# Ant Colony Optimization Applied to Minimum Weight Dominating Set Problem

Raka JOVANOVIC
Texas AM University
at Qatar
PO Box 23874, Doha
QATAR
rakabog@yahoo.com

Milan TUBA
Faculty of Computer Science
Megatrend University Belgrade
Bulevar umetnosti 29
SERBIA
tubamilan@ptt.rs

Dana SIMIAN
Department of Computer Science
Lucian Blaga University of Sibiu
5-7 dr. I. Ratiu str.
ROMANIA
d_simian@yahoo.com

*Abstract:* - In this paper we present an application of ant colony optimization (ACO) to the Minimum Weighted Dominating Set Problem. We introduce a heuristic for this problem that takes into account the weights of vertexes being covered and show that it is more efficient than the greedy algorithm using the standard heuristic. Further we give implementation details of ACO applied to this problem. We tested our algorithm on graphs with different sizes, edge densities, and weight distribution functions and shown that it gives greatly improved results over these acquired by the greedy algorithms.

*Key-Words:* - Ant Colony Optimization, Dominating Set Problem, Optimization Problems, Population Based Algorithms

## 1 Introduction

A dominating set for graph $G= (V, E)$ is a subset of vertices $W \in V$, such that every vertex in $V \setminus W$ is adjacent to some vertex in $W$. We call vertexes $u$, $v$ adjacent if there exists an edge $(u, v) \in E$. The minimum dominating set problem (MDSP) is to find a smallest possible dominating set in a graph. The minimum weight dominating set problem (MWDSP) is defined in the case that weights $c_i$ are added to vertexes $v_i$ and instead of finding $W$ with the minimum number of elements we search for $W$ that has the minimum sum of weights. The existence of a dominating set of $k$ elements is one of the classical NP-complete decision problems [1]. MDS for general graphs is equivalent to the covering set problem.

A wide range of practical problems can be transformed to this form, from positioning retail stores to Sensor Networks and Mobile ad hoc Networks (MANETs). Different extensions of the the MDS like connected or weighted, are used to more precisely define the properties of the problem being solved [2]. Due to the fact that finding the optimal solution for MDS problem and its variations cannot be done in polynomial time a wide range of methods have been used to acquire near optimal solutions like greedy algorithm [3], Constant-factor approximation [4], energy function algorithm [5], collaborative cover heuristic [6] and Polynomial Kernels [7].

Ant colony optimization (ACO) is another metaheuristic for solving combinatorial problems, that was first used on the Traveling Salesman Problem by M. Dorigo. The MDSP has also been solved using population based algorithms like ant colony optimization (ACO) and genetic algorithms with ACO giving better results [8]. In this paper we extend the application of ACO to the MWDSP. A heuristic function that takes into account the weights of vertexes being covered is also introduced. We show advantages of using ACO instead of the greedy algorithms for MWDSP.

This paper is organized as follows. In the second section we explain heuristic functions that are used for this problem. In the next section we give details on applying ACO on MWDSP. In the fourth section we show our test and results.

## 2 Greedy Algorithms for MWDSP

The idea of the basic greedy algorithm as presented in [3] for the MDS is to add a new node to $W$ in each iteration, until $W$ forms a dominating set. We use the term that a node is covered if $j \in W$ or $j$ is adjacent to some node $i \in W$, and uncovered in the opposite case.

In iteration $n$, we put a new node $i \in V_n$ into $W$ that covers the maximum number of uncovered vertexes. $V_n$ is the set on vertexes that have not been selected in the first $n$ iterations. The algorithm is finished when all the nodes have been covered. This heuristic needs to be extended by taking into account the weights of vertexes.

To implement this algorithm the first step is to represent the problem in a way that dynamically calculates the heuristic function, and makes tracking

the covered vertexes simple. We use a similar approach as presented by Shyu for the Minimum Weighted Vertex Cover Problem [9] which is a weighted version of the heuristic used in [3]. In it a fully connected graph $G_c(V,E_c)$ is derived from $G$. In the article [9] they propose adding weights of 1 if the edge exists in $G$ or 0 if it does not exist in the original graph. We have adopted this approach which is illustrated by Fig. 1.

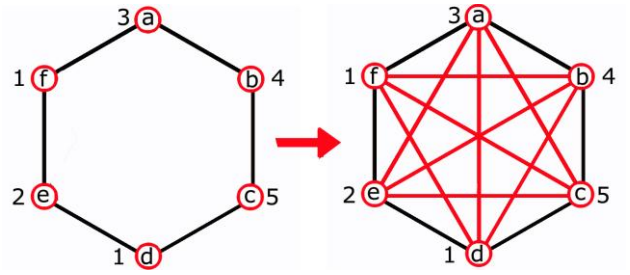


**Figure 1:** Transition from the original graph to a fully connected one. Black edges represent edges with a value one and red ones have a value zero.

