APPLICATIONS OF MATHEMATICAL PROGRAMMING IN GRACEFUL LABELING OF GRAPHS

KOUROSH ESHGHI AND PARHAM AZIMI

Received 19 October 2003 and in revised form 31 December 2003

Graceful labeling is one of the best known labeling methods of graphs. Despite the large number of papers published on the subject of graph labeling, there are few particular techniques to be used by researchers to gracefully label graphs. In this paper, first a new approach based on the mathematical programming technique is presented to model the graceful labeling problem. Then a "branching method" is developed to solve the problem for special classes of graphs. Computational results show the efficiency of the proposed algorithm for different classes of graphs. One of the interesting results of our model is in the class of trees. The largest tree known to be graceful has at most 27 vertices but our model can easily solve the graceful labeling for trees with 40 vertices.

1. Introduction

Let $G = (V,E)$ be an undirected finite graph without loops or multiple edges. All parameters in this paper are positive integers. Terms and notations not defined in this paper follow that used in [1, 2].

A *graceful labeling* (or *β-labeling*) of a graph $G = (V,E)$ with n vertices and m edges is a one-to-one mapping Ψ of the vertex set $V(G)$ into the set $\{0,1,2,\ldots,m\}$ with this property: if we define, for any edge $e = (u,v) \in E(G)$, the value $\Omega(e) = |\Psi(u) - \Psi(v)|$, then Ω is a one-to-one mapping of the set $E(G)$ onto the set $\{1,2,\ldots,m\}$. A graph is called graceful if it has a graceful labeling. The concept of a graceful labeling has been introduced by Rosa [8] as a means of attacking the famous conjecture of Ringel that K_{2n+1} can be decomposed into $2n+1$ subgraphs that are all isomorphic to a given tree with n edges.

Rosa proved that if G is graceful and if all vertices of G are of even degrees, then $|E(G)| \equiv 0$ or $3 \pmod 4$. Although most graphs are not graceful, graphs that have some sort of regularity of structure are graceful [4]. Many variations of graceful labeling have been introduced in recent years by researchers. A detailed history of graph labeling problems and related results is presented by Gallian [3, 4]. All cycles C_n are graceful if and only

Journal of Applied Mathematics 2004:1 (2004) 1–8
2000 Mathematics Subject Classification: 05C78
URL: http://dx.doi.org/10.1155/S1110757X04310065

if $n \equiv 0$ or $3 (\mathrm{mod}\, 4)$. All snakes P_n, wheels W_n, helms H_n, crowns R_n, and complete bipartite graphs $K_{m,n}$ are graceful. The complete graphs K_n are graceful only if $n \leq 4$. It has been conjectured that all trees are graceful. Although this conjecture has been the focus of more that 200 papers, it is still an open problem. It has been shown that trees with at most 27 vertices are graceful.

Although more than 400 papers have been published on the subject of graph labeling, there are few particular techniques to be used by authors. The graceful labeling problem is to find out whether a given graph is graceful, and if it is graceful, how to label the vertices. The common approach in proving the gracefulness of special classes of graphs is to either provide formulas for gracefully labeling the given graph, or construct desired labeling from combining the famous classes of graceful graphs. Although Redl [7] has presented integer programming and constraint programming approaches for formulating the graceful labeling problem, he has mostly focused on three classes of graphs. Unfortunately, the process of gracefully labeling a particular graph G is a very tedious and difficult task for many classes of graphs. In this paper, a new approach based on the mathematical programming technique is presented to model and solve the graceful labeling problem for different classes of graphs.

2. Mathematical programming model of graceful labeling problem

In modeling the graceful labeling problem, some of our variables cannot take the same value and should be formulated by inequality constraints. In 1996, Hajian [5] presented an efficient method for dealing with inequality constraints. He used variables named "nonzero variables" to convert inequality constraints to equality constraints. For example, assume that we have the below constraints:

$$x_1 \neq x_2, \quad x_1, x_2 \geq 0. \tag{2.1}$$

By introducing a new variable w as a nonzero variable, we have

$$x_1 - x_2 - w = 0, \quad x_1, x_2 \geq 0,\ w \neq 0. \tag{2.2}$$

This method is more efficient than the traditional methods, such as using zero-one variables, to deal with inequality constraints, and we use it in our model.

