Technical University of Denmark

42137 Optimization Using Metaheuristics

Set Cover Problem

Adaptive Large Neighborhood Search

Sefa Kayraklık s186295

Yaşar Harun KIVRIL s186296

29/04/2019

Contents

Introduction

Set cover problem is one of the classical questions in operations research. It is proven to be NP-complete by Karp in 1972 [1]. Therefore, even though the solver capabilities are improved with the new technologies, it takes a huge amount of time to solve large sized set cover problems. There are also non-unicost large sized set cover problems that couldn't solved to optimality with the solvers, yet. For this reason, many different heuristics are developed for the large sized problems.

In this report, a local search method and an adaptive large neighborhood search algorithm is introduced. At the beginning, TABU search is considered for the problem however, the size of the neighborhood grows exponentially as the neighborhood structure gets more complex. Then a simple neighborhood is searched, but the simplicity of neighborhood doesn't allow get good result. Therefore, it is decided to use ALNS to make the search in a larger neighborhood for more promising results.

The report also presents the parameter tuning of ALNS and the evaluation of the results using tuned parameters with the comparison of the known best solutions. At the end the performance of the method is discussed, and general conclusion is made.

Problem Description

The aim of the set cover problem is identifying the union of sub subsets $\boldsymbol{S} = \{1, 2, 3, \ldots m\}$ with minimum total cost that includes all elements in a given universe $\boldsymbol{U} = \{1, 2, 3, \ldots n\}$ where union of all subsets equals to $\boldsymbol{U}$. The integer linear problem of the problem is given below.

$$min \sum_{i \in S} c_i . x_i$$

$$\sum_{i \in S,\ j \in U} a_{ij} . x_i \geq 1$$

$$x_j \in \{0, 1\}$$

A feasible solution of the problem is a set of subsets such that union of all sets equals to universe.

Data Sets

The data sets used in report is taken from OR-Library [2]. As datasets SCPNGR1, SCPNGR2 with size 1000 x 10000, SCP49 and SCP410 with size 200 x 1000. The main aim is to approximate SCPNGRs because a standard IP solver cannot solve it in small amount of time. SCP4s are relatively

very easy to solve with an IP solver but it is used to see and consider the behavior of the algorithm in smaller examples. In the report, SCPNGR2 and SCP410 are chosen for parameter tuning, SCPNGR1 and SCP49 for testing. The best-known solutions for the datasets are given in the Table 1.

Problem number	Number of subgradient iterations	Optimal value	Total time Cray-1S seconds
4.9	385	641	6.3
4.10	131	514	1.8

Problem	Size	B-Lag	JB-Ann	BC-Gen	PD-Lag	Lower bounds
SCPNRG1	(1000 × 10,000)	184	179	176	176	159.5
SCPNRG2	(1000 × 10,000)	163	158	155	155	141.8

Table 1: Best-known solutions for the chosen datasets [7], [8]

Previous Research

In order to solve test cover problem many exact and heuristic approaches developed before. In their literature review Vijeyamurthy and Panneerselvam [3] mention dynamic programing, interior point, branch and bound based methods and some relaxations are used to solve the problem before. They also mention about lots of metaheuristic approach to the problem. In the review most of the researched are made by using direct or indirect genetic algorithms. Bautista and Pereria (2007) used GRASP (Greedy Randomized Adaptive Search Procedure) to solve unicost set cover problem.

Metaheuristic Description

General Description of Adaptive Large Neighborhood Search

The metaheuristic approach of adaptive large neighborhood search (ALNS) that was proposed by Pisinger and Røpke [4] is an extension of the large neighborhood search (LNS) that was invented by Shaw [5].