As we mentioned before we also have to update this graph as we add new vertexes to our result set. This is done in the following way when we add vertex $a$ all edges in $G_c$ that are connected to $a$ and to its neighbors, are set to 0. This is illustrated by Fig. 2.

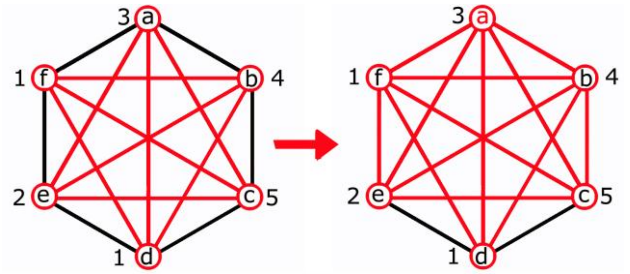


**Figure 2.** Correction of a graph when vertex $a$ is added to the solution set. Black edges represent edges with a value one and red ones have a value zero.

Now we can define $G_k$ $(V,E_{ck})$ as the state of the graph after $k$ vertexes have been added to the solution set, and a corresponding functions in Equation 1.

$$\psi_k(i,j) = Value(E_{ck}(i,j)) \qquad (1)$$

This update rule makes it possible to dynamically evaluate the preference of vertexes with function $\psi_k$. Now we can define a dynamic heuristic (Greedy1)

$$Cov = \sum_{(i,j)\in E_c} \psi_k(i,j)$$
$$\gamma_{jk} = \frac{1+Cov}{w(j)} \qquad (2)$$

The heuristic in Equation 2 states that vertexes with low weight and have a large number of unused edges are highly desirable. The addition of 1 to the sum is used to make a difference between vertexes that have no connections but are not yet covered. The heuristic $\gamma$ takes into account only the weight of the vertex that is currently being selected but not the weights of the vertexes that are being covered. We propose a new heuristic (Greedy2) that takes this into account

$$\eta_{jk} = \frac{Cov(1+\sum_{(i,j)\in E_c} w(i)\psi_k(i,j))}{w(j)} \qquad (3)$$

$\eta_{jk}$ states that we prefer vertexes that have low weight, a large number of connections and cover nodes with a high sum of weights.

For the two versions of the greedy algorithm to work, we have to be able to recognize if all the vertexes have been covered. This is done by adding set $Y_0 = V$ at the start of the algorithm and at each step $i$, when vertex $a$ is selected, we have

$$Y_i = \{v \mid (v,a) \in E \vee (a,v) \in E\} / Y_{i-1} \qquad (4)$$

The algorithm is finished when $Y_i = \emptyset$.

## 3 ACO for MWDSP

The use of ACO has proven to be effective on various types of problems from Economic Load Dispatch [10], Scheduling problems [11], even image processing [12]. The application of ACO on MWDSP differs in two main aspects to the original application on the Traveling Salesman Problem (TSP). In TSP our solution is an array of all the cities appearing in the problem or in other words the solution is the permutation of the set of vertexes. In the case of MWDSP the solution is a subset of the graph vertexes set, in which the order is unimportant. In the case of TSP the heuristic function being used is static in the sense that it represents the distance between vertexes and does not change during the generation of the solution. Contrary, for MWDSP, the heuristic function is the ratio between the weight and the sum of weights of neighboring vertexes, which is dynamic. This sum

changes as we add new vertexes to the solution set because more vertexes become covered.

These two differences affect the basic algorithm in two directions. First ants leave the pheromone on vertexes instead of edges. Second we dynamically update the graph, and with it, the heuristic function as shown in the previous section. Using the heuristic defined with $\eta_{jk}$ in Equation 3. we can setup the state transition rule for ants.

$$p_j^k = \begin{cases} 1 & , q > q_0 \ \& \ j = \arg\max_{i \in A_k} \tau_i \eta_{ik}^{\alpha} \\ 0 & , q > q_0 \ \& \ j \neq \arg\max_{i \in A_k} \tau_i \eta_{ik}^{\alpha} \\ \dfrac{\tau_j \eta_{jk}^{\alpha}}{\sum_{i \in Y_k} \tau_j \eta_{ik}^{\alpha}} & , q \leq q_0 \end{cases} \quad (5)$$

In Equation 5 $q_0$ is the standard parameter that appears in ACO that specifies the exploitation / exploration rate of individual ant searches. $q$ is a random variable that decides the type of selection on each step. $Y_k$ is a list of available vertexes. We point out that opposite to the TSP, transition rule does not depend on the last selected vertex and that is why we have $\tau_i$ instead of $\tau_{ij}$.

