A CLONALG-based Approach for the Set Covering Problem

Masruba Tasnim [a,b], Shahriar Rouf [a,c], M. Sohel Rahman [a,d]
[a]AℓEDA Group
Department of CSE, BUET,
Dhaka - 1000, Bangladesh.
[b]Email: masruba@gmail.com [c]Email: nafi3000@gmail.com [d]Email: sohel.kcl@gmail.com

***Abstract*— In this paper, we propose a CLONALG-based simple heuristic, which is one of the most popular artificial immune system (AIS) models, for the non-unicost set covering problem (SCP). In addition, we have modified our heuristic to solve the unicost SCP. It is well known that SCP is an NP-hard problem that can model several real world situations such as crew scheduling in airlines, facility location problem, production planning in industry etc. In real cases, the problem instances can reach huge sizes, making the use of exact algorithms impractical. So, for finding practically efficient approaches for solving SCP, different kind of heuristic approaches have been applied in the literature. To the best of our knowledge, our work here is the first attempt to solve SCP using Artificial Immune System. We have evaluated the performance of our algorithm on a number of benchmark non-unicost instances. Computational results have shown that it is capable of producing high-quality solutions for non-unicost SCP. We have also performed some experiments on unicost instances that suggest that our heuristic also performs well on unicost SCP.**

I. Introduction

The Set Covering problem (SCP) is a well known combinatorial optimization problem. Given a set of elements (called the universe) and sets whose union comprises the universe, the set cover problem is to identify the smallest number of sets whose union still contains all elements in the universe. For example, assume we are given the following elements $U = \{u_1, u_2, \ldots, u_m\}$, subsets $S_1, S_2, \ldots, S_n \subseteq U$ and costs $c_1, c_2, \ldots, c_n$. The goal is to find a set $I \subseteq \{1, 2, \ldots, n\}$ that minimizes $\sum_{i\in I} c_i$, such that $\bigcup_{i\in I} S_i = U$.

SCP is the problem of covering the rows of an m-row, n-column, zero-one matrix (a_{ij}) by a subset of columns of minimum global cost. So, we can formulate the problem through a binary linear programming model.

Defining x_j = 1 if column j (with cost $c_j \geq 0$) belongs to the solution and x_j = 0 otherwise, the SCP is to

Minimize

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

Subject to

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

$$\forall j \in \{1, 2, \ldots n\}, x_j \in \{0, 1\} \tag{3}$$

Equation 2 ensures that each row is covered by at least one column and Equation 3 is the *integrality constraint*. If the costs c_j are equal, the problem is referred to as the unicost SCP, otherwise, the problem is called the weighted or non-unicost SCP.

The problem of set covering has been considered in the literature as a basic formulation for many real-world optimization problems. Therefore, it is well-known for its numerous applications such as scheduling, manufacturing, service planning, information retrieval, etc. One important application of the problem is the delivery and routing problem. Another famous problem is the airline crew scheduling problem [7]. Balas [2] provided a survey on the applications of SCP in location, distribution and scheduling.

SCP is known to be NP-hard [22] and many algorithms have been developed for solving the problem. Exact algorithms are mostly based on branch-and-bound and branch-and-cut techniques [3], [21]. However, exact algorithms are feasible for instances of very limited size. For this reason, many research efforts have been focused on the development of heuristics to find good or near-optimal solutions within a reasonable period of time. Greedy algorithms may be the most natural heuristic approach for quickly solving large combinatorial problems. As for the SCP, the simplest such approach is the greedy algorithm of Chvatal [11]. Although simple, fast and easy to code, greedy algorithms could rarely generate solutions of good quality as a result of their myopic and deterministic nature.

Researchers have tried to improve greedy algorithms by introducing some randomness and memory into it and obtained promising results [24], [29]. To improve the solution quality, modern heuristics, such as simulated annealing (SA), genetic algorithms (GA), and neural networks (NN), introduce randomness in a systematic manner. These heuristics are often classified as meta-heuristics, since they are top-level general strategies that guide other heuristics to search for feasible solutions. There exist a number of heuristic based solutions for SCP, such as genetic algorithms [1], [6], simulated annealing algorithms [27], tabu search methods [9], neural network algorithms [33], ant colony optimization (ACO) based [13], [23], [32], [34] approaches etc. Rather than applying a general meta-heuristic, some other heuristics, such as those based on Lagrangian relaxation are developed based on the problem-specific information of the SCP [4], [7], [10]. For a deeper comprehension of most of the effective algorithms for the SCP in the literature, the interested readers are referred to the survey by Caprara et. el [8].

In this paper, we would like to enrich the repertoire of heuristic solutions for SCP by presenting a new meta-heuristic based algorithm. In particular, we present an algorithm based on Artificial Immune System (AIS) [15] [26].

Artificial immune systems (AIS) are a special class of biologically inspired algorithms, which are based on the immune system of vertebrates and derived from various immunological theories, namely, the clonal selection principle, negative selection, immune networks etc [14]. These algorithms typically exploit the immune system's characteristics of learning and memory to solve a problem. AIS has been successfully applied to solve several NP-hard combinatorial optimization problems such as job shop scheduling [12], weapon-target assignment problem [31], flow shop scheduling [20] etc. It has also been applied to domains like data analysis [18], soft computing [16], network security [25], [28] etc. Wang et. al [35] provided a survey of AIS based optimization methods and applications.