Denote the vertices of the graph $G = (V, E)$ by $v_1, v_2, \ldots, v_n$, respectively. Now suppose that the decision variables of the model are defined as follows:

(i) x_j: the label of vertex v_j;

(ii) x_{ij}: the label of an edge (v_i, v_j) and a nonzero variable that connects vertices v_i and v_j, where $x_{ij} \neq 0$ implies that the labels of adjacent vertices v_i and v_j are distinct;

(iii) s_{ijkl}: a nonzero variable, where $s_{ijkl} \neq 0$ implies that the labels of edges (v_i, v_j) and (v_k, v_l) are not equal;

(iv) w_{ijkl}: a nonzero variable, where $w_{ijkl} \neq 0$ implies that the value of an edge label (v_i, v_j) is unequal to the negative value of an edge label (v_k, v_l);

(v) y_{ij}: a nonzero variable, where $y_{ij} \neq 0$ implies that the labels of nonadjacent vertices v_i and v_j are distinct.

The following model has a feasible solution if G is graceful.

Problem 2.1. (1) $x_i - x_j = x_{ij}$ for all i, j, such that $(v_i, v_j) \in E(G)$;
(2) $x_{ij} - x_{kl} = s_{ijkl}$ for all $i, j, k, l, (i, j) \neq (k, l)$, such that $(v_i, v_j), (v_k, v_l) \in E(G)$;
(3) $x_{ij} + x_{kl} = w_{ijkl}$ for all $i, j, k, l, (i, j) \neq (k, l)$, such that $(v_i, v_j), (v_k, v_l) \in E(G)$;
(4) $x_i - x_j = y_{ij}$ for all $i, j, i \neq j$ such that $v_i, v_j \in V(G)$, $(v_i, v_j) \notin E(G)$;
(5) $0 \leq x_i \leq m$, integer, for all i such that $v_i \in V(G)$;
(6) x_{ij}, s_{ijkl}, w_{ijkl}, and y_{ij} are nonzero variables.

In the above model, the first constraint is related to the definition of an edge label as the difference between the corresponding vertex labels. This constraint also causes the edge vertex labels to be distinct. Note that here an edge label is defined as a nonzero, but free in sign variable. Constraints (2) and (3) ensure that the absolute values of edge labels are not equal. Constraint (4) causes the labels of nonadjacent vertices to be distinct. Constraint (5) is related to this fact that the vertex labels are positive integers bounded between 0 and m. Constraints (1)–(5) guarantee that the edge labels are to be distinct and their absolute values generate the set $\{1, 2, \ldots, m\}$. The number of constraints in each equality sets (1)–(4) is m, $(m^2 - m)/2$, $(m^2 - m)/2$, and $(n^2 - n)/2 - m$, respectively. Thus, the total number of constraints (1)–(4) of Problem 2.1 is $(m^2 + 1/2n^2 - m - n/2)$. Furthermore, in Problem 2.1, the total number of variables is equal to the total number of constraints.

3. Branching method for solving graceful labeling problem

Branch-and-bound (B&B) algorithm is widely considered to be the most effective method for solving integer programming problems. In this section a special case of B&B algorithm is developed for solving Problem 2.1. First, the relaxation form of Problem 2.1 is defined as follows.

Problem 3.1. (1) $x_i - x_j = x_{ij}$ for all i, j, such that $(v_i, v_j) \in E(G)$;
(2) $x_{ij} - x_{kl} = s_{ijkl}$ for all $i, j, k, l, (i, j) \neq (k, l)$, such that $(v_i, v_j), (v_k, v_l) \in E(G)$;
(3) $x_{ij} + x_{kl} = w_{ijkl}$ for all $i, j, k, l, (i, j) \neq (k, l)$, such that $(v_i, v_j), (v_k, v_l) \in E(G)$;
(4) $x_i - x_j = y_{ij}$ for all $i, j, i \neq j$ such that $v_i, v_j \in V(G)$, $(v_i, v_j) \notin E(G)$;
(5) $0 \leq x_i \leq m$ for all i such that $v_i \in V(G)$;
(6) x_{ij}, s_{ijkl}, w_{ijkl}, and y_{ij} are free variables.

