An Artificial Bee Colony Algorithm for Minimum Weight Dominating Set

C G Nitash*, Alok Singh†
School of Computer and Information Sciences
University of Hyderabad, Hyderabad 500 046
Telangana, India
Email: *cgnitash@gmail.com, †alokcs@uohyd.ernet.in

***Abstract*—The minimum weight dominating set (MWDS) problem is a classic NP-Hard optimisation problem with a wide range of practical applications. As a result, many algorithms have been proposed for this problem. Several greedy and approximation algorithms exist which provide good results for unit disk graphs with smooth weights. However, these algorithms do not perform well when applied to general graphs. There are a few metaheuristics in the literature such as genetic algorithms and ant colony optimisation algorithm, which also work for general graphs. In this paper, a swarm intelligence algorithm called artificial bee colony (ABC) algorithm is presented for the MWDS problem. The proposed ABC algorithm is compared with other metaheuristics in the literature and shown to perform better than any of these metaheuristics, both in terms of solution quality and time taken.**

I. Introduction

Given an undirected graph $G = (V, E)$, where V is the set of vertices or nodes and E is the set of edges, a dominating set is a subset $D \subseteq V$ such that every vertex $v \in V$ is either a member of D or is adjacent to a member of D. Vertices belonging to D are called dominating vertices, whereas those vertices which do not belong to D are called non-dominating vertices. A dominating set D' is a minimum dominating set if it has the minimum cardinality among all dominating sets. Given a non-negative weight function $w : V \rightarrow \Re^+$ associated with the vertices of G, a minimum weight dominating set (MWDS) is a dominating set D', that minimises $\sum_{v \in D'} w(v)$. Throughout this paper, the terms 'node' and 'vertex' mean the same and are used interchangeably.

Dominating sets are used in many domains, e.g. to construct routing backbones and for clustering in wireless networks [1], to help in gateway placement in wireless mesh networks [2] and to determine the nodes in a network that actively participate in intrusion detection [3]. MWDS has also been used in information retrieval for multi-document summarisation [4], where the sentences are nodes in a graph and similar sentences have an edge between them. In addition, if a structured web database is modelled as an attribute value graph, MWDS can be used for extracting a set of queries that can efficiently harvest data records from web databases [5].

Several distributed greedy algorithms have been proposed which efficiently compute dominating sets of minimum cardinality [6]–[9]. However, in heterogeneous wireless networks with non-uniform capabilities, individual nodes have different priorities according to real time factors such as available bandwidth, residual battery power, computational resources, etc. In this scenario, it is better to find a dominating set that minimises the sum of the weight of the nodes in the network, which is the problem of finding a MWDS. The MWDS problem has been shown to be $\mathcal{NP}$-Hard [10]. Hence, exact algorithms are not efficient for solving MWDS. Several approximation algorithms and also a Polynomial Time Approximation Scheme (PTAS) have been proposed for MWDS [1], [11]–[13]. However, these algorithms only work for unit-disk graphs with smooth weights, i.e., the ratio of the weight of adjacent nodes is bounded from above by a constant. The best approximation factor for unit-disk graphs without smooth weights is $(4 + \epsilon)$ times the optimal solution [12]. The best known approximation ratio for general graphs is $O(\log n)$ [14].

For problems that are not amenable to an exact solution, heuristics and metaheuristics are very useful. In fact, metaheuristic algorithms have been known to efficiently solve a variety of $\mathcal{NP}$-hard optimisation problems [15], [16]. The artificial bee colony (ABC) algorithm has already been shown to be an effective metaheuristic for optimisation of numerous $\mathcal{NP}$-Hard problems [17]. This has motivated us to develop an ABC algorithm based approach for MWDS. Already there exists three metaheuristic techniques in the literature which show good results for general graphs, viz. two ant colony optimisation techniques (One proposed in [18] and another in [19]) and a hybrid genetic algorithm [19]. We have compared our ABC approach with two best metaheuristic technique among these three techniques and show that our approach performs better. The results also show that the time taken for our approach grows slowly compared to the previous approaches.

The remaining part of this paper is organised as follows: In Section II, we outline some of the heuristic and metaheuristic approaches that have been proposed in the literature. Section III provides an overview of the ABC algorithm. Section IV describes our proposed ABC approach for MWDS. In Section V, we describe the computational experiments and compare the experimental results of our ABC approach with those of various metaheuristic approaches. Finally, we end the paper with our conclusions in Section VI.

II. Related Work

There is a distributed algorithm proposed in [1] which uses a heuristic on the neighbourhood of the node to compute the minimum weight dominating set by a greedy method. For unit-disk graphs with smooth weights a PTAS is described in [13] which uses the 2-hop neighbourhood of each node to

978-1-4799-4458-3/14/$31.00 ©2014 IEEE

compute the dominating set. These algorithms do not find good solutions for larger graphs or for general graphs. Also, due to the need to calculate the 2-neighbourhood information for each node these methods do not perform well in terms of time as either the size or average degree of the graph increases.