In this paper, to solve SCP, we have adopted a variation of the Clonal Selection Algorithm, originally called CSA in [17], and later renamed to CLONALG in [19]. CLONALG, inspired by the clonal selection theory of acquired immunity, has shown success on broad range of engineering problem domains. The clonal selection principle describes the basic features of an immune response to an antigenic stimulus. To the best of our knowledge, this work is the first attempt to solve SCP using AIS. The goal of this work is to design a simple heuristic that generates good results for non-unicost set covering problems. Furthermore, we have modified our heuristic to solve the unicost version of the SCP. To verify the efficiency of the proposed algorithms, extensive computational experiments are conducted on benchmark instances from Beasley's OR Library [5].

The remainder of this paper is organized as follows. In Section II, we present a brief description of CLONALG. In Section III, we present our main contribution, where we describe the proposed algorithms in detail. The computational results, along with an insightful discussion, are presented in Section IV. Finally, conclusions are drawn with some future research directions in Section V.

II. CLONALG

CLONALG is a well known model based on the clonal selection and affinity maturation principle which is similar to mutation-based evolutionary algorithms. Survival of the fittest concept of evolutionary algorithm also applies here. It is inspired by the following elements of the clonal selection theory:

- Maintenance of a specific memory set
- Selection and cloning of most stimulated antibodies
- Death of non-stimulated antibodies
- Affinity maturation (mutation)
- Re-selection of clones proportional to affinity with antigen
- Generation and maintenance of diversity

The goal of the algorithm is to develop a memory pool of antibodies that represents a solution to an engineering problem. In this case, an *antibody* represents an element of a solution or a single solution to the problem and an *antigen* represents an element or evaluation of the problem space.

The algorithm provides two mechanisms for searching for the desired final pool of memory antibodies. The first is a local search provided via affinity maturation of cloned antibodies. More clones are produced for better matched (selected) antibodies. The second search mechanism provides a global scope and involves the insertion of randomly generated antibodies to be inserted into the population to further increase the diversity and provides a means for potentially escaping local optima. Fig. 1 provides a complementing diagrammatic representation of the tasks and workflow of the CLONALG technique.

III. Proposed Algorithm

This section presents the details of the strategies used in our proposed algorithm. It can be broken down into three parts : Initial population generation, cloning principle and mutation strategy. As a basis, we first introduce the representation scheme and the affinity function.

A. Representation and Affinity Function

The first step in designing a heuristic algorithm for a particular problem is to devise a suitable representation scheme. The usual 0-1 binary representation is an obvious choice for the SCP since it represents the underlying 0-1 integer variables. We use an n-bit binary string as the antibody structure where n is the number of columns in the SCP. A value of 1 for the i-th bit implies that column i is in the solution. Similarly, we use an m-bit binary string as the antigen structure where m is the number of elements to be covered. A value of 1 for the i-th bit implies that object i must be covered.

The bit string lengths of the antibody and the antigen are equal to the number of subsets (n) and the number of elements to be covered (m) respectively. In our proposed algorithm, each antibody in the antibody pool is a feasible solution to the problem. The feasible solution construction method will be described shortly.

We define the affinity of an antibody as follows. It is measured by the sum of the cost of the subsets included in the antibody bit string. Suppose, the cost associated with two antibodies A_1 and A_2 are a_1 and a_2 respectively. Then, $Affinity(A_1) = a_1$ and $Affinity(A_2) = a_2$.

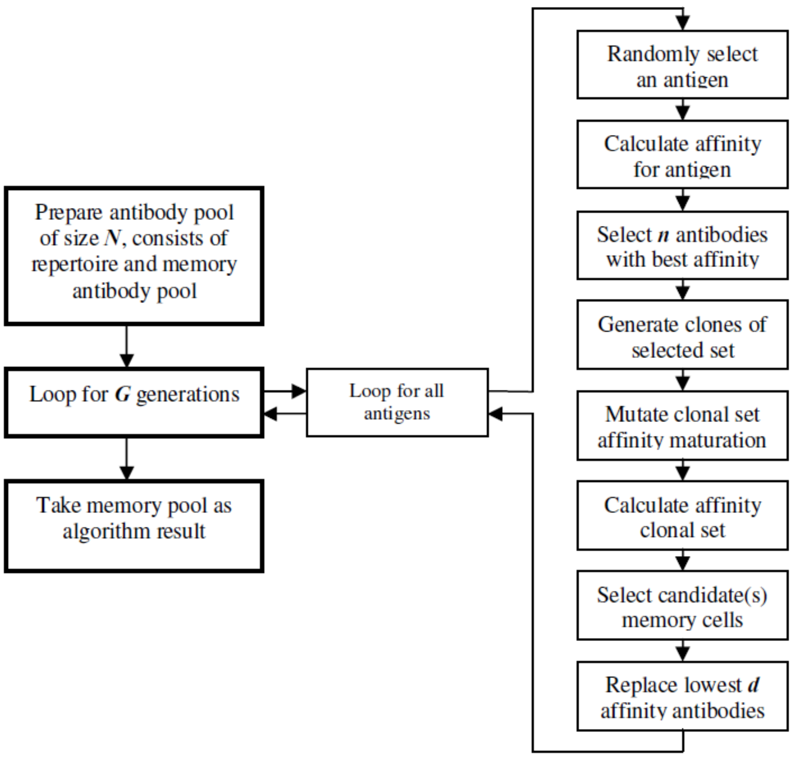
Prepare antibody pool of size N, consists of repertoire and memory antibody pool
Loop for G generations
Take memory pool as algorithm result
Loop for all antigens
Randomly select an antigen
Calculate affinity for antigen
Select n antibodies with best affinity
Generate clones of selected set
Mutate clonal set affinity maturation
Calculate affinity clonal set
Select candidate(s) memory cells
Replace lowest d affinity antibodies

Figure 1: Overview of the CLONALG algorithm

For the unicost SCP, the affinity is measured by the number of subsets included in the antibody bit string. Suppose, Number of elements in two antibodies A_1 and A_2 are n_1 and n_2 respectively. Then, $Affinity(A_1) = n_1$ and $Affinity(A_2) = n_2$.

