A Metaheuristic Approach to the Graceful Labeling Problem of Graphs

Houra Mahmoudzadeh[1] , Kourosh Eshghi [2]
[1,2] Industrial Engineering Department, Sharif University of Technology, Tehran, Iran.
[1] h_mahmoudzadeh@alum.sharif.edu, [2] eshghi@sharif.edu

Abstract: In this paper, an algorithm based on Ant Colony Optimization metaheuristic is proposed for finding solutions to the well-known graceful labeling problem of graphs. Despite the large number of papers published on the theory of this problem, there are few particular techniques introduced by researchers for gracefully labeling graphs. The proposed algorithm is applied to many classes of graphs, and the results obtained have proven satisfactory when compared to those of the existing methods in the literature.

Keywords: Graph Labeling, Graceful Graphs, Metaheuristic, Ant Colony Optimization.

1. GRACEFUL LABELING PROBLEM

Let G = (V, E) be an undirected graph without loops or double connections between vertices. A graceful labeling of G, with n vertices and m edges, is a one-to-one mapping f of the vertex set V into the set {0, 1, 2, ... , m}, so that if we assign an edge label $|f(x) - f(y)|$ to any edge (x,y), each edge receives a distinct positive integer label. A graph that can be gracefully labeled is called a graceful graph [11]. Examples of graceful graphs are shown in Figure 1.

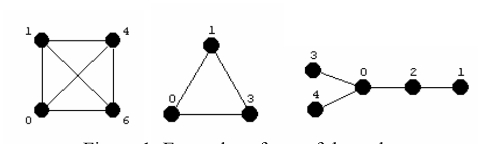


Figure 1: Examples of graceful graphs

Interest in graceful labeling began with a conjecture by Kötzig-Ringel and a paper by Rosa in 1966 [11]. Many variations of graph labeling have been introduced in recent years by researchers. Various classes of graphs have been proven mathematically to be graceful or non-graceful. A detailed survey of graph labeling problems and related results is presented in a survey by Gallian [8]. There is an unproved conjecture that all trees are graceful. Although, it is shown that trees with up to 27 vertices are graceful. All cycles C_n are graceful if and only if $n \equiv 0$ or 3 (mod 4). All wheels W_n, Helms H_n, and Crowns R_n are graceful. The complete graphs K_n are graceful if and only if $n \leq 4$. The necessary condition for a windmill $K_n^{(m)}$ $(n \geq 3)$ to be graceful is that $n \leq 5$; a windmill $K_n^{(m)}$ consists of m complete graphs K_n with one common vertex [8]. An example for each class of the graphs mentioned above and their graceful labelings are shown in Figure 2.

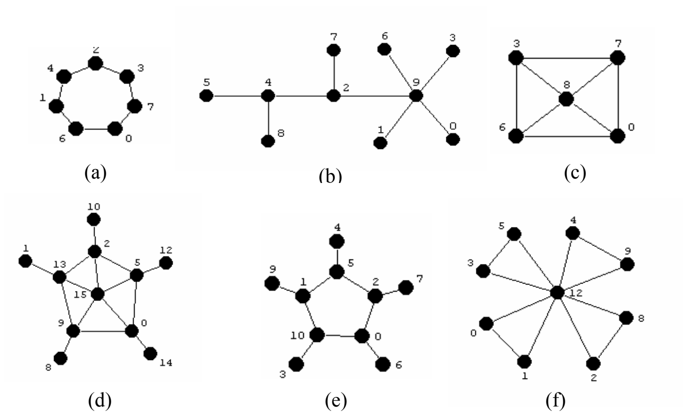


Figure 2: Examples for some classes of graceful graphs and their graceful labelings: (a) a cycle C_7, (b) a tree T_{10}, (c) a wheel W_4, (d) a helm H_5, (e) a crown R_5 and (f) a windmill $K_3^{(4)}$

The graceful labeling problem is to find out whether a given graph is graceful or not, and if it is, how to label the vertices. The process of gracefully labeling a graph is a very tedious and difficult task for many classes of graphs [7].

In this paper, the graceful labeling problem is encountered as an assignment-type problem with the aim of finding a feasible solution, and a metaheuristic approach based on ACO for gracefully labeling graphs is presented.