A. Heuristics

For solving MWDS some useful heuristics have been proposed in [19], [20] that perform well in terms of time but not solution quality. Hence, they are used to guide the meta-heuristic search process, by applying the heuristic to small subgraphs of the original graph. The working principle of the heuristics is as follows.

Heuristic Procedure. Initially the dominating set is empty, i.e., $D = \phi$ and all nodes are marked as non-dominated. From the set of nodes not in D, we select a node v according to one of the following equations.

$$v \leftarrow arg\, max_{v \in V-D} \frac{d(v)}{w(v)} \tag{1}$$

$$v \leftarrow arg\, max_{v \in V-D} \frac{W(v)}{w(v)} \tag{2}$$

$$v \leftarrow arg\, max_{v \in V-D} \frac{W(v) \times d(v)}{w(v)} \tag{3}$$

where $w(v)$ is the weight of the node, $d(v)$ is the degree of the node and $W(v)$ is the sum of the weights of the non-dominated neighbours of v. The selected node v is then added to D and all the neighbours of v are marked as dominated. This whole procedure is repeated until no non-dominated nodes remain.

The performance of the above heuristics have been compared to each other and were found to have similar solution quality. While the performance of these heuristics is not comparable to any of the meta-heuristics in the literature, these heuristics are an efficient and useful method to approximate the optimal dominating set for a small subgraph of the original graph. As such, we have used these heuristics in our approach to guide the search process.

B. Metaheuristics

An ant colony optimisation algorithm is proposed in [18]. In this algorithm, the graph is converted into a complete graph to initialise the pheromone trails. Edges that are present in the original graph are assigned a weight 1, otherwise they are assigned a weight 0. A value of 0 represents a covered neighbour and a value of 1 represents an uncovered neighbour. As a node is added to the dominating set, the weights of all edges incident on just added node and its neighbours are set to 0 thereby marking them as covered. Iteratively a dominating set is constructed where a node having the maximum ratio of sum of weights of its uncovered neighbors to its weight is added to the dominating set during each iteration. This is used to initialise the pheromone values. The state transition rule that is used by ants to determine the next node to be added to the dominating set employs both pheromone and heuristic information. In addition to the usual global pheromone update rule, a local pheromone update rule is also used. The global pheromone update rule updates the pheromone values at the end of an iteration when all ants have constructed their solutions. The local pheromone update rule changes pheromone values on a node as soon as it is selected for inclusion in the dominating set.

Another ant colony optimisation algorithm is proposed in [19]. This algorithm is hybridized with a local search. This algorithm initialises pheromone values on all nodes to 10. The state-transition rule used rely only on the pheromone values of uncovered nodes in the graph. No heuristic information is used by the state-transition rule. An ant selects the next node for addition into the dominating set with probability $p(v) = \tau_v / \sum_{u \in S} \tau_u$, where S is the set of non-dominating nodes. Only global pheromone update rule is employed. At the end of each iteration, the pheromone values of nodes in the best dominating set of current iteration are updated by $\tau_v = \tau_v \rho + 2.0/(5.0 + f - F)$, where f is the fitness of the best solution in the current iteration and F is the fitness of the best solution found so far. For nodes not belonging to the best dominating set, pheromone values are updated as $\tau_v = \tau_v \rho$; $0 \leq \rho \leq 1$, where ρ is the pheromone persistence rate. Solution constructed through an ant is improved further through a local search. The local search removes all redundant nodes from the solution. A dominating node is considered redundant if the new solution obtained after its removal is still a dominating set. Redundant nodes are removed one-by-one by following either a greedy strategy or a random strategy. The greedy strategy is used with certain probability, otherwise the random strategy is used. The greedy strategy selects a redundant node using $arg\, max_{r \in R} w(r)/d(r)$, where R is the set of redundant nodes and $d(r)$ is the degree of node r, so that nodes with higher weights and smaller degree are likely to be removed by this strategy. The random strategy removes a redundant node randomly. Once a redundant node has been removed, the set of redundant node is re-computed. The local search stops when no redundant node remains. This approach is henceforth referred to as ACO-LS.

An additional modification was used with ACO-LS, which includes a pre-processing step immediately after pheromone initialisation. This is done by randomly generating 100 maximal independent sets (MIS) which are also dominating sets. The pheromone values on the nodes that are part of these MIS are updated using pheromone update rule described in the previous paragraph for nodes belonging to the best dominating set. This pre-processing step was used to reduce the time taken for the algorithm to converge, while maintaining the quality of the solutions obtained. Henceforth this algorithm is referred to as ACO-PP-LS.

A hybrid steady-state genetic algorithm is also proposed in [19]. In this algorithm the initial population is generated randomly. A new member of population is usually generated through crossover and mutation. However, with a small probability, a new member is generated randomly in a bid to diversify the population. A fitness based crossover is employed, which ensures that bits of the more fit parent are more likely to be inherited by the child. The new solution obtained through crossover is subjected to mutation where each bit of the solution is flipped with a small probability. Since a newly obtained solution may not be a dominating set, it is repaired by iteratively adding nodes from the set S of nodes not in the newly obtained solution according to one of the two strategies.