B. Overview of the Proposed Approach

In our algorithm AIS-SCP, we have proposed two kinds of heuristics for prioritizing both the elements inside a subset and the subsets themselves. After processing the given SCP instance, elements in every subset are sorted according to their *usefulness*. We determine the usefulness of an element by tracking how many subsets covered the specific element. According to our heuristic, an element which is covered by lesser number of subsets is more useful than an element covered by more subsets. Element usefulness helps to determine the global usefulness of a subset.

On the other hand, we have sorted the subsets according to the global usefulness of the subsets. We have determined the relative global usefulness of two subsets S_1 and S_2 according to the following criteria:

1) Let, cost associated with S_1 and S_2 are c_1 and c_2 respectively.
    a) If $(c_1 < c_2)$, then subset S_1 has better global usefulness.
    b) Else If $(c_1 > c_2)$, then subset S_2 has better global usefulness.
    c) Else Check Criterion (2).
2) If the total number of elements covered by the subset S_1 and S_2 are e_1 and e_2 respectively, then
    a) If $(e_1 > e_2)$, then subset S_1 has better global usefulness.
    b) Else If $(e_1 < e_2)$, then subset S_2 has better global usefulness.
    c) Else Check Criterion (3).
3) The subset, which covers more useful elements, has better global usefulness.

For unicost SCP, criterion (1) is not meaningful and hence is ignored.

We have created a single antigen for the SCP problem. The single antigen is composed of “11 . . . 1”, i.e. m consecutive 1’s. Recall that, m is the length of the string is the number of objects to be covered. We have generated the initial antibody pool of a fixed size. Then, our algorithm proceeds by executing a number of iterations of exposing the system to all known antigens, in our case only one antigen.

Preprocessing is a popular method to speed up the algorithm. At the preprocessing step, we have applied column domination and column inclusion methods to reduce the run time of AIS-SCP.

- *Column domination:* Any column j whose rows can be covered by other columns for a cost less than c_j can be deleted from the problem. As expected, this reduction is ineffective for unicost problems.
- *Column inclusion:* If a row is covered by only one column after the above domination, this column must

be included in the optimal solution.

We formally describe our algorithm in Algorithm 1. The first step of the CLONALG technique is initialization, which involves preparing an antibody pool of a fixed size. This pool is then partitioned into two components, a memory antibody section that eventually becomes representative of the algorithms solution and the remaining antibody pool used for introducing additional diversity into the system.

Algorithm 1 AIS-based algorithm for solving SCP (AIS-SCP)

```
1: Preprocess the given SCP instance
2: Initialize parameters and generate Initial Population
3: for $iterationCounter = 1$ to $nIteration$ do
4:    Select the Antigen
5:    for $i = 0$ to $mem - 1$ do
6:       Calculate $Affinity(M[i])$
7:    end for
8:    Sort Memory Pool
9:    for $i = 0$ to $rem - 1$ do
10:      Calculate $Affinity(R[i])$
11:   end for
12:   Sort Remaining Pool
13:   for $i = 0$ to $N_s - 1$ do
14:      Select the $i$th best antibody
15:      Clone the selected antibody proportion to its affinity
16:   end for
17:   for $i = 0$ to $N_c - 1$ do
18:      Mutate each clone
19:   end for
20:   Sort Mutated clones
21:   for $i = 0$ to $memRep - 1$ do
22:      Replace $i$th worst antibody in the $M[\ ]$ by the mutated antibody
23:   end for
24:   $i = 0,\ j = rem - 1$
25:   while $i < remRep$ do
26:      for $k = 0$ to $RETRIAL\_COUNT$ do
27:         Create Random Antibody $A$
28:         if $Affinity(A) < Affinity(R[j])$ then
29:            Replace $R[j]$ by $A$
30:            break
31:         end if
32:      end for
33:      $i = i + 1$
34:      $j = j - 1$
35:   end while
36: end for
37: return the best solution found
```

In Algorithm 1, mem and rem denote the total number of antibodies in the memory pool and the remaining pool respectively. The variables $M[\ ]$ and $R[\ ]$ in Algorithm 1 denote the arrays which contain the antibodies of the memory pool and the remaining pool respectively. The function $Affinity(A)$ returns the affinity of the antibody against the antigen. The variable N_s is the size of the antibody pool from which clones are created for all antibodies. The number of clones created for each antibody is calculated according to our cloning principle which will be described shortly. The variable N_c denotes the total number of clones created for all antibodies per antigen exposure. The variable $memRep$ denotes the total number of replacement of worst antibodies in the memory pool by the mutated antibodies based on affinity. Similarly, $remRep$ is the total number of replacement of worst antibodies in the remaining pool by the randomly created antibodies. This step is done in order to introduce some randomness at each iteration.

