve the effectiveness of 3FNLS. The experimental results show that it is effective, especially on the very large-scale instances, for it generally produces better solution qualities than the original 3FNLS within the same time limits. Further, it discovered two new best known solutions for the largest two instances within reasonable time.

The rest of this paper is organized as follows: In Section 2, we give description of the 3FNLS algorithm. Then our improvement strategy is described in Section 3. Section 4 is the computational results and comparisons. Conclusion and future work are finally presented in Section 5.

2 3FNLS Review

To make this article self-contained, it is necessary to introduce the basic concepts of 3NFLS before presenting our improvement method. However, because the ideas in 3FNLS are complicated, we suggest the readers to refer [18] for the details. In this section, we only provide brief introduction to the key procedures.

The overall procedure of 3FNLS is shown as Algorithm 1, in which the *SUBGRADIENT* method is used to solve the Lagragian dual relaxation of SCP to obtain a Lagrangian multiplier vector $\boldsymbol{u}$, and then for each column j a reduced cost is calculated as $c_j(\boldsymbol{u}) = c_j - \sum_{i \in M} a_{ij} \cdot u_i$. Because of the *integrality property*, an optimal solution $\boldsymbol{u}^*$ to the dual of LP relaxation of SCP is also the optimal solution to the Lagrangian dual problem [19]. For a good Lagrangian multiplier vector $\boldsymbol{u}$, the reduced cost $c_j(\boldsymbol{u})$ can give reliable information for the goodness of column j, because each column j with $x_j = 1$ in an optimal solution tends to have small $c_j(\boldsymbol{u})$ value.

From Algorithm 1 we can see that in 3FNLS, initially, the candidate solution $\boldsymbol{x}$ is obtained greedily, and *UB* is set to *cost(**x**)*. 3FNLS calls the subgradient method many times. At the first time, $u_i^0 = \min\{c_j/I_j | i \in I_j\}$ ($\forall j \in N$ and I_j is the set of rows j covers), otherwise, $\boldsymbol{u}^0$ is set to $\boldsymbol{u}^+$, where $\boldsymbol{u}^+$ is the Lagrangian multiplier vector obtained by the first call. Let $\boldsymbol{x}^*$ be the stored best solution during the search, then *UB* is always maintained as *cost(**x***)*.

An essential feature of 3FNLS is its problem size reduction heuristic, which is indispensable when facing the very large-scale instances, because directly local search on the whole columns is quite expensive. At first, the selected columns are determined by columns with the first $\alpha \cdot min\_free \cdot$ small $c_j(\boldsymbol{u})$, where min_free and α are program parameters which are set to 100 and 3, respectively. This is corresponding to Line 4 in Algorithm 1.

Whenever some iterations of local search are finished, the selected columns are adjusted by randomly fixing some 'good columns' to 1 to call the subgradiment method, and then some new columns with $c_j(\boldsymbol{u}') < 0$ are added to the selected column set, where $\boldsymbol{u}'$ is the new Lagrangian multiplier. Let $N_{selected}$ be the selected columns which the local search is conducted on, N_1 be the set of columns in $\boldsymbol{x}$ fixed to 1 (not permitted to flip to 0 during the next period of local search). This is shown as Line 31 in Algorithm 1.

In Algorithm 1, *r-flip is possible*($r \leq 3$) means that there is at least one *r-flip* to decrease a penalty function defined in 3FNLS. At most 3-flip is permitted by 3FNLS and when there are no flips within 3-flip to find to reduce the penalty function value, the penalty weights of rows are updated. The details of how the penalty weights are updated is complicated. Usually, the penalty weights are updated by an increasing manner with the information provided by the last flip. Only when a certain rule is violated, then the penalty weights of rows will be decreased to make sure that there are *possible flips* again. The interested readers are suggested to refer [18] for details.

Algorithm 1: The 3FNLS algorithm

Input: A SCP instance
Output: Best found solution $\boldsymbol{x}^*$

Initiate candidate solution $\boldsymbol{x}$ greedily and $UB \leftarrow$ *cost(x)*;
Initiate $\boldsymbol{u}^0$, intiate penalty weights for all rows in M ;
$\boldsymbol{u}^+ \leftarrow SUBGRADIENT(UB, \boldsymbol{u}^0)$;
Select a subset of columns to $N_{selected}$ based on their reduced cost;
$trial \leftarrow 1$;
while *Time not exceeded* **do**
    while *Time not exceeded* **do**
        if *one flip is possible* **then** process one flip;
        if *better solution is detected* **then**
            update $\boldsymbol{x}^*$ and UB;
        end
        continue;
        ;
        if *two flip is possible* **then** process two flip;
        if *better solution is detected* **then**
            update $\boldsymbol{x}^*$ and UB;
        end
        continue;
        ;
        if *three flip is possilbe* **then**
            if *better solution is detected* **then**
                update $\boldsymbol{x}^*$ and UB;
            end
            process three flip;
            continue;
        else
            break;
        end
    end
    update penalty weights of rows;
    if *penalty weigts is updated by decrease* **then**
        if $\boldsymbol{x}^*$ *has not been updated for at least mintr_lsl iterations* **then**
            modify variable fixing;
        end
    end
    $trial \leftarrow trial + 1$;
end