LNS tries to make an improvement in the solution of the problem by making local changes in the solution and obtaining the neighbors. The neighborhood is described in a way that each solution is established by a destroy and a repair method; namely, local changes are done by using destroy and repair methods. As the name of the algorithm indicates that the destroy and the repair method, which describe how the search is done should look for far away neighbors in order not to get stuck in the local optimum solutions. A destroy method takes the current solution and removes some part of the solution; on the other hand, a repair method takes the destroyed solution and adds some part to the solution in order to obtain a new solution. Thus, a destroy method moves the solution in the

infeasible region and a repair method moves the destroyed solution back to the feasible region. It is a proper option to use in the problems that have very large neighborhood structure that makes the local search methods inefficient in terms of calculation time. However, destroying and repairing a solution with the same method every time can easily lead to the local optima. Therefore, using an Adaptive LNS can robust getting stuck in the local optima.

Apart from the LNS, the ALNS uses more than one method to destroy and to repair the current solution. The selection of the different destroy and repair methods is in accordance with their probabilities which are updated as the algorithm runs. The adjustment of the probabilities that are equal initially is done by the following equation 1:

$$\begin{matrix} p_d^- = \lambda p_d^- + (1-\lambda)\omega \\ p_r^+ = \lambda p_r^+ + (1-\lambda)\omega \end{matrix} , equation\ 1$$

, where p_d^- is the probability to choose the dth destroy method, p_r^- is the probability to choose the rth repair method, $\lambda \in [0,1]$ is the parameter that determines how much featured the previous probability in comparison to ω which the rewards to the methods. ω is determined by the following expression:

$$\omega = \begin{cases} w_4\ if\ new\ solution\ is\ new\ global\ best \\ w_3\ if\ new\ solution\ is\ better\ than\ the\ current\ one \\ w_2\ if\ new\ solution\ is\ accepted \\ w_1\ if\ new\ solution\ is\ rejected \end{cases}$$, where $w_4 \geq w_3 \geq w_2 \geq w_1$ equation 2

The pseudo code for the ALNS is given in the following:

```
procedure ALNS ()
  Generate feasible solution x (PURE: can be done in many ways)
  x^b = x, p^- = (uniform), p^+ = (uniform)
  repeat
    select destroy and repair methods d (PURE: More than one method) and r using p^- and p^+
    x^t = r (d(x))
    if accept (x^t, x) then x = x^t
    if c (x^t) < c (x^b) then x^b = x^t
    update p^- and p^+ (It is done by using equation 1)
  until time limit
return x^b
```

x is the current solution, x^t is the temporary solution, x^b is the best solution.

Just introducing the probability update requires five parameters and the methods that are used for repairing and destroying has their own parameters. Therefore, tuning parameters becomes a disadvantage of ALNS with high number of parameters. Another disadvantage of the ALNS is that it gets easily complex as a result of many different neighborhood structure [6].

The Acceptance Methods: accept ()

The acceptance method determines whether the temporary solution is accepted as the current solution or not.

Greedy acceptance:

In the paper that introduced the LNS method [4], the greedy acceptance method is used. The temporary solution is accepted if the cost of the temporary solution is better than the current solution's cost.

Simulated annealing:

In the paper that introduced the ALNS method [5], the acceptance is adapted from the simulated annealing: x^t is accepted if its cost is better than the current one; x^t is accepted with the probability $\exp(-(c(x^t) - c(x))/T)$, where T is called temperature and decreases at every iteration, if the cost is not improved.

In the papers that introduced the LNS method [4], the ALNS method [5], there are some described destroy and repair methods which will be adapted to the set cover problem in the specialization description.

The Repair methods: r ()

Greedy insertion:

It inserts the element that has the minimum cost until obtaining a feasible solution.

Regret heuristics:

It is the extended version of the greedy insertion with a kind of look ahead information when deciding which to add. Namely, it considers not only the 1^{st} insertion with the minimum cost but also the following insertions with the minimum costs. Then the method inserts the element in the best possible way.

The Destroy Methods: d ()

Random removal:

It removes n elements from the solution, randomly. It looks like unsuitable since it could remove some fitting elements that can lead to the optimal solution. However, this method produces a diversification in the solution and that prevents from getting stuck in the local optima.

Worst removal:

The purpose of worst removal is to remove the part that contributes the worst to the solution. The contribution of the parts can be defined in many ways, such as their costs, their effects on the solution.