In our algorithm, we have taken an approach to replace worst antibodies in the random pool by generating new random antibodies. For this purpose, we have used a constant $RETRIAL\_COUNT$. It is a predetermined value that denotes the maximum number of times random antibodies are created and compared with the existing antibody in the remaining pool. If the newly generated random antibody A_1's affinity value is less than an existing antibody A in the remaining pool, A gets replaced. Having lower affinity (and thus lower cost), A_1 serves as a better option than A. The variable, $nIteration$ determines the total number of iterations the our AIS will run. We report the iteration count or generation number needed to first reach the best solution value or the best solution found within $nIteration$.

C. Generation of Initial Population

We have generated the initial population in a mixture of probabilistic randomization and greedy addition and removal of subsets from the antibody bit strings. As mentioned earlier, each antibody in the antibody pool is a feasible solution to the problem. To construct a feasible solution, we use greedy addition and removal of subsets from the initial randomized antibody bit strings. For randomization purposes we have used an utility function probability(p, q) which returns true with a probability of $\frac{p}{q}$. We have initialized the antibody bit string by using two randomization function described below.

1) Randomization Function I: In the first randomization function, p_1 is computed by multiplying p with a probability multiplier factor using the following function:

$$p_1 = \max\left(1, \left\lceil p \times \exp\left(-\ln 2 \times \frac{i}{n}\right)\right\rceil\right) \quad (4)$$

where,

i	=	ith bit of the antibody string
n	=	Total length of the antibody string

In Equation 4, $\exp\left(-\ln 2 \times \frac{i}{n}\right)$ is the probability multiplier. If the computed value of p_1 becomes less that 1, then p_1 is set to 1. Finally, ith bit of the antibody is set with probability $\frac{p_1}{q}$.

2) Randomization Function II: In the second randomization function, ith bit is set to 1 based on the probability

$\frac{p_1}{q}$, where

$$p_1 = \max\left(1, \left\lceil p \times \exp\left(-\ln 2 \times \frac{n-i}{n}\right)\right\rceil\right) \quad (5)$$

In Equation 5, $\exp\left(-\ln 2 \times \frac{n-i}{n}\right)$ is the probability multiplier. So, ith bit is set to 1 based on the probability of $\frac{p_1}{q}$.

3) Usage of the Randomization Functions: We have passed different values of p, q to the two mentioned randomization functions in our algorithm to generate the initial population. Fig. 2 represents the relationship between the probability multiplier and the bit index (i) of the antibody string, for the two randomization functions. In the graph of Fig. 2, the total number of bits n in the antibody is set to 100. It is clear from the graph that for the case of randomization function I, it can be said that first bit gets set with $\frac{p}{q}$ probability while, it goes down exponentially to $\frac{p}{2\times q}$. On the other hand, randomization function II is the opposite of randomized function I. Here, last bit gets set with probability $\frac{p}{q}$ while, it goes down exponentially to $\frac{p}{2\times q}$ at the first bit of the antibody bit string.

As mentioned earlier, initially we sort the subsets according to global usefulness. So, the first randomization function increases the probability of selecting lower cost subsets in the initial population. On the other hand, the second randomization function increases the probability of selecting higher cost subsets in the initial population. As a result, our initial population become diverse.

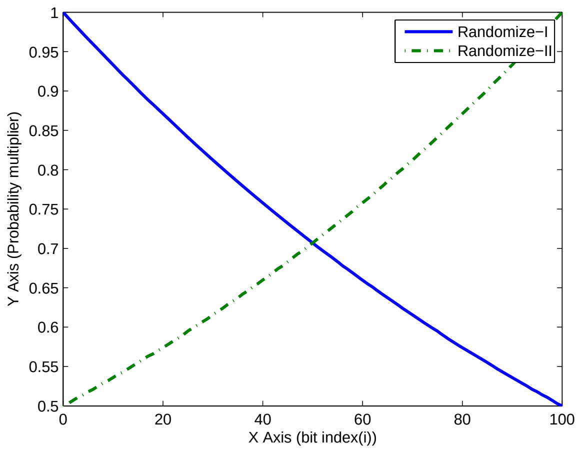

Figure 2: Plot of Probability Multiplier vs. bit index of the antibody bit string

4) Greedy Modifications: After initializing the whole antibody bit string, we apply greedy addition and remove operations to make the antibody bit string a feasible solution. The antibody must cover all the elements in the antigen to be a feasible solution. Hence, a function is called repeatedly to include a new subset greedily, until the antibody bit string becomes a feasible solution. In particular, a subset S constitutes a greedy choice if the inclusion of S would add the maximum number of new elements in the antibody bit string. If two subsets S_1 and S_2 add the same number of new elements, then the subset which has the lesser cost is selected as the desired subset to be added in the antibody bit string. The greedy remove operation tries to remove any redundant subset. If all the elements covered by a subset S, are also covered by any other subset or subsets in the antibody bit string, then the subset S can be removed from the antibody bit string.