Since today many methods have been applied for proving gracefulness of different classes of graphs, but most of them don't apply a general method for finding the graceful labeling of the graphs to be studied. There are only two mathematical programming methods in the literature for finding graceful labeling of graphs. The first is a constraint programming approach [10], and the second is based on integer programming [7]. It is shown that the mathematical programming method outperforms the constraint programming method for graceful labeling [7].

The rest of this paper is organized as follows: In section 2, Ant Colony Optimization is briefly introduced. In section 3 a problem representation of graceful labeling problem in the framework of ACO is defined and the proposed metaheuristic algorithm is introduced. Section 4 shows the obtained results and compares them with those of the existing methods in the literature.

2. ANT COLONY OPTIMIZATION

Many problems of practical importance can be modeled as combinatorial optimization problems. It is known that the majority of these problems cannot be solved by a polynomial time algorithm. For this reason, heuristics have been invented to find solutions for these problems in a reasonable amount of time. Some of these heuristic algorithms are not restricted to specific problem types, but may be applied, with suitable modifications, to a broad class of optimization problems. Often, these "general-purpose" algorithms are called *metaheuristics* [5].

Metaheuristics are usually inspired by a natural process such as genetic algorithms or simulated annealing. Recently one of the most successful metaheuristics is Ant Colony Optimization (ACO), which was first introduced by Marco Dorigo [3]. Many ACO based algorithms have been proposed to solve different types of combinatorial optimization problems such as symmetric and asymmetric traveling salesman problems [3], the graph coloring problem [1], the knight's tour problem in graphs [9] and the adaptive routing in packet-switched communications networks [2].

An important insight of early research on ants' behavior was that in many ant species the visual perceptive faculty is very rudimentarily developed (there are even ant species which are completely blind) and that most communication among individuals, or between individuals and their environment, is based on the use of chemicals, called *pheromones*, produced by the ants. Particularly important for the social life of some ant species is the *trail pheromone*, a pheromone that individuals deposit while walking in search for food. While walking, ants deposit pheromone on the ground, and follow, probabilistically, pheromones previously deposited by other ants. By sensing pheromone trails, ants can follow the path to food discovered by other ants. This collective pheromone-laying / pheromone-following behavior whereby an ant is influenced by a chemical trail left by other ants was the inspiring source of Ant Colony Optimization [5].

In ACO, artificial ants make walks on a graph, and lay artificial pheromone trails on the vertices and/or edges of the graph. This artificial pheromone is accumulated at run-time through a learning mechanism that gives reward to good problem solutions. Artificial ants start from random states: next, each ant chooses probabilistically the new state to visit using a probabilistic function mainly based on the pheromone intensity. At the end of each iteration, the pheromone on the best solution is increased according to a learning rule. The rationale is that in this way the structure of "preferred sequences" emerges in the pheromone trail and future ants will use this information to generate new and better solutions [4].
Ants can be characterized as stochastic construction procedures which build solutions moving on the construction graph $G' = (C, L)$. The construction graph, G', is not always the original problem graph, specially in assignment-type problems like graph coloring. Ants do not move arbitrarily on G', but rather follow a construction policy which is a function of the problem constraints Ω.

In general, artificial ants try to build feasible solutions, but, if necessary, they can generate infeasible solutions. The constraints that must be satisfied necessarily are called *hard constraints*, and others that may be ignored when not possible to be satisfied are called *soft constraints* [5]. Components $c_i \in C$ and connections $l_{ij} \in L$ can have associated a pheromone trail π (π_i if associated to components, π_{ij} if associated to connections) encoding a long-term memory about the whole ant search process that is updated by the ants themselves, and a heuristic value η (η_i and η_{ij} , respectively) representing a priori information about the problem instance definition or run-time information provided by a source different from the ants. In many cases heuristic information are used to make the ants satisfy the problem constraints. These values are used by the ants' heuristic rule to make probabilistic decisions on how to move on the graph [5].

After the ants complete their solutions at each iteration, a procedure called *pheromone evaporation* occurs. This procedure is designed to avoid a too rapid convergence of the algorithm towards a sub-optimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas of the search space [4].

3. PROPOSED ALGORITHM

In this section first a representation for the graceful labeling problem is defined and next an ACO-algorithm for gracefully labeling graphs is proposed. The graceful labeling problem can be represented in the framework of ACO as it will appear in this section. The main features to be defined are the construction graph G', pheromone trails and their update rules, heuristic information and the probabilistic decision rule. Remember that unlike most combinatorial optimization problems, the goal in the graceful labeling problem is just to find a *feasible* solution among all possibilities.