Related removal:

It removes the part that are similar to each other. It tries to improve the relations between the parts of the solution. Since the related parts can be easily exchanged, the repair methods can choose to add these parts in a more efficient way.

Specialization Description of Adaptive Large Neighborhood Search

As an overall algorithm, the pseudo code mentioned in the previous section is used. The feasible initial solution is constructed randomly. And the accept, repair, and destroy methods are described by the following:

Acceptance Method:

It allows the current solution to replace with the temporary solution if the cost of the temporary solution is lower than the cost of current solution multiplied by (1 + allowance). allowance is a fixed parameter that is determined to be 0.1.

Repair Methods:

Repair methods take the destroyed solution (x^t) and inserts the selected subsets to get a feasible solution.

GreedyRepair ()

It inserts the subset that has the lowest score. Score of a subset is defined as following:

$score = \frac{weight}{n}, equation\ 1$, where n is the number of missing elements that the subset can cover.

If the subset cannot cover a missing element in the solution, its score becomes a big value.

OtherRepair (k)

It is an extension of the greedy repair. It inserts the subset with the highest score. Score of a subset is described by the following:

$score = \frac{x_0 + x_1/k}{weight}, equation\ 4$, where x_0 is the binary variable that indicates whether the subset includes a missing element; x_1 is the binary variable that indicates whether the subset includes an element that is not repeated in the solution, and k is the parameter that defines the priority of the missing element in the solution in comparison to the non-repeated element in the solution.

The pseudo codes for the repair methods are given in the following:

```
procedure GreedyRepair ()
  repeat
    Calculate each subsets' score using eqn. 3
    Select the subset that has the lowest score
    Insert the selected subset
  until a feasible solution is obtained
```

```
procedure OtherRepair (k)
  repeat
    Calculate each subsets' score using eqn. 4
    Select the subset that has the highest score
    Insert the selected subset
  until a feasible solution is obtained
```

Destroy Methods:

The destroy methods take a feasible solution and removes n selected subsets from the solution.

Freq (n)

It removes n subsets from the solution based on their scores. Score of a subset is determined by the total number of frequencies of the elements in the subset (def1). And the frequencies correspond how many times an element is included in the solution.

Weight (n)

It removes n subsets from the solution based on their scores. Score of a subset is determined by weight/(reg+1), reg is the number of unique elements (def2). This reg is penalized the scores; when a subset that includes a unique element in the solution is removed, the solution will have a missing value to cover up.

Mixed (n)

It removes n subsets from the solution again based on their scores. Score of a subset is calculated as the summation of the regularized scores from Freq () and Weight () (def3).

Random (n)

It removes n subsets from the solution randomly.

The pseudo codes for the destroy methods are given in the following:

```
procedure Freq (n)
  Calculate scores of subsets in x^t using def1
  Select the first n subsets that have the highest score
  Remove the selected subsets
```

```
procedure Weight (n)
  Calculate scores of subsets in x^t using def2
  Select the first n subsets that have the highest score
  Remove the selected subsets
```

```
Procedure Mixed (n)
  Calculate scores of subsets in x^t using def3
  Select the first n subsets that have the highest score
  Remove the selected subsets
```

```
procedure Random (n)
  Calculate scores of subsets in x^t randomly
  Select the first n subsets that have the highest score
  Remove the selected subsets
```

Local Search

Description

Searching all the neighborhood in the set cover problem is not easy since the number of sub sets are in a feasible solution is not constant and all the combinations should be searched for every missing element for each neighborhood structure. Therefore, it is decided to not to implement TABU search for this problem. However, a basic neighborhood that is constructed by removing a sub set that does not make the solution infeasible. The pseudo code of the local search with the basic neighborhood is given below.

```
procedure LocalSearch (solution)
  repeat
    calculate the saving values of each subset in solution
    remove the element that does not affect the feasibility and has the highest saving
  until time or convergence
return solution
```

The local search only gets the initial solution and remove the elements in a greedy way. Due to the limited neighborhood structure it cannot improve the solution much and in general it does not perform well. However, it can be used to stabilize the initial solution.