D. Cloning Principle

The number of clones created from each of the N_s selected antibodies is proportional to their affinity using a rank based measure. This is achieved by first sorting the set of selected antibodies in increasing order of their respective affinities to the antigen. Then, clones are created according to the rank. So, the antibody with the lowest affinity is cloned more. The number of clones created from each antibody is calculated as follows:

$$numClones[i] = \left\lfloor \frac{\beta \times N_s}{i+1} + 0.5 \right\rfloor, \quad (6)$$

where, β is the clonal factor and N_s is the number of the selected antibodies, and i is the current rank of the antibody where $i \in [0, N_s - 1]$. The variable $numClones[i]$ stores the number of clones created for the i-th ranked antibody. The total number of clones prepared for each antigen exposure to the system is thus calculated as:

$$N_c = \sum_{i=0}^{N_s-1} numClones[i] \quad (7)$$

E. Mutation Strategy

Mutation is very common in AIS algorithms and in most of the cases it is done in totally random fashion. It generally works by inverting each bit in the solution with some small probability. Mutation is generally seen as a background operator which provides a small amount of random noise. But this kind of blind mutation is not suitable for SCP. Because there are constraints, it is very unlikely that random alteration of objects would produce valid solutions. So, we have adopted a special method of mutation.

We have applied mutation to each cloned antibody. In our algorithm, the number of bits changed in the cloned antibody is based on a combination of the affinity factor, A_F and the time factor T_F. Total number of bits mutated in the antibody bit string calculated as follows.

$$B = s \times A_F \times T_F, \quad (8)$$

where,

s	=	Number of subsets included in the antibody string

The affinity factor, A_F is calculated as follows.

$$A_F = \exp(-(AMF \times minCost \times (1 + aff - minCost)^{1/8}) \quad (9)$$

where, AMF is the Affinity Multiplication Factor, a constant value. We set AMF to 0.1. The variable aff denotes the affinity of the cloned antibody selected for mutation. The $minCost$ denotes the minimum cost of the best antibody solution found so far. For unicost SCP, $minCost$ is the number of subsets of the best antibody solution found so far.

Fig. 3 plots the Equation 9, assuming $minCost$ = 100. It is evident from the Fig. 3 that the greater the affinity (aff) is, the greater the value of affinity factor (A_F). Higher value of A_F increases the number of bits mutated according to Equation. 8. Note that, as the AIS progresses, the affinity factor will also becomes stable at some point. This is bound to happen because the change in $minCost$ gets lower as the AIS advances.

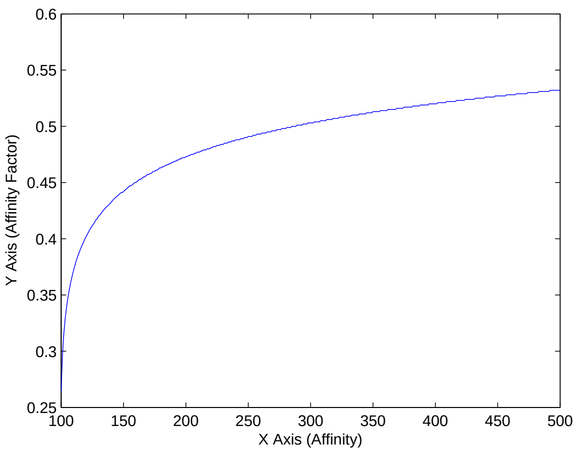

Figure 3: Affinity (aff) vs. Affinity Factor (A_F) curve

Time factor, T_F is calculated according to the following equation:

$$T_F = 1 - \exp(-5t), \tag{10}$$

where,

t	$=$	i/T
i	$=$	Iteration Number
T	$=$	Total No of Iterations

The relationship between the iteration number and time factor T_F is illustrated in Fig. 4, for the value of T = 50. At the initial phase of mutation, we have searched for a better quality solution near the current antibody's search space. If the min cost solution can be found by doing this, then we are done. After the initial phase of mutation, we have adopted an approach to increase versatility of our solution space by increasing the mutation rate exponentially.

In our algorithm, we have maintained a list. The list keeps the track of the subsets included in the antibody string. By randomly choosing a bit position in that list, the selected subset is removed from the cloned antibody bit string. Finally, to make an antibody bit string a feasible solution, we have applied the greedy addition and removal operations.

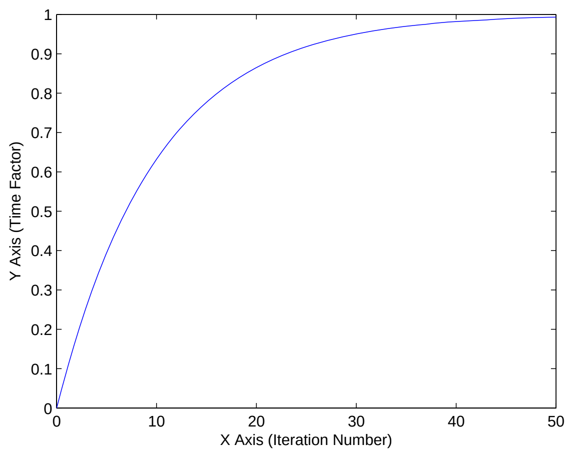

Figure 4: Iteration Number vs. Time Factor, T_F curve

IV. Experimental Results

A. *Experimental study on non-unicost SCPs*

The effectiveness of the heuristic presented in this paper is tested on 55 non-unicost SCP test instances from Beasley's OR-library [5]. Table I shows the details of these test problems, where density is the percentage of non-zero entries in the SCP matrix. Optimal solutions values for problem sets 4-6 and A-D are known. Problem sets E-F are large SCP's for which optimal solutions values are not known.