3 Random Flat Move for 3FNLS

From Algorithm 1, we can see that 3FNLS can be viewed as a multi-start algorithm. Each period of local search starts from fixing some variables in $N_{selected}$ to 1 and calling the subgradient method, and then based on the information from solving Lagrangian relaxation, it continues to flip variables to decrease the *pcost* of the candidate solution until the stop condition is reached. Because the N_1 is determined by the stored best solution and current candidate solution, thus the quality of the stored best solution can directly influence the performance of the following period of local search.

The problem size reduction heuristic (or variable fixing) divides all the columns into two parts, in which the $N_{selected}$ represents the columns selected into the search. Then, at the beginning of each period of local search, it further reduces the search size by fixing some columns in $N_{selected}$ as 1, which means they are not permitted to be flipped to 0 the following period of local search. It is easy to see that the correctness of selecting columns to fix to 1 is crucial to the search, because wrong fixing could drastically mislead the search. As the columns in N_1 is also selected heuristically, it is obvious that the correctness of the selection of columns to fix to 1 can not be guaranteed.

However, we observe that 3FNLS only updates the stored best solution when another better solution is detected; i.e., only when $cost(\boldsymbol{x}) < UB$ and $\boldsymbol{x}$ is feasible. In the variable fixing modification algorithm, the N_1 is chosen from the intersection between the current candidate solution $\boldsymbol{x}$ and the stored best solution $\boldsymbol{x}^*$ which could be a no-promising local optimum. Therefore we propose a simple search strategy to 3FNLS, which is randomly update the $\boldsymbol{x}^*$ when $\boldsymbol{x}$ becomes feasible and $cost(\boldsymbol{x}) = UB$ by a probability. The idea of random flat move is that the stored best solution could be on a local optima plateau, neutral walking at a probability may lead to a better chance for finding a portal. The modification of the local search of 3FNLS is as below:

Algorithm 2: Random flat move for 3FNLS

```
1  if r-flip is possible then
2  |  if x is feasible then
3  |  |  if UB > cost(x) then
4  |  |  |  x* ← x;
5  |  |  |  UB ← cost(x);
6  |  |  end
7  |  |  if UB = cost(x) and rand(0,1) > prfm then
8  |  |  |  x* ← x;
9  |  |  end
10 |  end
11 end
```

In Algorithm 2, *r-flip* refers to one, two or three flip in Algorithm 1, and *prfm* is the probability of the flat move. We set *prfm* to 0.5 in our implementation.

4 Experimental Results

In order to show the effectiveness of our search strategy to 3FNLS, we test the modified algorithm on instances from the OR-Library [17], which contains the randomly generated instances as well as the very large-scale crew scheduling instances from Italy railways.

4.1 The Benchmark Instances

We test 3FNLS on 4 type random instances and 7 challenging instances from Italy railways, shown in Table 1. For type NRE to NRH, each type contains 5 instances. The density is the number of non-zero entries in the problem instance matrix. The optima of these instances are all unknown.

Table 1. Details of the test instances

Instance type	m	n	Range of cost	Density(%)	Number of instances
NRE	500	5000	1–100	10	5
NRF	500	5000	1–100	20	5
NRG	1000	10000	1–100	2	5
NRH	1000	10000	1–100	5	5
RAIL507	507	63009	1– 2	1.2	1
RAIL516	516	47311	1– 2	1.3	1
RAIL582	582	55515	1– 2	1.2	1
RAIL2536	2536	1081841	1– 2	0.4	1
RAIL2586	2586	920683	1– 2	0.4	1
RAIL4284	4284	1092610	1– 2	0.2	1
RAIL4872	4872	968672	1– 2	0.2	1

For Table 1, we can see that one obvious characteristic of these instances is that they all have many more columns (n) than rows (m), especially for the crew scheduling instances from railways, with up to 1 million columns, whereas no more than 5 thousands rows.

4.2 Comparison Results

The source code of 3FNLS is provide by the author Mutsunori Yagiura, written in C. Our improvement algorithm with random flat move (3FNLS-rfm) is directly modified upon the source of 3FNLS. Both the two algorithms are compiled in g++ with -O2 option, run on the same Intel(R) Xeon(R) E5450 3.00 GHz CPU machine with 16 GB RAM, under 64-bit Linux system. Due to the randomness of the algorithms, for each instance, 10 independent runs are performed with random seeds from 11 to 20. All times are measured in CPU seconds in our experiments.