The next step is to define the global (when an ant finishes its path) and a local (when an ant chooses a new vertex) update rules. The role of the global update rule is to make paths that create better solutions to become more desirable, or in other words, it intensifies exploitation.

$$\Delta\tau_i = \frac{1}{\sum_{j \in W} w(j)} \quad , \forall i \in W \quad (6)$$

$$\tau_i = (1-p)\tau_i + \Delta\tau_i \quad (7)$$

Equations 6, 7 define the global update rule. In it $\Delta\tau_i$ is a quality measure of solution subset $W$ that contains vertex $i$, and with it we define an global update rule in Equation 7. This measure is inverse proportional to the weight of a solution. Parameter $p$ is used to set the influence of newly found solution on the pheromone trail.

The local update rule purpose is to shuffle solutions and to prevent all ants from using very strong vertexes. The idea is to make vertexes less desirable as more ants visit them. In this way, exploration is supported. The formula for the local update rule has the standard form

$$\tau_i = (1-\varphi)\tau_i + \varphi\tau_0 \quad (8)$$

For the value of $t_0$ we take the quality measure of the solution acquired with the greedy algorithm (Greedy2). Parameter φ is used to specify the strength of the local update rule.

# 4 Tests and Results

In this section we compare the results of applying the standard greedy algorithm for MWDSP (Greedy1), our improved version of this algorithm (Greedy2) and ACO optimization using the same heuristic as Greedy2. The program for our experiments was written in C#, using the framework from article [13]. We have created a plug-in for this system that implements ACO for MWDSP.

We used the following parameters for the ACO algorithm. Colonies consisted of 10 ants. The exploration rate was $q_0=0.1$, evaporation rates were $\varphi=0.1$ and $p=0.1$. For the influence factor of the heuristic we used $\alpha=4$. We implemented ACO in the MMAS variation. The initial best solution $V'$ was given by Greedy2 and the initial value of the pheromone trail is given by Equation 9.

$$\tau_0 = \frac{1}{n\sum_{j \in V'} w(j)} \quad (9)$$

In Equation 9, $n$ is the number of nodes in the $V'$. Each colony had 10 000 iterations.

We generated two types of random problem instances for our tests. In the first one weights for nodes where randomly selected for vertexes from the interval [20, 70]. In the second group the weights where dependent of the number of connections vertex $v$ had and it would be randomly selected from the $[1, e(v)^2]$. $e$ gives the number of connections for $v$. We used graphs from 50 to 1000 nodes with different number of randomly created edges but always making the graph connected. For all the tested vertex-node pairs, we created 10 different problem instances and we observed the average solution values for each of the three methods. We show our results in Tables 1, 2, 3, 4.

When comparing the two greedy algorithms, we first notice that the improved heuristic that takes into account the weights of covered vertexes gives almost uniformly better results, for both types of generated problem. The Greedy2 would give results that would be better up to 10%. The improvement was smallest for sparse graphs.

**Table 1**. Comparison of results for small and medium problems for Type1 problems

| Size | Greedy1 | Greedy2 | ACO |
|---|---|---|---|
| 50*50 | 610.3 | 609.9 | 539.8 |
| 50*100 | 509.5 | 472.7 | 391.9 |
| 50*250 | 262.6 | 260.2 | 195.3 |
| 50*500 | 157.7 | 155.5 | 112.8 |
| 50*750 | 114.1 | 99.8 | 69.0 |
| 50*1000 | 86.2 | 83.0 | 44.7 |
| | | | |
| 100*100 | 1232 | 1223.7 | 1087.2 |
| 100*250 | 864.4 | 819.7 | 698.7 |
| 100*500 | 564.1 | 554.3 | 442.8 |
| 100*750 | 448.2 | 413.5 | 313.7 |
| 100*1000 | 352.6 | 336.4 | 247.8 |
| 100*2000 | 195.3 | 210.6 | 125.9 |
| | | | |
| 150*250 | 1799.5 | 1758.6 | 1630.1 |
| 150*250 | 1548.0 | 1496.4 | 1317.7 |
| 150*500 | 1064.2 | 1051.8 | 899.9 |
| 150*750 | 870.0 | 840.3 | 674.4 |
| 150*1000 | 704.4 | 685.8 | 540.7 |
| 150*2000 | 415.2 | 366.3 | 293.1 |
| 150*3000 | 288.8 | 283.9 | 204.7 |
| | | | |
| 200*250 | 2329.2 | 2274.1 | 2039.2 |
| 200*500 | 1729.2 | 1707.8 | 1389.4 |
| 200*750 | 1349.4 | 1324.9 | 1096.2 |
| 200*1000 | 1124.6 | 1102.0 | 869.9 |
| 200*2000 | 703.5 | 665.3 | 524.1 |
| 200*3000 | 513.1 | 523.9 | 385.7 |