TABLE I.: Test Problem Details

Prob. Set	Rows (m)	Columns (n)	Density (%)	No. of Ins.
4	200	1000	2	10
5	200	2000	2	10
6	200	1000	5	5
A	300	3000	2	5
B	300	3000	5	5
C	400	4000	2	5
D	400	4000	5	5
NRE	500	5000	10	5
NRF	500	5000	20	5

We have carried out the experiments on a PC with *Intel Core i5 2.30 GHz processor* and *6 GB memory under Windows 7*. We have coded the algorithm presented in this paper in *C++*. Our algorithm AIS-SCP has solved each test instance 10 times. For solving each test instance by our algorithm, We have set a maximum number of 50 iterations per run. Other parameters of our proposed algorithm are set as follows:

Memory Antibody Pool Size	$=$	1000
Remaining Antibody Pool Size	$=$	1000
Number of Antibody Chosen	$=$	250
Clonal Factor	$=$	0.95
Memory Replacement Size	$=$	100
Remaining Replacement Size	$=$	100

Table II reports the detailed results for AIS-SCP and the greedy heuristics (G) presented in [11]. In Table II we give, for each problem,

- the optimal solution value (Opt.) for problem set 4-6 and A-D or best known solution value (BKS) for problem sets E-F;
- for greedy heuristics, the solution value and the time needed to reach the solution. The solution time is measured in CPU seconds;
- for AIS-SCP, the best solution value found among the 10 trials, the solution time and IC, the iteration count or generation number needed to first reach the best solution value. The solution time is measured in CPU seconds and is the time that the AIS-SCP takes to first reach the final best solution.

By examining Table II, we observe that, AIS-SCP outperforms greedy heuristics [11] for all of these 55 test instances. It is also noted that iteration count and solution time needed to reach the best solution are reasonably small for all the problems. From Table II, we can say that AIS-SCP found all the optimal or best-known solutions for the 55 non-unicost instances except for one problem set C.3. At the time of testing the effectiveness of our algorithm, we have given emphasis on the fact of finding better quality solution in relatively small number of iterations. As mentioned earlier, we set the value of maximum number of iterations in each run to 50. Under this restriction, our algorithm has produced encouraging and high quality solutions.

In order to bring out the efficiency of the proposed AIS-SCP algorithm the solutions of the same set of test instances have been compared with the greedy heuristics (G) as well as the following existing SCP heuristics

Be	:	Lagrangian heuristic by [4]
BeCh	:	Genetic algorithm by [6]
PROG	:	Probabilistic greedy search by heuristic [24]
Meta-RaPS	:	Effective and simple heuristic approach by [30]

Table III is the summarized results for the solution quality that reports the average GAP for all these different heuristics mentioned earlier, where GAP is the percentage of the deviation from the optima (Opt.) or best known solution (BKS), i.e. $Gap = ((solution - BKS)/BKS) \times 100$. It is clear from that Table III, the average gap for our algorithm AIS-SCP is promising under the restriction of finding better quality solution in relatively smaller number of iterations.

TABLE II.: Detailed results for non-unicost instances

Ins	Opt./	Greedy		AIS-SCP		
	BKS	Sol.	Time	Sol.	Time	IC
4.1	429	439	0	429	2.57	12
4.2	512	547	0	512	1.33	6
4.3	516	546	0	516	2.452	13
4.4	494	510	0	494	6.006	22
4.5	512	519	0	512	3.089	16
4.6	560	594	0	560	4.275	17
4.7	430	447	0	430	2.761	15
4.8	492	502	0	492	1.902	11
4.9	641	672	0	641	2.871	12
4.10	514	521	0	514	1.482	8
5.1	253	271	0	253	4.793	11
5.2	302	329	0	302	4.608	9
5.3	226	232	0	226	44.708	47
5.4	242	253	0	242	1.781	4
5.5	211	220	0	211	3.697	9
5.6	213	234	0	213	11.42	15
5.7	293	302	0	293	2.254	6
5.8	288	308	0	288	7.562	16
5.9	279	290	0	279	5.475	8
5.10	265	275	0	265	5.086	12
6.1	138	147	0	138	0.656	4
6.2	146	160	0	146	2.886	15
6.3	145	152	0	145	5.273	20
6.4	131	137	0	131	1.919	11
6.5	161	178	0	161	2.886	14
A.1	253	271	0	253	2.449	4
A.2	252	267	0	252	6.973	7
A.3	232	244	0	232	16.172	18
A.4	234	246	0	234	79.965	37
A.5	236	247	0	236	10.483	10
B.1	69	73	0	69	8.597	7
B.2	76	78	0	76	12.371	9
B.3	80	85	0	80	2.216	3
B.4	79	85	0	79	10.717	8
B.5	72	76	0	72	1.872	1
C.1	227	246	0	227	19.255	15
C.2	219	231	0	219	29.735	15
C.3	243	256	0	247	14.888	50
C.4	219	239	0	219	7.612	8
C.5	215	228	0	215	6.271	5
D.1	60	68	0.01	60	14.963	4
D.2	66	70	0	66	15.444	8
D.3	72	78	0.02	72	10.51	7
D.4	62	65	0	62	5.678	5
D.5	61	71	0.01	61	2.889	4
NRE.1	29	31	0.02	29	20.849	2
NRE.2	30	34	0	30	25.372	8
NRE.3	27	32	0	27	5.788	8
NRE.4	28	32	0.01	28	7.005	3
NRE.5	28	31	0	28	5.195	3
NRF.1	14	17	0.02	14	3.931	1
NRF.2	15	16	0.02	15	3.48	50
NRF.3	14	16	0.01	14	4.828	50
NRF.4	14	15	0.01	14	45.943	1
NRF.5	13	15	0.01	13	14.888	14

B. Experimental study on unicost SCPs

We also perform experiments on unicost SCP. The unicost test instances used are the set E and CLR from the OR-Library, as shown in Table IV. For the purpose of testing our algorithm, we have also used non-unicost test instances of set NRE, two test instances of set 4 (4.2, 4.6) and two test instances of set 5 (5.1, 5.9). The cost of subsets are ignored for non-unicost test instances of set NRE, 4.2, 4.6, 5.1 and 5.9.