The results are reported as the best solutions (*best*) obtaind from the 10 runs, the average solution of the 10 runs (*mean*), number of runs that the best is detected (*#best*), and the average times over the runs that detecting the best (*Avg Time*). The time limit for instances from NRE to NRH is set to 20 seconds, RAIL507, RAIL516 and RAIL582 is set to 200 seconds, and RAIL2536, RAIL2586, RAIL4284 and RAIL4872 is set to 2000 seconds. The comparison results are shown in Table 2.

Table 2. Comparison results of 3FNLS and 3FNLS-rfm on benchmark instances

Instance	BKS	3FNLS				3FNLS-rfm			
		best	mean	#best	Avg Time	best	mean	#best	Avg Time
NRE1	29	29	29	10	0.24	29	29	10	0.24
NRE2	30	30	30.7	3	6.74	30	30.2	8	12.89
NRE3	27	27	27	10	0.30	30	30	10	0.31
NRE4	28	28	28	10	0.24	28	28	10	0.23
NRE5	28	28	28	10	0.25	28	28	10	0.25
NRF1	14	14	14	10	0.36	14	14	10	0.36
NRF2	15	15	15	10	0.31	15	15	10	0.31
NRF3	14	14	14	10	0.30	14	14	10	0.31
NRF4	14	14	14	10	0.28	14	14	10	0.27
NRF5	13	13	13	10	0.54	13	13	10	0.27
NRG1	176	176	176	10	0.65	176	176	10	0.62
NRG2	154	154	154	10	0.86	154	154	10	1.25
NRG3	166	166	166	10	3.38	166	166	10	5.94
NRG4	168	168	168	10	0.99	168	168	10	1.27
NRG5	168	168	168	10	0.86	168	168	10	1.29
NRH1	63	63	63	10	1.82	63	63	10	2.47
NRH2	63	63	63	10	1.94	63	63	10	2.04
NRH3	59	59	59	10	1.17	59	59	10	0.63
NRH4	58	58	58	10	0.97	58	58	10	0.89
NRH5	55	55	55	10	0.60	55	55	10	0.59
RAIL507	174	174	174	10	26.45	174	174	10	9.70
RAIL516	182	182	182	10	1.67	182	182	10	1.64
RAIL582	211	211	211	10	2.83	211	211	10	2.31
RAIL2536	690[a]	691	691.3	7	709.83	**690**	690.2	9	910.80
RAIL2586	945[a]	946	947.0	2	1269.67	**945**	946.7	1	1002.79
RAIL4284	1064[a]	1063	1064.1	3	1350.27	**1062***	1063.3	2	1826.34
RAIL4872	1528[a]	1528	1530.0	2	1214.32	**1527***	1529.0	1	910.40
SUM	6136	6137	6143.1	237	4529.61	6133	6138.4	241	4596.41

[a] The BKSs of RAIL2536, RAIL2586, RAIL4284 and RAIL4872 are all previously found by 3FNLS, reported in [18].

[+] The better solution of 3FNLS-rfm is highlighted by boldface.

[+] The updated BKS of 3FNLS-rfm is with a following asterisk.

From Table 2, we can see that for instances from NRE to NRH, both 3FNLS and 3FNLS can achieve the best known solution (BKS) within short times. The only instance they have not all success is NRE2, where the solution quality of

3FNLS-rfm is still better than 3FNLS. For the NRG type instances, the average times of 3FNLS are only slightly smaller than that of 3FNLS-rfm.

In Table 2, we highlight the better solutions of 3FNLS-rfm than 3FNLS in boldface, and the updated best known solutions with a following asterisk. It is easy to see that 3FNLS-rfm has 4 better best solutions than 3FNLS on the 7 railway instances. Moreover, 3FNLS-rfm also has better solution qualities than 3FNLS on the instance RAIL507, RAIL516 and RAIL582, because its average times are generally smaller than 3FNLS. For RAIL2536 and RAIL2586, the origianl 3FNLS fails to achieve the BKS of these two instances within 2000 seconds. For the RAIL4284 and RAIL4872, 3FNLS-rfm has updated the best known solutions of these two instances. The results in Table 2 show that 3FNLS-rfm can obviously improve the performance of 3FNLS, especially on the challenging large-scale railway problems, for it is always achieves better solution qualities, given the same time limits on the same machine.

5 Conclusion and future work

In this paper, we have reviewed the state-of-the-art 3-flip neighborhood local search (3FNLS) algorithm for the set covering problem. Through the analysis of the process of 3FNLS, we notice that it can be regarded as a multi-start algorithm, which starts each period of local search by solving the Lagrangian and fixing some variables to 1. However, 3FNLS does not allow the *stored best solution* to move on the other 'equal-best 'solutions which may be portal to better solutions. Therefore, we propose a random flat move strategy to 3FNLS to make sure that the *stored best solution* can be updated by a probability on the possible plateau.