3.1. Construction Graph

The task in assignment-type problems (for example graph coloring problem) is to assign a set of items to a given number of objects or resources subject to some constraints. A solution which satisfies the constraints is said to be a feasible solution [1].
In this paper, graceful labeling problem is seen as an assignment-type problem in which a set of numbers are assigned to the vertices of a graph. In the framework of ACO, the construction graph is defined to be a complete bipartite graph, $G' = (C, L)$, where C, the set of components, consists of two parts: the first part is V, the set of vertices of graph G, and the second is the set $\{0, 1, \dots , m\}$. The *best* solution is the one with the least number of repeated edge labels. This number shows the solution quality in each iteration. Depending on the solution quality, at each iteration, ants update pheromone trails on the connections L which fully connect the two parts of C. The pheromone update rule will be explained in section 3.2. The amount of pheromone on each connection (i, j), where $i \in V$ and $j \in \{0, 1, \dots, m\}$, shows the desirability of assigning label j to vertex

i. The connections which are part of a *good* solution previously made, receive more and more pheromone, and pheromones on bad solution connections evaporate. When iterations continue, the feasible solution will have the most desirability to be chosen by ants. As an example, the construction graph for the complete graph K_3 and an example of pheromone accumulation on connections corresponding to a feasible solution are shown in Figure 3.

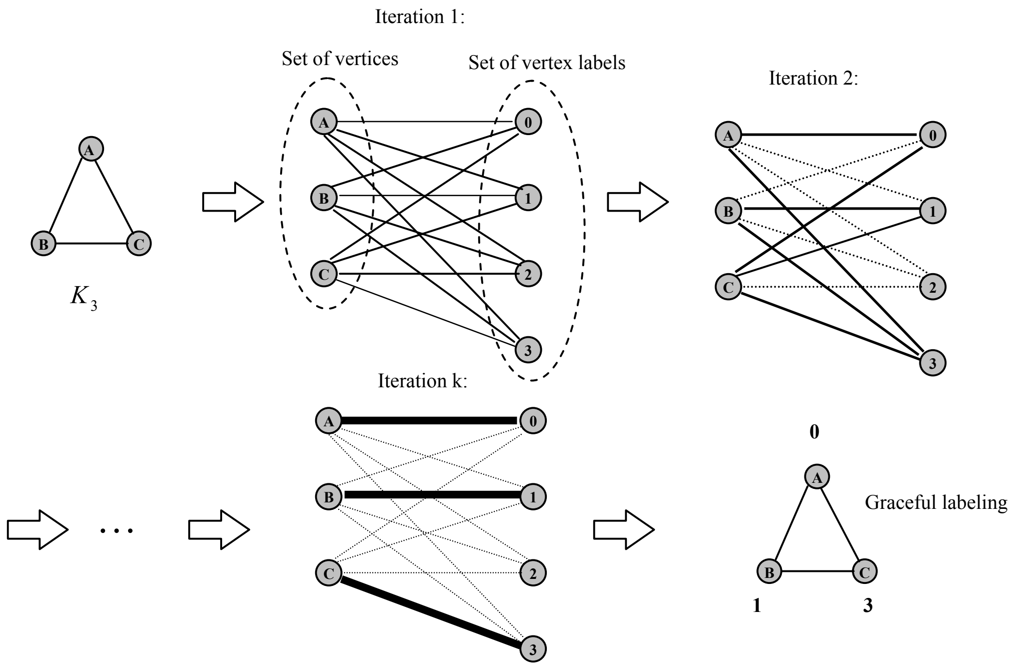


Figure 3: the construction graph for the complete graph K_3, and an example of pheromone accumulation on connections corresponding to a feasible solution.

3.2. Pheromones

Different types of pheromones, their update rules, the corresponding decision probabilities, and the pheromone evaporation procedure are defined in this section.

3.2.1. Different types of pheromones

Three types of artificial pheromones are defined and used by foraging ants to make probabilistic decisions for making solutions. Each pheromone $\pi_k(v, j)$ shows the desirability of assigning label j to vertex i. The difference among them is the definition of a desirable assignment and the corresponding update rule.