The test results for mentioned SCP instances are reported in Table V. The table reports optimal solution

TABLE III.: Summarized results for the solution quality (average GAP) for non-unicost instances

Ins	PROG	Be	Greedy	BeCh	Meta RaPS	AIS-SCP
4	0.57	0.06	3.78	0.00	0.00	0.00
5	0.88	0.18	5.51	0.09	0.00	0.00
6	0.69	0.56	7.22	0.00	0.00	0.00
A	0.75	0.82	5.61	0.00	0.00	0.00
B	0.00	0.81	5.57	0.00	0.00	0.00
C	0.87	1.93	6.88	0.00	0.00	0.33
D	0.00	2.75	9.79	0.00	0.00	0.00
NRE	0.00	3.5	12.75	0.00	0.00	0.00
NRF	1.43	7.16	12.98	0.00	0.00	0.00

TABLE IV.: Unicost Instances

Set	No. of Ins.	Rows (m)	Columns (n)	Density (%)	Optimal Solution
E	5	50	500	20	Known
CLR.10	1	511	210	2	Unknown
CLR.11	1	1023	330	5	Unknown
CLR.12	1	2047	495	2	Unknown
CLR.13	1	4095	715	5	Unknown
NRE	5	500	5000	10	Known
4.2	200	1000	2	10	Known
4.6	200	1000	2	10	Known
5.1	200	2000	2	10	Known
5.9	200	2000	2	10	Known

value (Opt.) or best known solution value (BKS). The table also reports solution, solution time, iteration count or generation number needed to first reach the best solution value and the GAP for AIS-SCP. The solution time is measured in CPU seconds and GAP is the percentage of the deviation from the optima (Opt.) or best known solution (BKS) as mentioned earlier.

By examining Table V, we observe that, AIS-SCP is capable of producing good solutions for unicost instances. It is also noted that iteration count and average solution time needed to reach the best solution found are reasonably small for all the unicost problems.

TABLE V.: Test results for unicost instances

Instance	Opt./	AIS-SCP			
	BKS	Sol.	Time	IC	GAP
E.1	5	5	0	0	0.00
E.2	5	5	0	0	0.00
E.3	5	5	0	0	0.00
E.4	5	5	0	0	0.00
E.5	5	5	0	0	0.00
CLR.10	25	25	1.864	8	0.00
CLR.11	23	23	3.441	9	0.00
CLR.12	23	23	83.641	46	0.00
CLR.13	23	23	81.147	21	0.00
NRE.1	17	18	0	0	5.88
NRE.2	17	17	216.473	43	0.00
NRE.3	17	17	185.901	39	0.00
NRE.4	17	17	178.058	39	0.00
NRE.5	17	17	184.051	39	0.00
4.2	37	37	6.855	21	0.00
4.6	38	38	5.018	18	0.00
5.1	35	35	11.482	20	0.00
5.9	36	36	6.174	6	0.00

V. Conclusion and Future Work

In this paper, a simple CLONALG-based heuristic is proposed for the non-unicost set covering problem. Then, we have modified the heuristic to solve the unicost SCP problems. To the best of our knowledge, this paper is a first attempt to solve SCP using artificial immune system. Our proposed algorithm systematically introduces randomness into a simple greedy heuristic to construct a feasible solution. Computational results show that the proposed algorithm is efficient in generating encouraging and high quality solutions for solving non-unicost problems in terms of the solution quality. Comparison with some other SCP heuristics also shows that it is competitive in solving the non-unicost SCP. Furthermore, another important attractiveness of this heuristic is its good performance for solving unicost problems.

Future works consist in using a wider parameter calibration in order to better explore the search space, as it is trying to determine a relation between the parameters and features of the instances. This aims to improve the quality of the solutions for some problem instances and solve the problem for more larger instances. It is also possible to improve the performance of the algorithm by making an adaptive adjustment of the parameters while the algorithm is running.

References

[1] Uwe Aickelin. An indirect genetic algorithm for set covering problems. *CoRR*, abs/0803.2965, 2008.

[2] Egon Balas. A class of location, distribution and scheduling problems: Modeling and solution methods. In *Proceedings of the ChineseUS Symposium on System Analysis*, 1983.

[3] Egon Balas and Maria C. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44:875890, 1996.

[4] John E. Beasley. A lagrangian heuristic for set covering problems. *Naval Research Logistics*, 37:151164, 1990.

[5] John E. Beasley. Obtaining test problems via internet. *Journal of Global Optimization*, 8:429433, 1996.

[6] John E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392404, 1996.

[7] Alberto Caprara, Matteo Fischetti, and Paolo Toth. A heuristic method for the set covering problem. *Operations Research*, 47:730743, 1999.