With a certain probability, the first strategy is used, otherwise the second strategy is used. The first strategy is greedy and add a node using $arg\,max_{s \in S} W(s)/w(s)$, where $w(s)$ is the weight of node s and $W(s)$ is the sum of the weights of the uncovered nodes adjacent to s. The second strategy add a node from S randomly. The feasible solution thus obtained is improved further by the same local search as used in ACO-LS and ACO-PP-LS. The worst member of the population is replaced by this newly generated solution. Henceforth this algorithm is referred to as HGA.

The distributed algorithm and PTAS described above work only for unit-disk graphs with smooth weights. All the metaheuristics described above also work for general graphs and are the best available in the literature. Hence, these metaheuristics have been used for comparison with our approach.

III. Overview of ABC Algorithm

The artificial bee colony (ABC) approach is inspired by the foraging behaviour of natural honey bee colonies [21], [22]. This approach is proposed by Karaboga [23]. Actually, in a bee colony, there are three types of foraging bees, viz. *scouts*, *employed* and *onlooker* bees. Initially *scouts* spread out radially from the hive and explore the surroundings for food sources. When a food source is found, the *scout* bee becomes an *employed* bee and starts collecting nectar. The *employed* bee continues to exploit the food source until it is depleted. The employed bee shares the information it has obtained about the food source, i.e., the direction of the food source, its distance, the food source quality etc. by performing a dance at the hive. This dance is observed by *onlooker* bees at the hive, each of which then combine the information from several *employed* bees and use this information to choose a food source with a probability that favors good food sources over bad ones. Once an *onlooker* bee choose a food source, it becomes employed. When a food source is exhausted, the *employed* bees at that source becomes either a *scout* bee and or an *onlooker* bee. The various roles are performed by members of the colony throughout the colony's life span. Scout bees which search for new food sources can be perceived as performing the job of exploration, whereas employed and onlooker bees can be perceived as performing the job of exploitation.

The ABC algorithm has been applied to a variety of optimisation problems with good results. The performance of ABC algorithm has been found to be comparable with other metaheuristic algorithms like genetic algorithms, particle swarm optimisation, differential evolution etc. [24]. Original ABC algorithm was designed for optimisation in continuous domain. It was extended in [25] for discrete optimisation problems. A good survey about ABC algorithm and its applications can be found in [17]. ABC algorithm also classifies the (artificial) bees into scout, employed and onlooker bees with similar functions. In ABC algorithm, a food source and its nectar content correspond respectively to a potential solution to the problem currently being solved and its fitness. There exists a one-to-one correspondence between employed bees and food sources. Frequently, though not always, the number of onlooker bees is same as the number of employed bees. Basically, ABC algorithm consists of three broad phases, viz. the scout bee phase, the employed bee phase and the onlooker bee phase. The outline of this algorithm is as follows.

- **Initialise:** Each scout bee generates a solution randomly and becomes employed by associating with this newly generated solution.
- **Repeat:**
  - Each employed bee search for a solution in the neighbourhood of its associated solution and move to the newly found solutions in case it is better than its currently associated solution.
  - Onlooker bees tend to select employed bee solutions according to their fitness and then search for new solutions in the neighbourhood of their respective selected solutions. If the new solution found by an onlooker bee is better than its selected employed bee solution then the the selected employed bee solution is replaced with this new solution.
  - If an employed bee solution is not improved over certain number of iterations then the associated employed bee is turned into an scout. This scout generates a new solution immediately and becomes employed again by associating with this newly generated solution.
- **Until:** termination conditions are met.

As can be seen from this outline, the three phases are not necessarily distinct and can be interleaved with each other. This outline can then be refined for specific problem domains.

Algorithm 1 Procedure $ABC(G)$

```
1: for i ← 1 to N_emp do
2:     S_i ← Scout()
3: end for
4: repeat
5:     for i ← 1 to N_emp do
6:         S' ← Employ(S_i)
7:         if cost(S') < cost(S_i) then
8:             S_i ← S'
9:         end if
10:    end for
11:    for i ← 1 to N_onl do
12:        S_k ← Onlooker({S_1, S_2, ..., S_{N_emp}})
13:        S' ← Employ(Sk)
14:        if cost(S') < cost(S_k) then
15:            S_k ← S'
16:        end if
17:    end for
18:    S_best ← S_i : cost(S_i) ≤ cost(S_j); ∀S_j
19: until total iterations reach N_iter
20: return S_best
```

IV. ABC Approach for MWDS

A. ABC() Procedure

The pseudo-code of our ABC() procedure is given in Algorithm 1. The ABC() procedure takes as input a graph $G = (V, E)$ and returns a dominating set for $G = (V, E)$. This procedure begins by generating a population of N_{emp} solutions $\{S_1, S_2, \ldots, S_{N_{emp}}\}$ using $Scout()$ and then an iterative process takes over. Then in each iteration, for each member

of the population, a neighbour is constructed by $Employ()$. This neighbour replaces the original solution if it has higher fitness (lower cost). Also, N_{onl} solutions are selected using the $Onlooker()$ procedure. These solutions are selected such that they represent more of the fitter solutions. These N_{onl} solutions are again processed by $Employ()$ in a bid to find higher fitness solutions. This process repeats for N_{iter} iterations, and finally, returns the best solution S_{best} found over all the iterations. It is to be noted that ABC() procedure never converts an employed bee into a scout during the iterations of the algorithm as computational experiments suggested against doing so. The $Scout()$ and $Employ()$ methods make use of $Repair()$ method to convert an infeasible solution into a feasible one with no redundant nodes. The methods $Scout()$, $Employ()$ and $Onlooker()$ used in the ABC() procedure along with $Repair()$ are described in the next subsection.