Parameter Tuning

Determining Running Time

The running time is not a parameter but in order to tune the parameters an appropriate running time should be determined. For the determination, the evolution of the best value is investigated. The algorithm that proposed is converging to a value after some amount of time. Therefore, it is decided to select a time that it starts to converge. To detect the conversion of best value the following plots are obtained for SCPNGR2 and SCP410 from two minutes run.

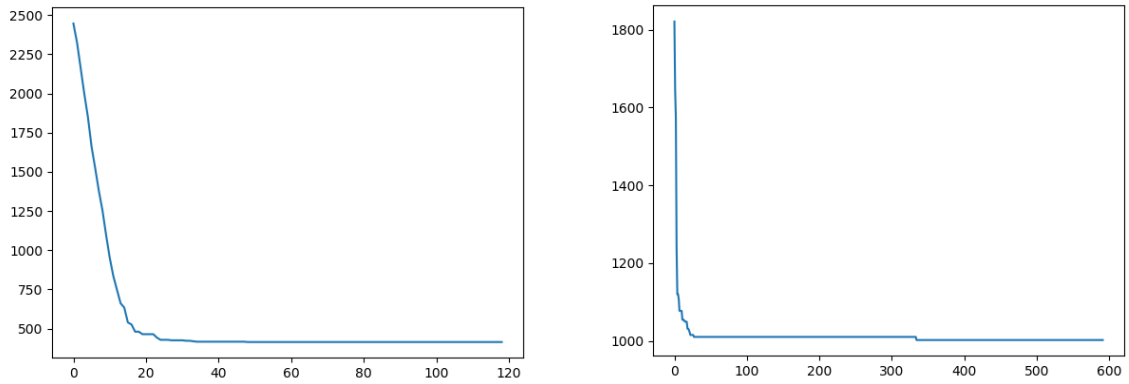

Figure 1:Evulution of best set cost values for large and small train dataset respectively.

According to the plots, it is decided to use 30 seconds for small data set and 60 seconds for the big data set.

Description and Tuning

The parameters that are considered through the tuning process are

- n: The number of subsets which will be removed by the destroy methods.
- lambda: The priority measurement of the previous probabilities of the destroy and repair methods. It is used in the equation 1.
- weights (w1, w2, w3, w4): They are defined in the equation 2 and used in the equation 1.
- k: The priority of the missing element in comparison to the non-repeated element in a solution.

Parameters	Values
n	5 10 20
lambda	0.1 0.5 0.9
w1	0 (fixed)
w2	0.2 (fixed)
w3	0.4 0.6
w4	1 1.5
k	3 10
allowance	0.1 (fixed)

Table 2: Ranges for the parameters

Since there eight parameters to tune and that will need so much time to process, the parameters w1, w2, and allowance are fixed. Reasonable ranges for the parameters are given in table 2.

In the tuning, the datasets of *scp410* and *scpnrg2* are used. The running time for the datasets are 30, 60 seconds, respectively. Each dataset is run 3 times with the same set of parameters.

In order to fill the tables of average percentage gaps and average spreading, the relative percentage values of the results using each combination of the parameters are calculated with the following formula:

$$val_{rel} = \frac{abs(result - bestknown)}{bestknown} * 100$$

, where bestknown is the best value that is obtained through the tuning process. Since in the selection of the parameters, the relative standings are considered, as a bestknown doesn't have to be the optimum value

The relative values are calculated for the 3 run of the same parameters. The mean of the relative values from the 1st tuning dataset and the mean of the relative values from the 2nd tuning dataset are averaged. These averages are named as average percentage gaps and are given in the following tables; the mean of the relative values from the 1st tuning dataset and the mean of the relative values from the 2nd tuning dataset are used to calculate the standard deviation. The resulting values are called average spreading and shown in the following table 3 and table 4.