Pheromone type 1: The first pheromone ($\pi_1(v, j)$) shows the desirability of assigning label j to vertex i, considering the number of repeated edge labels produced in the neighborhood of vertex v by making such an assignment in past iterations. After a solution is completed, the artificial ant checks every vertex to see whether there are repeated edge labels produced in the edges adjacent to it. Whenever a repeated edge label is recognized, a part of pheromone 1 for the corresponding assignment is evaporated, otherwise some additional pheromone is deposited on the connection related to the assignment on the construction graph G'.

Pheromone type 2: By means of pheromone 2, the assignments with those edge labels that are less produced in total iterations, are more desirable. While deciding which label to assign to vertex v, the assignment in which an edge label in the neighborhood of vertex v is not yet produced, will be made with more probability by the artificial ant. Thus, the second pheromone ($\pi_2(v, j)$) is updated during each step of the solution construction.

Pheromone type 3: The third type of pheromones ($\pi_3(v, j)$) checks the role of assigning label j to vertex v in the total infeasibility of solutions in past iterations. After completing a solution, the artificial ant checks the partial role of the edge labels adjacent to every vertex in the total infeasibility; if all the edge labels adjacent to a vertex are not repeated in the whole graph, the amount of pheromone 3 for the corresponding assignment is increased, otherwise if labels in the neighborhood of vertex v are repeated somewhere in the graph, a part of pheromones on the edge connecting label j to vertex v on the construction graph is evaporated.

3.2.2. Pheromone Probability

The probability of assigning label j to vertex v according to each type of pheromones is calculated by equation (3.1).

$$pr_i(v,j) = \frac{\pi_i(v,j)}{\sum_{k=0}^{m} \pi_i(v,k)} \quad i = 1,2,3 \tag{3.1}$$

All these pheromone probabilities affect the final probability of assigning label j to vertex v at each step of labeling.

3.2.3. Pheromone Evaporation

We saw in the previous section that whenever a solution is desirable, the amount of pheromone on the corresponding edges on the construction graph increases, and otherwise the amount of pheromones decreases (i.e. we have both positive and negative pheromones in each step of solution construction). Therefore in the proposed algorithm the definition of *evaporation* should be different, because it should avoid accumulation of both positive and negative pheromones. Pheromone *evaporation* occurs after each iteration. The *evaporation* procedure doesn't always decrease the amount of pheromones; instead, whenever the amount of pheromone on a connection is more than 1 (the initial value of pheromone on all connections is set to be 1), evaporation occurs by a rate of σ, which means the amount of pheromone is decreased by being multiplied to (1-σ); on the other hand, if the amount of pheromone on a connection is less than 1, it's value will be divided by (1-σ) and so it will be increased. As we see, the *evaporation* procedure is modified to a avoid too rapid convergence to *good* but still *infeasible* solutions and also to implement a useful form of forgetting to avoid the complete elimination of the search areas which have not resulted in good solutions in past iterations.

3.3. Heuristic Information

Four types of heuristic information are defined to be used by artificial ants in their probabilistic decision making. These information are defined according to the special characteristics of the graceful labeling problem, and are calculated at each step of the algorithm. Heuristics help ants to avoid making infeasible assignments in the early steps of the algorithm. The two first heuristic information are used to satisfy the problem constraints, and the next two help the ants to make better decisions. First heuristic information includes a *hard constraint* which must be satisfied in all solutions made by ants, but the other three are *soft constraints* which may be ignored if they can't be satisfied with the hard constraint simultaneously.

Heuristic value type 1: The first heuristic information is defined to implement a hard constraint. This constraint says that all *vertex* labels must be distinct. Heuristic value 1 for assigning label j to vertex v ($\eta_1(v,j)$) is set to be zero if label j is already used for another vertex in this iteration. Otherwise, for all labels that are not yet used in this iteration, it is set equal to a positive constant value.

Heuristic value type 2: The constraint of *edge* labels to be distinct is considered in the second heuristic value. Heuristic value 2 for assigning label j to vertex v ($\eta_2(v,j)$) is set to be zero if the edge labels produced by such an assignment are used before in this iteration. This constraint is a soft constraint and at each step, if the remaining vertex labels (according to the hard constraint) can't satisfy this constraint, this heuristic value is ignored and ants can build infeasible solutions.