B. ABC Methods

This subsection provides the details of 3 methods, viz. $Scout()$, $Employ()$ and $Onlooker()$ used by ABC() procedure. In addition, $Repair()$ method is also described in detail.

1) $Scout()$: This method returns a randomly generated MWDS for the given graph. Initially, the dominating set D is empty and all the nodes are marked as non-dominated. A small percentage $perc_r$ of the nodes are randomly selected from the graph and added to D. For each node v that is added to D, all its adjacent nodes u, where $u \notin D$ are marked as dominated. Then the dominating set is repaired using the $Repair()$ method based on the heuristic procedure described in Section II-A. Finally, the dominating set D is minimised using a minimisation heuristic. For $perc_r = 0\%$, this method reduces to exactly the heuristic procedure described in Section II-A. On the other hand, for $perc_r = 100\%$ this method adds all the nodes in the graph to D and then minimises the set using the minimisation heuristic.

Algorithm 2 Procedure $Scout(perc_r)$

```
1: D ← φ
2: for i ← 1 to |V| * perc_r do
3:     D ← D ∪ Select_random(V)
4: end for
5: Repair(D)
6: return D
```

2) $Employ()$: This method takes as input a dominating set D and returns a new dominating set D' with a weight less than or equal to that of D. D' is constructed as follows. A certain percentage $perc_d$ of the nodes in the dominating set are removed. This destroyed dominating set is then repaired using the $Repair()$ method. This method constructs a dominating set in the neighbourhood of the original set by keeping some of the dominating nodes and repairing the set using only the non-dominated portion of the original graph. If the newly constructed dominating set has less weight than the original, the new set is returned. Otherwise, the original dominating set D is returned.

Algorithm 3 Procedure $Employ(D, perc_d)$

```
1: for i ← 1 to |D| * perc_d do
2:     v ← select_random(D)
3:     D ← D − v
4: end for
5: Repair(D)
6: return(D)
```

3) $Repair()$: This method takes as input a set of nodes D that may not be a dominating set and adds nodes to D until it becomes a dominating set. While D is not dominating, the next node to add is chosen according to one of the heuristics described in Section II-A. However, for each node, the heuristic used is randomly chosen from one of the three heuristics as shown in line 2 of Algorithm 4. This does not add extra computational burden as the information required for each of the heuristics is the same, yet it results in a wider exploration of the search space in comparison to the use of any single heuristic all the times. $Heuristic\_Value(v, h)$ is exactly the value of heuristic number h calculated at node v. Clearly, the computational time of ABC is dominated by $Repair()$. In the worst case, this time is the same as the heuristic procedure, which takes polynomial time. Once a dominating set is constructed it may contain some redundant nodes. A redundant node v is one that if removed from a dominating set D still leaves D as a dominating set. These redundant nodes are pruned according to the greedy strategy ($v \leftarrow arg\,max_{v \in R} \frac{w(v)}{d(v)}$) described in [19].

Algorithm 4 Procedure $repair(D)$

```
1: repeat
2:     h ← Select_Random_Heuristic()
3:     max ← 0
4:     for v ∉ D do
5:         if Heuristic_Value(v, h) ≤ max then
6:             max ← Heuristic_Value(v, h)
7:             v' = v
8:         end if
9:     end for
10:    D = D ∪ v'
11: until D is a dominating set
12: R ← {v : D − {v} is still a dominating set}
13: while R ≠ φ do
14:    v' ← arg max_{v∈R} w(v)/d(v)
15:    D ← D − v'
16:    recalculateR
17: end while
```

4) $Onlooker()$: This method chooses relatively fit solutions from the population of solutions as a whole. We have used binary tournament selection to choose this solution as it is computationally cheaper and gives comparable results to other selection strategies such as rank based selection [26]. In this method, two members of the population are randomly picked and the member having higher fitness between the two is returned. The purpose of this method is to accelerate the search process in the direction of globally best solutions.