	Average Percentage Gaps							
	lambda=0.1							
	w3=0.4				w3=0.6			
	w4=1		w4=1.5		w4=1		w4=1.5	
	k=3	k=10	k=3	k=10	k=3	k=10	k=3	k=10
n=5	7.157258	6.688325	6.175144	5.813006	7.06427	6.152188	6.970949	6.271292
n=10	5.54868	5.07825	4.50851	5.657305	5.200515	4.607653	5.452365	5.139466
n=20	3.428417	3.498616	3.283196	3.599256	3.495455	3.131987	3.660472	3.795047
	lambda=0.5							
	w3=0.4				w3=0.6			
	w4=1		w4=1.5		w4=1		w4=1.5	
	k=3	k=10	k=3	k=10	k=3	k=10	k=3	k=10
n=5	6.167325	6.508005	6.393391	5.423254	6.989247	6.042066	6.715772	5.948911
n=10	5.301155	4.192118	5.46168	3.969212	4.656559	5.04315	6.239187	5.310471
n=20	3.854433	2.887456	2.583207	2.092981	3.648329	3.851439	3.313638	3.067777
	lambda=0.9							
	w3=0.4				w3=0.6			
	w4=1		w4=1.5		w4=1		w4=1.5	
	k=3	k=10	k=3	k=10	k=3	k=10	k=3	k=10
n=5	6.054376	6.549259	6.789132	5.32095	5.974862	5.615718	6.728082	5.212492
n=10	5.452531	5.072095	5.389818	4.733079	5.944586	4.786477	4.888947	4.823073
n=20	3.593268	2.850693	3.770427	2.644423	3.752129	2.493379	3.454201	3.238782

Table 3: The average percentage gaps from parameter tuning

	Average Spreading							
	lambda=0.1							
	w3=0.4				w3=0.6			
	w4=1		w4=1.5		w4=1		w4=1.5	
	k=3	k=10	k=3	k=10	k=3	k=10	k=3	k=10
n=5	0.712355	0.6483	0.802342	1.020665	0.711746	1.190213	1.014821	1.139726
n=10	0.817681	0.455949	1.051588	0.987491	1.218468	0.897138	0.900008	1.551664
n=20	0.810453	0.463243	1.341786	0.855813	0.729684	1.322447	0.812207	0.920734
	lambda=0.5							
	w3=0.4				w3=0.6			
	w4=1		w4=1.5		w4=1		w4=1.5	
	k=3	k=10	k=3	k=10	k=3	k=10	k=3	k=10
n=5	0.552866	0.350486	1.236617	1.177091	0.471097	0.438695	0.413289	0.963125
n=10	0.536498	0.224664	0.74529	0.798054	0.578064	0.737061	0.983835	0.484824
n=20	0.626162	1.046711	0.9169	1.037556	0.995525	0.943579	0.552589	0.792692
	lambda=0.9							
	w3=0.4				w3=0.6			
	w4=1		w4=1.5		w4=1		w4=1.5	
	k=3	k=10	k=3	k=10	k=3	k=10	k=3	k=10
n=5	1.52453	1.038206	1.048468	0.738379	0.368425	0.775432	0.38913	0.747465
n=10	0.603324	0.612039	0.646892	0.543364	0.741969	0.409936	0.649786	0.925408
n=20	1.518418	0.836227	0.985107	1.197571	1.398915	0.655319	0.982949	0.398836

Table 4: The average spreading from parameter tuning

It can be seen from the average percentage gaps tables that n=20 gives much better results than n=5 and 10; k=10 gives better results than k=0.3. lambda=0.1 gives the worse results in comparison to the other values; and lambda=0.5 and lambda=0.9 give close results to each other. But lambda=0.5 is chosen since it has a lower spreading than lambda=0.9. w3=0.4 gives better results than w3=0.6. Finally, w4=1.5 gives better solutions than w=1. In fact, by considering the correlation of the parameters with each other, the set of the parameters which are chosen gives the best results. It is shown in the following table:

Parameters	Values
n	20
lambda	0.9
w1	0 (fixed)
w2	0.2 (fixed)
w3	0.4
w4	1.5
k	10
allowance	0.1 (fixed)

Table 5: Candidate parameter values

Test Results

The test of the algorithm is made by using two test datasets SCP49, SCPNRG1. In order to deal with the stochastic structure of the algorithm, for each dataset the algorithm run ten times with the running time of 30 for dataset SCP49 and 60 for SCPNRG1. Then, average percentage gap and spreading of the results for each dataset is calculated for the evaluation of the algorithm for each dataset. The results and the iteration numbers are presented in the tables below.