Heuristic value type 3: This heuristic value makes the ants prefer to produce the edge label 'm' if possible in any step. To explain the necessity of this heuristic value let us consider a graph $G = (V, E)$ with *m* edges and *n* vertices. For G to be graceful, all the edge labels '1', '2', … , 'm' must be assigned to the edges of G. Therefore, for a solution made by artificial ants in the metaheuristic algorithm, all the labels '1', '2', … , 'm' must be produced. The remarkable point is that the edge label '1' can be produced in *m* different ways: by assigning one of the following pairs of vertex labels to adjacent vertices:

(0, 1), (1, 2), … , (m-1, m)

Similarly, the edge label '2' can be produced in *m-1* different ways by assigning one of the following pairs of vertex labels to adjacent vertices:

(0, 2), (1, 3), … , (m-2, m)

Unlike the edge labels mentioned above, there is only one way for edge label 'm' to be produced; by assigning vertex labels '0' and 'm' to adjacent vertices. Therefore, heuristic value 3 is defined to help the ants make better solutions by necessarily assigning labels '0' and 'm' to adjacent vertices. If at a step of solution construction, edge label 'm' is not yet produced and can be produced at this step by a certain assignment, and by losing this opportunity, the edge label 'm' cannot be produced anywhere else in the graph, the ants prefer the assignment that produces it. Experimental results in section 4 show how efficient this heuristic value is in finding the graceful labeling of graphs.

Heuristic value type 4: Similar to heuristic value type 3, the fourth heuristic value is defined for edge label 'm-1' to be preferred. Edge label 'm-1' can be produced by assigning one of the two pairs (0, m-1) or (1, m) of vertex labels to adjacent vertices. Heuristic value 4 works exactly like heuristic value 3 except that it prefers making edge label 'm-1'. Experimental results show that in many iterations, the cause of infeasibility was that edge label 'm-1' could not be produced. By adding this heuristic value, the results were noticeably improved as it can be seen in section 4.

3.4. Probabilistic Decision Rule

The probability distribution function for choosing a label for a vertex depends on pheromone and heuristic information values for the corresponding assignment on the construction graph. Equation (3.2) shows the probability for assigning label 'l' to vertex 'v' in any step.

(3.2) $$\Pr(v,l) = \frac{\prod_{t=1}^{3} \left(pr_t(v,l)\right)^{b_t} \times \prod_{k=1}^{4} \left(\eta_k(v,l)\right)}{\sum_{j=0}^{m} \left(\prod_{t=1}^{3} \left(pr_t(v,l)\right)^{b_t} \times \prod_{k=1}^{4} \left(\eta_k(v,l)\right)\right)}$$

The weights b_1, b_2 and b_3 are considered for controlling the effect of each type of pheromones. For different classes of graphs, different values for these weights prove more efficiency as it will be shown in section 4.

3.5. Stopping Criteria

The algorithm stops when one of the following criteria are met:

1. A feasible solution is found. A solution with no repeated vertex or edge labels is a feasible solution. If such a solution is found, it will be the output of the algorithm.
2. The number of iterations reach a maximum limit. If no feasible solution is found during a certain number of iterations, the best solution found yet will be the output of the algorithm, mentioning that the result shows that no graceful labeling is found in a certain number of iterations.

3.6. Algorithmic Structure

The main steps of the proposed algorithm are the following:

Step 0. Start
Step 1. Input the adjacency matrix of graph G.
Step 2. Calculate the number of vertices (n) and edges (m) according to the adjacency matrix.
Step 3. Initialize pheromones and heuristic information according to the adjacency matrix.
Step 4. Choose next vertex (v) to be labeled.
Step 5. Calculate heuristic values $\eta_k(v,j)$ for every label j to be assigned to vertex v for k = 1, 2, 3, 4 and j = 0, 1, 2, … , m.
Step 6. Calculate pheromone probabilities $\Pr_t(v,j)$ for every label j to be assigned to vertex v for t = 1, 2, 3 and j = 0, 1, 2, … , m.
Step 7. Calculate total probability for assigning any label j to vertex v, according to equation (2).
Step 8. Choose a label probabilistically for vertex v.
Step 9. Save the partial solution found until this step.
Step 10. Check the solution to find out whether it is complete or not.
   a. If all vertices are labeled, go to step 11.
   b. Otherwise, if the partial solution isn't worse than the best solution ever found, go back to step 4, else go to step 11.
Step 11. Calculate the degree of infeasibility of the current solution and update the best solution found until now.
Step 12. Update pheromones according to the quality of the current solution.
Step 13. Check the stopping criteria:
   a. If any of the stopping criteria are met, go to step 14.
   b. Otherwise, go back to step 3.
Step 14. Output the feasible solution or the best solution found.
Step 15. End

4. COMPUTATIONAL RESULTS

In this section, first the effects of heuristic information and each type of pheromones are discussed, and next the results obtained for different classes of graphs will be presented and compared with those of the existing methods.