V. Experimentation and Results

A. Experimentation

We have implemented our ABC approach in C++ and executed it on an on an Intel core i5-2400 processor based system with 4 GB RAM running under Fedora 16 at 3.10GHz. The experiments were conducted on the same data sets as used in [18] and [19]. The data set used in [18] consists of two types of graphs, both of which are connected and undirected. In Type I graphs, the weights are randomly assigned from the interval $[20, 70]$. In Type II graphs, the weight of a node v is randomly assigned from the interval $[1, d(v)^2]$, where $d(v)$ is the degree of node v. For both Type I and Type II graphs, the number of nodes varies from 50 to 1000, and for each node size, the average degree of the graph is varied from 2 to 40. For each combination of number of nodes and number of edges considered, 10 instances were generated. A total of 530 instances (53×10) were generated for each of Type I and Type II datasets leading to a grand total of 1060 instances in these two datasets. For both Type I and Type II datasets, instances with 50 to 250 nodes are classified as small instances and instances with 300 to 1000 nodes are classified as large instances. In addition to Type I and II instances, UDG (unit disk graph) instances were also used in [19]. The nodes are distributed randomly in an area of 1000×1000 units in these instances. The number of nodes in these instances belong to $\{50, 100, 250, 500, 750, 1000\}$. Transmission range of all nodes are fixed to either 150 or 200 units. Similar to Type I and Type II instances, 10 instances are generated for each combination of number of nodes and transmission range. This leads to a total of 120 UDG instances. All these instances can be downloaded from http://dcis.uohyd.ernet.in/~apcs/wmds/wmds-data-org.tar.gz. The various algorithms were executed once on each of these instances and the results were averaged over 10 instances with same characteristics.

B. ABC Parameters

We have experimented with several combination of parameter values for the ABC algorithm for both Type I & II as well as UDG instances and used those which provide the best results. The number of employed bees is $N_{emp} = 20$ and number of onlooker bees $N_{onl} = 10$. For the $Scout()$ method, a percentage upto 20% was found to produce random solutions with good fitness, i.e., $0\% \leq perc_r \leq 20\%$. Since $perc_r$ can be 0%, the exact heuristic solution may also be generated. For the $Employ()$ method all possible values of $perc_d$ above 10% , i.e., $10\% \leq perc_d \leq 100\%$, were found to produce neighbours with higher fitness. High values of $perc_d$ also serve the purpose of acting like the $Scout()$ method, thereby exploring some of the search space that might have been ignored in the initial scout phase. We have run the algorithm for various number of iterations. For upto 15 iterations the algorithm finds good solutions for graphs of upto 200 nodes. For graphs with 250 nodes between 20 to 25 iterations were needed. For graphs between 300 and 1000 nodes, we have found that 40 iterations gives better solutions than those published in the literature. Additional number of iterations increases the solution quality, but at the expense of longer computational time than previous approaches. Hence, we have used 40 iterations for all the graphs in the data set. This means that our ABC approach generates 1,200 solutions as compared to 20,000 solutions for the previous algorithms proposed in [18], [19]. For UDG instances, we have used fewer employed and onlooker bees, i.e., $N_{emp} = 10$ and $N_{onl} = 5$. However, we have increased the number of iterations to 80. This ensures generation of 1,200 solutions even for the UDG instances. The remaining parameters have been kept the same as they were found to give good results.

TABLE I. MWDS results for small Type I graphs

Nodes	Edges	HGA		ACO-PP-LS		ABC	
		Avg	Time	Avg	Time	Avg	Time
50	50	**531.3**	0.98	**531.3**	0.4	534	0
50	100	**371.2**	0.87	**371.2**	0.3	**371.2**	0
50	250	**175.7**	0.77	176	0.2	**175.7**	0
50	500	**94.9**	0.8	**94.9**	0.2	**94.9**	0
50	750	**63.1**	0.7	**63.1**	0.1	**63.1**	0
50	1000	**41.5**	0.7	**41.5**	0.1	**41.5**	0
100	100	1081.3	3.45	**1066.9**	1.4	1077.7	1.3
100	250	626.2	3.3	627.2	1.1	**621.6**	1
100	500	358.3	2.61	362.5	0.9	**356.4**	0
100	750	261.2	2.02	263.5	0.8	**255.9**	0
100	1000	205.6	2.31	209.2	0.8	**203.6**	0
100	2000	108.2	2.86	**108.1**	0.7	108.2	0
150	150	1607	9.31	**1582.8**	3.5	1607.9	4.1
150	250	1238.6	8.6	1237.2	3.1	**1231.2**	2.8
150	500	763	6.68	767.7	2.4	**752.1**	1.8
150	750	558.5	5.69	565	2.1	**549.3**	1
150	1000	438.7	5.9	446.8	2	**435.1**	1
150	2000	245.7	4.74	259.4	1.7	**242.2**	1
150	3000	169.2	4.74	173.4	1.6	**167.8**	1
200	250	1962.1	17.99	**1934.3**	6.6	1941.1	6.6
200	500	1266.3	14.81	1259.7	5.1	**1246.9**	4
200	750	939.8	13.74	938.7	4.5	**923.7**	3
200	1000	747.8	12.45	751.2	4	**730.4**	2.8
200	2000	432.9	8.61	440.2	3.4	**417.6**	2
200	3000	308.5	8.27	309.9	3.1	**294.4**	2
250	250	2703.4	31.23	**2655.4**	12.2	2685.7	15.7
250	500	1878.8	27.08	1850.3	9.9	**1836**	8
250	750	1421.1	26.55	1405.2	8.4	**1391.9**	5.9
250	1000	1143.4	23.34	1127.1	7.5	**1115.3**	5
250	2000	656.6	14.64	672.8	6	**630.5**	3.8
250	3000	469.3	13.8	474.1	5.4	**454.9**	3
250	5000	300.5	12.92	310.4	4.8	**292.4**	3