	ITERATIONS										
	1	2	3	4	5	6	7	8	9	10	Mean
SCP49	219	209	207	200	183	194	193	174	200	196	197,5
SCPRNG1	75	58	57	56	54	57	62	58	62	61	60

Table 6: Iteration Numbers of Test

	RESULTS												
	1	2	3	4	5	6	7	8	9	10	Error	Std.	Best Known
SCP49	1165	1214	1201	1211	1217	1194	1206	1199	1214	1207	87,6443	2,366866	641
SCPRNG1	457	454	457	463	456	459	468	463	467	463	161,761	2,718974	176

Table 7: Results of Test

The optimum solution for SCP49 data is taken from Beastley [7] and the best-known solution for SCPRNG1 is taken from Sebastian et al [8]. The results of the algorithm do not seem very promising. The average percent gap in both datasets is very high and for the large dataset its even higher than 100. The performance on the small data set seems better than the large one but the high value of relative gap could be deceiving while comparing two datasets with different objectives.

The difference in the iteration numbers are dependent on the dataset size. When the convergence issue mentioned in determining time part is considered, this number of iterations for each data is enough for the both datasets.

Discussion and Conclusion

In this paper, first a simple local search algorithm is tried. However, due to the limited capabilities of the simple local search an ALNS algorithm is constructed with four different destroy and two repair methods. The repair and destroy methods are mainly based on greedy approach except the randomized removal. The algorithm parameters are tuned in using two datasets with different sizes and tested in another two datasets.

The results of the test for the algorithm is not very promising for both datasets. This is due to getting stuck at a local optimum around some certain values of objective. It is expected to avoid from the local optima since the ALNS algorithm uses more than one destroy and repair methods; however,

the algorithm is not able to escape from a local optimum. This problem can be easily seen from the best value evolution graph. This means that the algorithm even with a random removal cannot diversify enough to get better solutions. In order to diversify the solution enough to get rid of the local optimum, different approaches such as regret heuristic rather than greedy based approaches can be tried to repair the solution. Also, related removal could be considered to have a destroy method which takes the relatedness of the different parts in the solution into account.

References

[1]Richard M. Karp (1972). "Reducibility Among Combinatorial Problems" (PDF). In R. E. Miller; J. W. Thatcher; J.D. Bohlinger (eds.). Complexity of Computer Computations. New York: Plenum. pp. 85–103. doi:10.1007/978-1-4684-2001-2_9

[2] People.brunel.ac.uk. (2019). Set covering. [online] Available at: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html [Accessed 28 Apr. 2019].

[3] Panneerselvam, Ramasamy. (2010). Literature review of covering problem in operations management. International Journal of Services, Economics and Management. 2. 267-285. 10.1504/IJSEM.2010.033367.

[4] D. Pisinger and S. Røpke. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[5] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming), *volume 1520 of Lecture Notes in Computer Science*, pages 417–431, 1998.

[6] Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers and Operations Research*, Volume 34, Issue 8, pp. 2403-2435.

[7] J.E. Beasley, An algorithm for set covering problem, European Journal of Operational Research, Volume 31, Issue 1, 1987, Pages 85-93, ISSN 0377-2217.

[8] Ceria, Sebastián & Nobili, Paolo & Sassano, Antonio. (1998). A Lagrangian-based heuristic for large-scale set covering problems. Math. Program.. 81. 215-228. 10.1007/BF01581106.

APPENDIX

The python codes that are used in this project is in Codes.zip file. The content of file is following py files and the datasets.

- Main.py
- DataObject.py
- ALNS.py
- ParamTuning.py
- Test.py