In this section, first the effects of heuristic information and each type of pheromones are discussed, and next the results obtained for different classes of graphs will be presented and compared with those of the existing methods.

4.1. The Effect of Heuristic Values

As mentioned in section 3, the first two heuristic values are defined for implementing the problem constraints, but the third and fourth heuristics help the algorithm find solutions faster. This latter effect is illustrated here by an example. The algorithm is applied for three random trees with 10 vertices (an example is shown in Figure 4), and the results for the algorithm with and without heuristics are compared. As shown in Figure 4, a tree with 10 vertices has 9 edges, so for a labeling to be graceful, the edges must receive the labels 1 through 9 distinctively. Without using heuristic values 3 and 4, in more than 80 percent of iterations the edge label '9' (which here corresponds to 'm' in section 3) is not used. Similarly, in more than 55 percent of iterations edge label '8' is not produced. After implementing heuristic values 3 and 4, both of these percentages decrease to about 40 percent. Although the procedure of calculating heuristic values 3 and 4 at each step of the algorithm takes running time, but it remarkably decreases the average number of iterations before finding the feasible solution. Table 4.1 summarizes these results for 30 times runs of the algorithm.

TABLE 4.1: USEFULNESS OF THE HEURISTIC VALUES 3 & 4

	Running time (seconds)	Number of iterations
Without heuristic values 3 and 4	1.72	23
With heuristic values 3 and 4	1.03	8

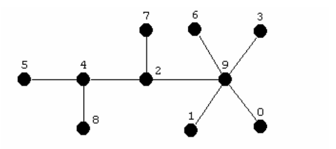


Figure 4: A random tree with 10 vertices

4.2. The Effect of Pheromones

The effect of each type of pheromones has been studied for different classes of graphs. We have seen by trial and error that for certain classes of graphs, a specific setting for pheromone weights (see section 4.4) gives the best results. One may obtain better results by modifying the weights for other special classes of graphs. We have studied three classes of graphs to check the effect of pheromones. These classes include cycle-related graphs, trees and combined complete graphs. The best parameter settings found for each class of graphs are summarized in Table 4.2.

TABLE 4.2: BEST PARAMETER SETTINGS FOUND FOR EACH CLASS OF GRAPHS

Graph class	Parameter settings
Cycle-related graphs	b_1=5; b_2=10; b_3=5
Trees	b_1=5; b_2=5; b_3=5
Combined complete graphs	b_1=2; b_2=2; b_3=10

These parameter settings have been derived according to a sensitivity analysis and have proven to be almost best settings found for the related classes of graphs. It has been seen during algorithm runs that by increasing any parameter more than the proposed settings, the algorithm performance becomes less effective. By increasing the parameters much more than appropriate settings, the algorithm fails to find the graceful labeling, because it continues searching within the initial ‘good’ but still ‘infeasible’ solutions.

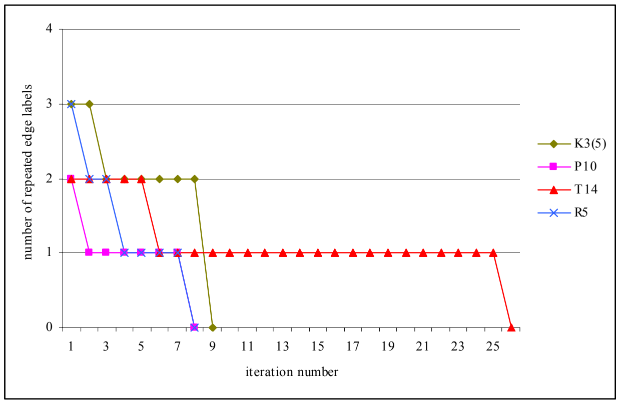
As you see in Table 4.2, for cycle-related graphs like simple cycles, wheels, helms, etc. by partially increasing the effect of the second pheromone, better results are obtained. It seems that this effect is because the second pheromone doesn’t insist on the absolute position of labels, but it rather provides information for searching the unexplored solution space. This ability, for cycle graphs which are completely symmetric and the absolute position of labels is not important, is very useful. On the other hand, for other classes like combined complete graphs, there exist some vertices that always get a certain label in a graceful labeling (like the common middle vertex). This causes the third pheromone to be useful more than the others for this class of graphs. The third pheromone pays extra attention to the absolute position of each label. However, for the class of trees, all tree types of pheromone seem to be equally efficient.