The proposed strategy has been tested on instances from the OR-Library, and the computational comparison results show that our strategy can obviously improve the performance of 3FNLS on the very large-scale instances. Moreover, it has updated the best known solutions of the last two instances. Observing that the very large-scale railway instances are commonly regarded as most challenging in the SCP benchmark instances, we believe our improvement method should be worth of existing.

In this paper, the probability of our flat move is intuitively set to 0.5, which may not be the ideal value for this algorithm. In the future, further empirical studies will be conducted to explain the behaviors of 3FNLS with different flat move probabilities. The relationship between the effectiveness of 3FNLS and instance features is also worth for further study.

Acknowledgments The authors are grateful for M. Yagiura for providing their source of 3FNLS. This research work was partially supported by an EPSRC grant (No. EP/I010297/1), the Fundamental Research Funds for the Central Universities (WK0110000023), the National Natural Science Foundation of China under Grants 61150110488, the Special Financial Grant 201104329 from the China Postdoctoral Science Foundation, and the Chinese Academy of Sciences (CAS) Fellowship for Young International Scientists 2011Y1GB01.

References

1. Chao Gao, Thomas Weise, and Jinlong Li. Improve the 3-flip neighborhood local search by random flat move for the set covering problem. In Ying Tan, Yuhui Shi, and Carlos Artemio Coello Coello, editors, *Advances in Swarm Intelligence: Proceedings of the Fifth International Conference on Swarm Intelligence, Part 1*, volume 8794 of *Lecture Notes in Computer Science (LNCS)*, pages 27–35, Hefei, Anhui, China, October 17–20, 2014. Springer International Publishing.
2. Michael R Gary and David S Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness, 1979.
3. Egon Balas and Maria C Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44(6):875–890, 1996.
4. Sebastián Ceria, Paolo Nobili, and Antonio Sassano. A lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81(2):215–228, 1998.
5. Barbara M Smith and Anthony Wren. A bus crew scheduling system using a set covering formulation. *Transportation Research Part A: General*, 22(2):97–108, 1988.
6. Alberto Caprara, Paolo Toth, and Matteo Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1-4):353–371, 2000.
7. Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.
8. Francis J Vasko. An efficient heuristic for large set covering problems. *Naval Research Logistics Quarterly*, 31(1):163–171, 1984.
9. Thomas A Feo and Mauricio GC Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71, 1989.
10. Larry W Jacobs and Michael J Brusco. Note: A local-search heuristic for large set-covering problems. *Naval Research Logistics*, 42(7):1129–1140, 1995.
11. John E Beasley and Paul C Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404, 1996.
12. Lucas Lessing, Irina Dumitrescu, and Thomas Stützle. A comparison between aco algorithms for the set covering problem. In *Ant Colony Optimization and Swarm Intelligence*, pages 1–12. Springer, 2004.
13. Zhi-Gang Ren, Zu-Ren Feng, Liang-Jun Ke, and Zhao-Jun Zhang. New ideas for applying ant colony optimization to the set covering problem. *Computers & Industrial Engineering*, 58(4):774–784, 2010.
14. Shyam Sundar and Alok Singh. A hybrid heuristic for the set covering problem. *Operational Research*, 12(3):345–365, 2012.
15. Guanghui Lan, Gail W DePuy, and Gary E Whitehouse. An effective and simple heuristic for the set covering problem. *European journal of operational research*, 176(3):1387–1403, 2007.
16. Alberto Caprara, Matteo Fischetti, and Paolo Toth. A heuristic method for the set covering problem. *Operations research*, 47(5):730–743, 1999.
17. John E Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, pages 1069–1072, 1990.
18. Mutsunori Yagiura, Masahiro Kishida, and Toshihide Ibaraki. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, 172(2):472–499, 2006.
19. Marshall L Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, 27(1):1–18, 1981.

Reference


```
@inproceedings{GWL2014IT3FNLSBRFMFTSCP,
  author    = {Chao Gao and Thomas Weise and Jinlong Li},
  title     = {Improve the 3-flip Neighborhood Local Search by
               Random Flat Move for the Set Covering Problem},
  booktitle = {Advances in Swarm Intelligence: Proceedings of
               the Fifth International Conference on Swarm
               Intelligence, Part 1},
  editor    = {Ying Tan and Yuhui Shi and
               Carlos Artemio {Coello Coello}},
  series    = {Lecture Notes in Computer Science (LNCS)},
  volume    = {8794},
  publisher = {Springer International Publishing},
  address   = {Hefei, Anhui, China},
  pages     = {27--35},
  year      = {2014},
  month     = oct # {~17--20, },
  doi       = {10.1007/978-3-319-11857-4_4},
  isbn      = {978-3-319-11856-7},
},
```