TABLE II. MWDS results for large Type I graphs

Nodes	Edges	HGA		ACO-PP-LS		ABC	
		Avg	Time	Avg	Time	Avg	Time
300	300	3255.2	52.64	**3198.5**	19.1	3240.7	23.6
300	500	2509.8	49.77	**2479.2**	16.3	2484.6	13.9
300	750	1933.9	44.43	1903.3	14.3	**1901.4**	10.1
300	1000	1560.1	40.54	1552.5	12.4	**1523.4**	8.1
300	2000	909.6	26.6	916.8	9.5	**875.5**	5.9
300	3000	654.9	22.43	667.8	8.6	**635.3**	5
300	5000	428.3	17.13	437.4	7.5	**411**	4.4
500	500	5498.3	204.83	**5398.3**	77.5	5480.1	78.1
500	1000	3798.6	250.38	3714.8	81	**3707.6**	38.7
500	2000	2338.2	145.95	2277.6	54.4	**2266.1**	22.8
500	5000	1122.7	75.35	1115.3	30.1	**1070.9**	15.3
500	10000	641.1	43.62	652.8	24.4	**596**	13.1
800	1000	8017.7	841.64	8117.6	409.2	**7907.3**	208
800	2000	5318.7	576.34	5389.9	292.2	**5193.2**	95.1
800	5000	2633.4	346.05	2616	148.9	**2548.6**	50.9
800	10000	1547.7	162.32	1525.7	95.9	**1471.7**	40.6
1000	1000	11095.2	2193.48	11035.5	922.4	**10992.4**	589
1000	5000	3996.6	626.1	4012	326.5	**3853.7**	95.8
1000	10000	2334.7	380.27	2314.9	194.7	**2215.9**	69.8
1000	15000	1687.5	254.64	1656.3	150.8	**1603.2**	64
1000	20000	1337.2	174.5	1312.8	139.2	**1259.5**	60.4

TABLE III. MWDS RESULTS FOR SMALL TYPE II GRAPHS

Nodes	Edges	HGA		ACO-PP-LS		ABC	
		Avg	Time	Avg	Time	Avg	Time
50	50	**60.8**	1.27	**60.8**	0.4	**60.8**	0
50	100	**90.3**	1.04	**90.3**	0.3	**90.3**	0
50	250	**146.7**	1.46	**146.7**	0.3	**146.7**	0
50	500	**179.9**	0.93	**179.9**	0.2	**179.9**	0
50	750	**171.1**	1.25	**171.1**	0.2	**171.1**	0
50	1000	**146.5**	0.95	**146.5**	0.1	**146.5**	0
100	100	124.5	4.13	**123.5**	1.5	124.4	2
100	250	211.4	4.47	210.4	1.3	**209.6**	1.1
100	500	306	4.9	308.4	1.2	**305.8**	1
100	750	385.3	3.74	386.3	1	**384.5**	0.3
100	1000	429.1	4.47	430.3	1	**427.3**	0
100	2000	**550.6**	4.68	559.8	0.9	**550.6**	0
150	150	186	12.01	**184.9**	3.5	185.9	6.1
150	250	234.9	12.68	**233.4**	3.4	**233.4**	5
150	500	350	11.46	351.9	2.9	**349.5**	3.3
150	750	455.8	10.12	454.7	2.6	**453.7**	2.4
150	1000	**547.5**	8.88	549	2.3	547.8	2
150	2000	**720.1**	8.96	725.7	2.1	**720.1**	1
150	3000	**792.6**	29.94	806.2	7	793.2	1
200	250	275.1	21.77	**272.6**	6.3	273.5	11.9
200	500	390.7	22.22	388.4	5.7	**387.6**	8.6
200	750	507	22.33	501.4	5.4	**498.5**	6.5
200	1000	601.1	17.92	605.8	4.9	**599.3**	5.2
200	2000	893.5	16.93	892.9	4.3	**885.5**	3.5
200	3000	**1021.3**	18.37	1034.4	4.3	**1021.3**	2.8
250	250	310.1	41.76	**306.7**	12.1	308.6	23.7
250	500	444	43.77	443.2	11.2	**442.6**	18
250	750	578.2	39.3	575.9	10.5	**569.9**	13.7
250	1000	672.8	39.05	675.1	10	**670.3**	11.5
250	2000	1030.8	31.97	1031.5	8.5	**1010.4**	6.9
250	3000	1262	29.76	1277	8.2	**1251.3**	5.6
250	5000	1480.9	29.69	1520.1	6.9	**1464.7**	4