The proposed algorithm tries to decrease the number of repeated edge labels in the ongoing iterations. If, during an iteration, a solution made is worse than the best solution found yet, the algorithm doesn’t continue making the solution and starts a new iteration. By using this feature and also pheromones, the algorithm always continues to find better solutions. Figure 5 shows the performance of sample algorithm runs on random graphs of 8 different classes of graphs. The vertical axis corresponds to the number of repeated edge labels and the horizontal axis shows the number of iterations. Each bolded dot with coordination (x,y) represents one or more points at which in iteration x of algorithm, the number of repeated edge labels was equal to y. The lines show the iterative improvement in solutions during the algorithm run time.

Although the random graphs used had up to 40 edges, it can be seen that in the worst case, the starting solution has at most 8 repeated edge labels and the degree of infeasibility decreases with use of pheromones when the iterations continue. The maximum number of iterations in which the graceful solution was found for these 8 classes of graphs was 26 iterations, and the graceful labeling was found in all cases.

4.3. Experimental Results

The proposed algorithm program was written by MATLAB 7.0 software. The program receives the adjacency matrix of a given graph, and outputs the graceful labeling of its vertices in a row vector wherein the i-th component shows the label of the i-th vertex according to the adjacency matrix. The average time of 20 runs of the algorithm for instances from different classes of graphs are shown in Table 4.3. The graceful labeling is found in all mentioned cases.



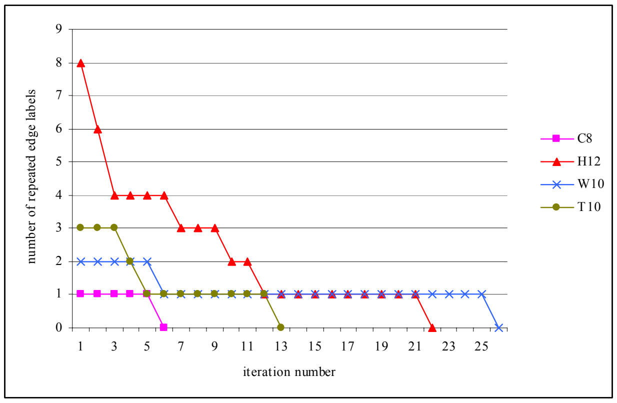


Figure 5: Decrease in the number of repeated edge labels when the proposed ACO algorithm is run on random graphs from 8 different classes of graphs

An advantage of the proposed graph labeling software is that it is very easy to use, and can be applied for gracefully labeling different types of graphs with entering only its adjacency matrix. As it can be seen from Table 4.3, the ant-based algorithm for graceful labeling provides solutions in a reasonable run time for different graph types.

The current exact method for graceful labeling of different types of graphs is a mathematical programming model which can be solved by a branch and bound method [7]. This method is very efficient for the classes of trees and cycles, but for more complex graphs like wheels, helms and windmills, etc. where the ratio of the number of edges to the number of vertices (m/n) increases, our algorithm performs much better than the exact method.

The results of our algorithm are compared with those of the mathematical programming model [7] in Table 4.4. The shorter running time in each case is indicated by bold letters. Both methods were performed on computers with same specifications.

Table 4.4 shows that the results of our metaheuristic algorithm are comparable with those of the exact method which is based on mathematical programming. In some classes of graphs, like wheels and helms, the comparison shows that our algorithm finds the solution very faster than the exact algorithm.

TABLE 4.3: AVERAGE RUN TIMES OF OUR ALGORITHM FOR DIFFERENT GRAPHS

Graph type	Graph name	Average running time (seconds)
Cycles	C_8	0.00
	C_{10}	4.26
	C_{15}	166.26
Complete graphs	K_3	0.00
	K_4	0.00
	$K_3^{(4)}$	0.25
	$K_3^{(5)}$	2.57
Wheels	W_4	0.10
	W_5	0.21
	W_8	3.50
	W_{10}	12.03
	W_{15}	139.37
	W_{20}	1057.34
Helms	H_5	0.75
	H_8	22.4
	H_{10}	37.71
	H_{12}	120.55
	H_{15}	897.36
Crowns	R_5	0.76
	R_8	3.14
	R_{10}	20.10
	R_{15}	235.54
Trees	T_5	0.00
	T_{10}	0.05
	T_{15}	120.64
	T_{20}	368.42
	T_{25}	1288.24

For the largest wheel tested in the mathematical programming method, our algorithm performs about 25 times faster than the exact method, and for the largest helm it is about 90 times faster. The largest wheel that is tested by the exact algorithm has 10 vertices and the largest helm has 15 vertices, therefore comparison was not possible for larger graphs on these classes.