[8] Alberto Caprara, Matteo Fischetti, and Paolo Toth. Algorithms for the set covering problem. *Annals of Operations Research*, 98:353371, 2000.

[9] Marco Caserta. Tabu search-based metaheuristic algorithm for large-scale set covering problems. *Metaheuristics: Progress in complex systems optimization*, page 4363, 2007.

[10] Sebastián Ceria, Paolo Nobili, and Antonio Sassano. A lagrangian-based heuristic for large-scale set covering problems. *Math. Program.*, 81:215–228, 1998.

[11] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233235, 1979.

[12] Carlos A. Coello Coello, Daniel Cortés Rivera, and Nareli Cruz Cortés. Use of an artificial immune system for job shop scheduling. In *ICARIS*, pages 1–10, 2003.

[13] Broderick Crawford and Carlos Castro. Integrating lookahead and post processing procedures with aco for solving set partitioning and covering problems. In *ICAISC*, pages 1082–1090, 2006.

[14] Dasgupta, Dipankar, and Fernando Nino. *Immunological Computation: Theory and Applications*. Auerbach Publications, Boston, MA, USA, 1 edition, 2008.

[15] Dipankar Dasgupta, Senhua Yu, and Fernando Niño. Recent advances in artificial immune systems: Models and applications. *Appl. Soft Comput.*, 11(2):1574–1587, 2011.

[16] Leandro Nunes de Castro and Jon Timmis. Artificial immune systems as a novel soft computing paradigm. *Soft Comput.*, 7(8):526–544, 2003.

[17] Leandro Nunes de Castro and Fernando J. Von Zuben. The clonal selection algorithm with engineering applications. In *workshop on artificial immune systems and their applications*, volume 3637, 2000.

[18] Leandro Nunes de Castro and Fernando J. Von Zuben. ainet: An artificial immune network for data analysis. *Data Mining: a heuristic approach*, 1:231–259, 2001.

[19] Leandro Nunes de Castro and Fernando J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Trans. Evolutionary Computation*, 6(3):239–251, 2002.

[20] Orhan Engin and Alper Döyen. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Comp. Syst.*, 20(6):1083–1095, 2004.

[21] Marshall L. Fisher and Pradeep Kedia. Optimal solution of set covering/partitioning problems using dual heuristics. *management Science*, 36:674688, 1990.

[22] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[23] Riad Hadji, Malek Rahoual, El-Ghazali Talbi, and Vincent Bachelet. Ant colonies for the set covering problem. In *Abstract proceedings of ANTS*, pages 63–66, 2000.

[24] Mohamed Haouaria and Jouhaina Siala Chaouachia. A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem. *Journal of the Operational Research Society*, 53:792799, 2002.

[25] Paul K. Harmer, Paul D. Williams, Gregg H. Gunsch, and Gary B. Lamont. An artificial immune system architecture for computer security applications. *IEEE Trans. Evolutionary Computation*, 6(3):252–280, 2002.

[26] Emma Hart, Chris McEwan, Jon Timmis, and Andrew Hone. Advances in artificial immune systems. *Evolutionary Intelligence*, 4(2):67–68, 2011.

[27] Larry W. Jacobs and Michael J. Brusco. A local-search heuristic for large set-covering problems. *Naval Research Logistics*, 42:11291140, 1995.

[28] Jungwon Kim and Peter J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1330–1337, 2001.

[29] Guanghui Lan and Gail W. DePuy. On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the set covering problem. *Computer and Industrial Engineering*, 51:362374, 2006.

[30] 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.

[31] Zne-Jung Lee, Chou-Yuan Lee, and Shun-Feng Su. An immunity-based ant colony optimization algorithm for solving weapon-target assignment problem. *Appl. Soft Comput.*, 2(1):39–47, 2002.

[32] Lucas Lessing, Irina Dumitrescu, and Thomas Stützle. A comparison between aco algorithms for the set covering problem. In *ANTS Workshop*, pages 1–12, 2004.

[33] Mattias Ohlsson, Carsten Peterson, and Bo Söderberg. An efficient mean field approach to the set covering problem. *European Journal of Operational Research*, 133(3):583–595, 2001.

[34] 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 and Industrial Engineering*, 58(4):774–784, 2010.

[35] Xiaolei Wang, Xiao Zhi Gao, and Seppo J. Ovaska. Artificial immune optimization methods and applications - a survey. In *SMC (4)*, pages 3415–3420, 2004.

M. Sohel Rahman received his B.Sc. Engg. degree from the Department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2002 and the M.Sc. Engg. degree from the same department, in 2004. He received his Ph.D. degree from the Department of Computer Science, King's College London, UK in 2008. He is currently serving as a Professor in the Department of CSE, BUET. His research interests include String and sequence algorithms, Bioinformatics, Musicolgy, Design and Analysis of Algorithms, Meta-heuristics etc.

Shahriar Rouf received his B.Sc. Engg. degree from the Department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2011. He participated in the International Mathematical Olympiad (IMO) 2005 and the ACM-ICPC World Finals 2008 and 2009. He served as a Lecturer in the Department of CSE, BUET. His research interests include Design and Analysis of Algorithms, Computational Geometry, Computer Graphics etc.

Masruba Tasnim received her B.Sc. Engg. degree from the Department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2011. She served as a Lecturer in the Department of CSE, BUET. Her research interests include Design and Analysis of Algorithms, Bioinformatics, Data Mining etc.