**Table 2**. Comparison of results for large problems for Type1 problems

| Size | Greedy1 | Greedy2 | ACO |
|---|---|---|---|
| 500*500 | 6046.2 | 5944.6 | 5476.3 |
| 500*1000 | 4785.8 | 4664.2 | 4069.8 |
| 500*2000 | 3248.0 | 3140.8 | 2627.5 |
| 500*5000 | 1712.0 | 1689.8 | 1398.5 |
| 500*10000 | 990.6 | 1006.1 | 825.7 |
| | | | |
| 800*1000 | 9160.3 | 8953.4 | 8098.9 |
| 800*2000 | 6729.8 | 6597.6 | 5739.9 |
| 800*5000 | 3833.4 | 3747.5 | 3116.5 |
| 800*10000 | 2325.2 | 2248.9 | 1923 |
| | | | |
| 1000*1000 | 12146.3 | 11987.7 | 10924.4 |
| 1000*5000 | 5595.4 | 5501.3 | 4662.7 |
| 1000*10000 | 3550.5 | 3414.1 | 2890.3 |
| 1000*15000 | 2562.6 | 2428.1 | 2164.3 |
| 1000*20000 | 2017.8 | 1918.1 | 1734.3 |

**Table 3**. Comparison of results for small and medium problems for Type2 problems

| Size | Greedy1 | Greedy2 | ACO |
|---|---|---|---|
| 50*50 | 73.7 | 72.2 | 62.3 |
| 50*100 | 137.7 | 126.6 | 98.4 |
| 50*250 | 383.3 | 362.8 | 202.4 |
| 50*500 | 872.7 | 750.6 | 312.9 |
| 50*750 | 1358.7 | 1227.2 | 386.3 |
| | | | |
| 100*100 | 150.6 | 144.2 | 126.5 |
| 100*250 | 327.5 | 299.4 | 236.6 |
| 100*500 | 794.3 | 725.9 | 404.8 |
| 100*750 | 1075.3 | 998 | 615.1 |
| 100*1000 | 1358.4 | 1243.9 | 697.3 |
| 100*2000 | 3398.7 | 3203.2 | 1193.9 |
| | | | |
| 150*250 | 223.4 | 214.2 | 190.1 |
| 150*250 | 323.6 | 318 | 253.9 |
| 150*500 | 652.9 | 609.4 | 443.2 |
| 150*750 | 1023.3 | 900.6 | 623.3 |
| 150*1000 | 1426.1 | 1265.2 | 825.3 |
| 150*2000 | 3233.8 | 2701.9 | 1436.4 |
| 150*3000 | 5041.6 | 4356.2 | 1751.9 |
| | | | |
| 200*250 | 348.9 | 335.5 | 293.2 |
| 200*500 | 618.7 | 589.1 | 456.5 |
| 200*750 | 913.3 | 867.3 | 657.9 |
| 200*1000 | 1342.2 | 1222.6 | 829.2 |
| 200*2000 | 2620.7 | 2433.6 | 1626 |
| 200*3000 | 4450.8 | 3702.0 | 2210.3 |

**Table 4**. Comparison of results for large problems for Type2 problems

| Size | Greedy1 | Greedy2 | ACO |
|---|---|---|---|
| 500*500 | 743.7 | 705.7 | 651.2 |
| 500*1000 | 1294.7 | 1198.1 | 1018.1 |
| 500*2000 | 2632.2 | 2378.1 | 1871.8 |
| 500*5000 | 6642.8 | 5714 | 4299.8 |
| 500*10000 | 14619.9 | 14163.3 | 8543.5 |
| | | | |
| 800*1000 | 1375.8 | 1325.5 | 1171.2 |
| 800*2000 | 2551 | 2359.3 | 1938.7 |
| 800*5000 | 6643.7 | 5957.5 | 4439.0 |
| 800*10000 | 14068.4 | 12443.1 | 8951.1 |
| | | | |
| 1000*1000 | 1479.0 | 1432.0 | 1289.3 |
| 1000*5000 | 6722.9 | 6082.9 | 4720.1 |
| 1000*10000 | 14352.8 | 13141.7 | 9407.7 |
| 1000*15000 | 23032.8 | 20476 | 14433.5 |
| 1000*20000 | 32295.5 | 28427.8 | 19172.6 |