TABLE IV. MWDS RESULTS FOR LARGE TYPE II GRAPHS

Nodes	Edges	HGA		ACO-PP-LS		ABC	
		Avg	Time	Avg	Time	Avg	Time
300	300	375.6	69.7	**371.1**	19.2	373.5	41.1
300	500	484.2	72.5	**481.2**	18.1	481.6	28.2
300	750	623.8	64.33	618.3	17	**617.6**	26.1
300	1000	751.1	59.28	**743.5**	16	743.6	20.2
300	2000	1106.7	55.25	1107.5	14.3	**1095.9**	11.6
300	3000	1382.1	46.14	1415.3	13.6	**1361.7**	9.1
300	5000	1686.3	44.85	1698.6	11.7	**1682.7**	6.9
500	500	632.9	284.9	**627.3**	73.9	630.4	162.3
500	1000	919.2	268.61	912.6	68.5	**906.7**	108.3
500	2000	1398.2	233.33	1383.9	65	**1383.6**	66
500	5000	2393.2	122.3	2468.8	51	**2337.9**	31
500	10000	3264.9	118.96	3369.4	37.6	**3211.5**	21.1
800	1000	1128.2	1091.83	1125.1	268.3	**1119.2**	505
800	2000	1679.2	927.69	1697.9	282.2	**1656.4**	338.9
800	5000	3003.6	525.34	3120.9	224	**2917.4**	127.3
800	10000	4268.1	387.92	4447.9	148.2	**4121.3**	70.9
1000	1000	1265.2	2149.54	1258.6	514.2	**1256.2**	1173.7
1000	5000	3320.1	1112.49	3415.1	461.3	**3240.7**	303.6
1000	10000	4947.5	739.45	5101.9	324.4	**4781.2**	150.6
1000	15000	6267.6	617.16	6470.6	251.8	**5934**	111.9
1000	20000	7088.5	536.83	7340.8	213.8	**6729**	95.3

C. Results

We have compared the results of our approach with HGA and ACO-PP-LS approaches of [19], which are the best available metaheuristic approaches in the literature. Results of the ACO approach of [18] were quite inferior to those of HGA and ACO-PP-LS approaches and that is why we have not included ACO approach of [18] in our comparison. Tables I-IV contain the solution quality and time taken for each of the three approaches on Type I & II instances. The results for UDG instances are given in Table V. The Avg column contains the average solution values for each of the 10 instances of graphs of a particular size. The bold values indicate the best average solution quality among the different approaches compared. The time columns report the average execution time (over 10 instances) for each of the algorithms. The execution times for HGA and ACO-PP-LS are different than those published in [19] as we have re-executed these algorithms on the same system as the ABC algorithm to provide a fair comparison of execution times.

TABLE V. MWDS RESULTS FOR UDG GRAPHS

Nodes	Range	HGA		ACO-PP-LS		ABC	
		Avg	Time	Avg	Time	Avg	Time
50	150	394.3	0.48	**393.9**	0.3	**393.9**	0.01
50	200	**247.8**	0.45	**247.8**	0.3	**247.8**	0.01
100	150	450.4	1.58	**449.7**	1.1	**449.7**	0.3
100	200	217.3	1.47	**216**	0.8	**216**	0.22
250	150	294.2	8.99	294.5	6.1	**293.9**	2.6
250	200	119.1	5.05	**118.9**	5.2	119	2.1
500	150	172.7	27.05	170.9	23.4	**170.7**	6.7
500	200	68.1	22.01	**68**	19.7	**68**	5.9
800	150	115.9	79.82	**114**	63.5	**114**	13.9
800	200	42.6	64.8	**40.8**	50.9	**40.8**	12.56
1000	150	125	115.31	121.9	97.5	**121.4**	28.4
1000	200	47.2	83.61	46.2	78.4	**46.1**	23.5

For small Type I graphs, ABC finds better solutions than the other approaches, except for graphs with very small average degree. The time taken for ABC is also less than for HGA and ACO-PP-LS. The results are similar for large Type I graphs, where ABC finds better solutions in less time. For small Type II graphs, ABC finds better solutions in most cases in comparison to HGA or ACO-PP-LS. ABC finds solutions close to those of HGA for graphs with very high degree, whereas ABC returns similar results as ACO-PP-LS for graphs with very small degree. Also, the time taken by ABC is more than ACO-PP-LS for smaller degree graphs. For large Type II graphs, ABC is better than HGA or ACO-PP-LS as the size of the graphs increase. While the time taken for small degree graphs is more for ABC, the solution quality is better for graphs of size 800 and 1000 nodes. For both Type I and Type II graphs with 50 nodes, all three approaches return similar solutions.

For small UDG instances with 50 and 100 nodes, ABC provides the same solution quality as ACO-PP-LS on all instances. On the other hand, HGA performs as well as ABC only on instances with 50 nodes and transmission range 200. On remaining instances, solution quality of HGA is always inferior in comparison to ABC. For larger UDG instances with 250, 500, 800 and 1000 nodes, ABC obtains better solution quality in 5 and 4 cases in comparison to HGA and ACO-PP-LS respectively, worst solution quality in comparison to ACO-PP-LS in one case and solutions equivalent to HGA and ACO-PP-LS in remaining cases. In all the cases, the time taken by ABC is considerably less than the previous approaches.