TABLE 4.4: THE RESULTS OF OUR ALGORITHM COMPARED WITH THOSE OF THE MATHEMATICAL PROGRAMMING MODEL

Graph type	Graph name	Average run time (seconds)	
		Proposed ACO algorithm	Mathematical programming method
Wheels	W_{10}	**12.03**	55.50
	W_{15}	**139.37**	3358.11
Helms	H_8	**22.4**	1585.44
	H_{10}	**37.71**	3471.22
Cycles	C_{10}	4.26	**0.00**
	C_{15}	166.26	**0.65**
Trees	T_{20}	368.42	**149.12**
	T_{25}	**1288.24**	2898.14

VI. FUTURE RESEARCH DIRECTIONS

As it can be seen in Table 4.4, the only class of graphs where the mathematical programming method outperforms the proposed ACO algorithm, is the class of cycles which are completely symmetric graphs. Finding symmetry breaking methods in gracefully labeling graphs may be a suitable research area, and will increase the effectiveness of the proposed ant based algorithm for gracefully labeling symmetric graphs. Also one may propose an ACO algorithm for a certain class of graphs by defining additional heuristic information considering their special characteristics in order to increase algorithm efficiency. Another research area may be applying other metaheuristic methods like Genetic Algorithm, Simulated Annealing and Neural Networks, etc. to the graceful labeling problem and comparing the results with the ant based algorithm. Furthermore, ACO will probably be efficient when applied to other types of graph labeling problems like harmonious or magic labeling of graphs.

VII. CONCLUSION

The graceful labeling problem is one of the best known labeling methods in graphs. Despite the large number of papers published on this subject, there are few techniques for finding graceful labelings of a given graph. In this paper an ant based algorithm is proposed for gracefully labeling the vertices of a graph. The proposed algorithm was tested on a set of randomly generated graphs from different classes of graphs. The computational results showed that ACO metaheuristic was a powerful tool for finding solutions for the graceful labeling problem of graphs and outperforms the other existing methods in some certain classes of graphs.

REFERENCES

[1] D. Costa and A. Hertz , "Ants can colour graphs", Journal of the Operational Research Society, Vol. 48, pp. 295-305, 1997.

[2] G. Di Caro and M. Dorigo , "AntNet: Distributed stigmergetic control for communications networks", Journal of Artificial Intelligence Research, Vol. 9, pp. 317– 365, 1998.
[3] Dorigo M., Gambardella L. M., "Ant colonies for the traveling salesman problem", Biosystem, Vol. 43, pp. 73-81, 1997.
[4] Dorigo M. and Stutzle T., "Ant Colony Optimization: algorithms, applications and advances",. Metaheuristics Handbook, International Series in Operations Research and Management Science, Kluwer, 2001.
[5] Dorigo M. and Stützle T., "Ant Colony Optimization", MIT press, Cambridge, Massachusetts, 2004.
[6] Eshghi K., "Existence and construction of α-labeling of 2-regular graphs with three components", Ph.D. Thesis, University of Toronto, Ontario, Canada, 1997.
[7] Eshghi K. and Azimi P., "Application of mathematical programming in graceful labeling problem of graphs", Journal of Applied Mathematics Vol. 2004:1, pp. 1-8, 2004.
[8] Gallian J. A., "A Dynamic survey of graph labeling", Electronic Journal of Combinatorics Vol. 5, 2005.
[9] Hingston Ph. And Kendall G., "Ant colonies discover knight tour*s*", Springer-Verlag, Berlin Heidelberg, 2004.
[10] Redl T. A., "Graceful graphs and graceful labelings: two mathematical programming formulations and some other new results", Technical Report TR03-01, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, 2003.
[11] Rosa A. On certain valuation of the vertices of a graph, Theory of Graphs (International Symposium, Rome, July 1996) Gordon and Breach, New York and Dund Paris, 349-355, 1967.