ACO used as initial guess the result acquired by Greed2 so we shall compare its results to it. First we point out that ACO has improved results in all the tested cases. The improvement would vary for more and less dense graphs. In the case of graphs with lowest density, or in other words the average number of edges per vertex, the improvement would be the smallest 10-20%. The advantages of ACO would be increased with the increase of the density and in the cases when the average number of edges per vertex is 20 it would be from 20% to even 70%. This shows that ACO is very efficient on this problem, and that both greedy algorithms are not good for dense graphs.

# 5 Conclusion

In this paper we have shown an implementation of ACO for MWDSP. We have presented a new heuristic function for the greedy algorithm that takes into account the weights of vertexes being covered and shown that it is an improvement to the standard one. We used this heuristic function in our implementation of ACO. Tests for these methods have been done on a variety of graphs with different sizes, edge densities and weight generation algorithms. Our results show that our heuristic function improves the performance of the greedy algorithm. ACO proved to be very efficient on the MWDSP and greatly improved the quality of results especially in dense graphs.

In the future we wish to extend work to connected dominating set problems due to the fact that they are more closely related to MANETs than the non-connected version.

**Acknowledgment:** This research is supported by Project 144007, Ministry of Science, Republic of Serbia.

*References:*

[1] M. R. Garey, and D. S. Johnson, Computers and Intractability. A Guide to the Theory of NPCompleteness., New York-San Francisco: W. H. Freeman and Company, 1979.

[2] J. Blum, M. Ding, A. Thaeler et al., "Connected Dominating Set in Sensor Networks and MANETs," Handbook of Combinatorial Optimization, D. D.-Z. and P. P., eds., pp. 329 - 369, Kluwer: Academic Publishers, 2004.

[3] A. K. Parekh, "Analysis of a greedy heuristic for finding small dominating sets in graphs," Inf. Process. Lett., vol. 39, no. 5, pp. 237-240, 1991.

[4] C. Ambühl, T. Erlebach, M. Mihalák et al., "Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs." pp. 3-14.

[5] X. Xu, Z. Tang, W. Sun et al., "An Algorithm for the Minimum Dominating Set Problem Based on a New Energy Function," in SICE Annual Conference, Sapporo, Japan, 2004, pp. 924-926.

[6] R. Misra, and C. Mandal, "Minimum Connected Dominating Set Using a Collaborative Cover Heuristic for Ad Hoc Sensor Networks," IEEE Transactions on Parallel and Distributed Systems, vol. 21, pp. 292-302, 2010.

[7] S. Gutner, "Polynomial Kernels and Faster Algorithms for the Dominating Set Problem on Graphs with an Excluded Minor," Parameterized and Exact Computation: 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers, pp. 246-257: Springer-Verlag, 2009.

[8] C. K. Ho, Y. P. Singh, and H. T. Ewe, "An Enhanced Ant Colony Optimization Metaheuristic for the Minimum Dominating Set Problem," Applied Artificial Intelligence, vol. 20, pp. 881-903, 2006.

[9] S. S. Jian, Y. Peng-Yeng, and L. B. M.T., "An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem," Annals of Operations Research, vol. 131, pp. 283-304, 2004.

[10] A. Vlachos, "An Ant Colony Optimization (ACO) algorithm solution to Economic Load Dispatch (ELD) problem," WSEAS Transactions On Systems, vol. 5, no. 8, pp. 1763 - 1771, 2006.

[11] [11]F. Kolahan, M. Abachizadeh, and S. Soheili, "A comparison between Ant colony and Tabu search algorithms for job shop scheduling with sequence-dependent setups," WSEAS Transactions on Systems, vol. 12, pp. 2819-2824, 2006.

[12] N. E. Mastorakis, and X. Zhuang, "Image processing with the artificial swarm intelligence," WSEAS Transactions on Computers, vol. 4, no. 4, pp. 333-341, 2005.

[13] R. Jovanovic, M. Tuba, and D. Simian, "An Object-Oriented Framework with Corresponding Graphical User Interface for Developing Ant Colony Optimization Based Algorithms," WSEAS Transactions on Computers, vol. 7, no. 12, 2008