VI. CONCLUSION

We have presented an artificial bee colony (ABC) algorithm based approach to solve the minimum weight dominating set

problem. This approach has been compared with two best metaheuristic approaches available in the literature, viz. a genetic algorithm based approach and an ant colony optimisation based approach. We have compared the various approaches on the benchmark instances available in the literature and shown that the ABC approach performs better than other approaches, both in terms of solution quality and time taken. ABC approach explores the search space more efficiently by combining a number of heuristics to guide the search process. This also drives the search process towards the optimal solution faster. Except for small degree graphs, we have shown that the proposed ABC approach works better than the available metaheuristics.

As a future work, we intend to extend our approach to other variations of dominating set problem. A similar ABC approach can be developed for minimum weight vertex cover problem also.

REFERENCES

[1] Y. Wang, W. Wang, and X.-Y. Li, "Distributed low-cost backbone formation for wireless ad hoc networks," in *MobiHoc*, vol. 5, 2005, pp. 25–27.

[2] B. Aoun, R. Boutaba, Y. Iraqi, and G. Kenward, "Gateway placement optimization in wireless mesh networks with qos constraints," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 11, pp. 2127–2136, 2006.

[3] D. Subhadrabandhu, S. Sarkar, and F. Anjum, "Efficacy of misuse detection in ad hoc networks," in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004 (IEEE SECON 2004)*. IEEE, 2004, pp. 97–107.

[4] C. Shen and T. Li, "Multi-document summarization via the minimum dominating set," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 984–992.

[5] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma, "Query selection techniques for efficient crawling of structured web sources," in *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE, 2006, pp. 47–47.

[6] C. R. Lin and M. Gerla, "Adaptive clustering for mobile wireless networks," *Selected Areas in Communications, IEEE Journal on*, vol. 15, no. 7, pp. 1265–1275, 1997.

[7] S. Basagni, "Distributed clustering for ad hoc networks," in *Parallel Architectures, Algorithms, and Networks, 1999.(I-SPAN'99) Proceedings. Fourth InternationalSymposium on*. IEEE, 1999, pp. 310–315.

[8] K. Erciyes, O. Dagdeviren, D. Cokuslu, and D. Ozsoyeller, "Graph theoretic clustering algorithms in mobile ad hoc networks and wireless sensor networks," *Appl. Comput. Math*, vol. 6, no. 2, pp. 162–180, 2007.

[9] F. G. Nocetti, J. S. Gonzalez, and I. Stojmenovic, "Connectivity based k-hop clustering in wireless networks," *Telecommunication systems*, vol. 22, no. 1-4, pp. 205–220, 2003.

[10] R. G. Michael and D. S. Johnson, "Computers and intractability: A guide to the theory of np-completeness," *WH Freeman & Co., San Francisco*, 1979.

[11] D. Dai and C. Yu, "A 5+ϵ-approximation algorithm for minimum weighted dominating set in unit disk graph," *Theoretical Computer Science*, vol. 410, no. 8, pp. 756–765, 2009.

[12] F. Zou, Y. Wang, X.-H. Xu, X. Li, H. Du, P. Wan, and W. Wu, "New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs," *Theoretical Computer Science*, vol. 412, no. 3, pp. 198–208, 2011.

[13] X. Zhu, W. Wang, S. Shan, Z. Wang, and W. Wu, "A ptas for the minimum weighted dominating set problem with smooth weights on unit disk graphs," *Journal of combinatorial optimization*, vol. 23, no. 4, pp. 443–450, 2012.

[14] C. Ambühl, T. Erlebach, M. Mihalák, and M. Nunkesser, "Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2006, pp. 3–14.

[15] T. G. Stützle, *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. Infix Sankt Augustin, Germany, 1999, vol. 220.

[16] I. H. Osman and J. P. Kelly, "Meta-heuristics: an overview," in *Meta-Heuristics*. Springer, 1996, pp. 1–21.

[17] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: artificial bee colony (ABC) algorithm and applications," *Artificial Intelligence Review*, vol. 42, no. 1, pp. 21–57, 2014.

[18] R. Jovanovic, M. Tuba, and D. Simian, "Ant colony optimization applied to minimum weight dominating set problem," in *Proceedings of the 12th WSEAS international conference on automatic control, modelling & simulation*. World Scientific and Engineering Academy and Society (WSEAS), 2010, pp. 322–326.

[19] A. Potluri and A. Singh, "Hybrid metaheuristic algorithms for minimum weight dominating set," *Applied Soft Computing*, vol. 13, no. 1, pp. 76–88, 2013.

[20] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[21] A. Manning, "Some aspects of the foraging behaviour of bumble-bees," *Behaviour*, pp. 164–201, 1956.

[22] C. D. Michener, *The social behavior of the bees: a comparative study*. Harvard University Press, 1974, vol. 73, no. 87379.

[23] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," *Technical Report TR06, Erciyes University Press, Erciyes*, 2005.

[24] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (abc) algorithm," *Applied soft computing*, vol. 8, no. 1, pp. 687–697, 2008.

[25] A. Singh, "An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem," *Applied Soft Computing*, vol. 9, pp. 625–631, 2009.

[26] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.