In the relaxation form of Problem 2.1, the hard constraints are relaxed to produce an easy subproblem. The hard constraints of Problem 2.1 are the integer constraints and the nonzero constraints. First, in Problem 2.1, the integrality constraint is removed and then the sign of the variables is changed from nonzero to free in sign to generate Problem 3.1. It is clear that Problem 3.1 is a linear model and it is much more easier to solve than Problem 2.1. In our branching method for solving Problem 3.1, in each node, the corresponding problem 3.1 is solved, and if the solution satisfies integrality and nonzero constraints, then a feasible solution of Problem 2.1 is found and the algorithm is terminated. If in each node, the corresponding problem 3.1 has no feasible solution, then the related node is fathomed. If the corresponding problem 3.1 has a feasible solution in the current node which is not a feasible solution of Problem 2.1, then at least one of the following cases occurs:

(1) noninteger values for integer variables;
(2) zero values for nonzero variables.

A node in a branching tree is called an active node if it has not been fathomed or separated yet. Active nodes are maintained in an active list. Each of the above cases (or both) can be the reason for being a node in the active list. Suppose that X^* is the optimal solution of the current Problem 3.1. Now define the following sets:

$$\begin{aligned} N_1 &= \{\forall x_{ij}, y_{ij} \in X^* \mid x_{ij} = y_{ij} = 0\}, \\ N_2 &= \{\forall s_{ijkl}, w_{ijkl} \in X^* \mid s_{ijkl} = y_{ijkl} = 0\}, \\ N_3 &= \{\forall x_i \in X^* \mid x_i \text{ has noninteger value in } X^*\}. \end{aligned} \tag{3.1}$$

There are two important steps that are the most critical to the performance of our algorithm as follows:

(1) *branching strategy*: selection of the next node from the active list to branch on,
(2) *separation rule*: selection of which variable in the selected node to separate on.

Let N be the total number of variables of the corresponding Problem 3.1 in the current node which are not feasible in constraints (5) or (6) of Problem 2.1. Denote the cardinality of set S by $|S|$. In fact, $N = |N_1| + |N_2| + |N_3|$, and N is a degree of infeasibility of the current node regarding Problem 2.1. If N is very small, then the corresponding solution is very close to a feasible solution for Problem 2.1. According to our experimental results in the implementation of the proposed algorithm, the "jumptracking strategy" is chosen as branching strategy. In this strategy, a node from the active list with the minimum value of N is chosen to branch on. If there is a tie, then a node with the minimum value for $|N_1| + |N_2|$ is selected. If there is still a tie, then a node with minimum value for $|N_1|$ is selected. Finally, if the tie is not broken, then a node from the remaining nodes is selected arbitrarily.

Suppose that according to our jumptracking strategy, the current active node j is selected. This node can have both types of variables causing infeasibility of node j for Problem 2.1. In the separation rule, one of the variables such as $x \in N_1$, N_2, N_3 in the selected node is chosen to branch on. If the selected variable $x \in N_3$, then the two new subproblems are generated from the selected node by using the integer part of x. Denote this strategy of separating current node by strategy A. If the selected variable $x \in N_1$ or N_2, then branch it into two different subproblems in which additional constraints $x \geq 1$ and $x \leq -1$ guarantee the nonzero values for x in the next nodes. Denote it by strategy B. In the experimental results section of this paper, these two methods are applied to more than 100 samples of different types of graphs and it is shown that the second method is much more effective than the first method. Furthermore, according to our test problems, separation on variable $x \in N_1$ is more effective than on variable $y \in N_2$. This fact shows that the potential effect of distinct edges is more powerful than that of distinct vertices in gracefully labeling a particular graph. Therefore, in separation rule of branching method, in the process of selection, the next variable, variable $x \in N_1$ has priority to variable $y \in N_2$ and in a similar way, $y \in N_2$ has priority to $z \in N_3$. Furthermore, when the branching method continues, many branches on the same variable will be generated in different parts of the branching tree. If a variable is chosen many times in different parts

of our branching tree, then probably the separation on this variable will not lead us to a feasible solution. Thus, in separation rule of our method, first variable $x \in X^*$ in the selected node is chosen according to our priority list, and if there is a tie, then a variable with the minimum number of selections in the other nodes of the branching tree has priority to the other variables. Finally, if there is still a tie, then it is broken arbitrarily.

3.1. The branching method for solving Problem 2.1

Step 1 (initializing). Suppose that a graph $G = (V, E)$ with n vertices and m edges is given and we want to know whether or not the graph G is graceful. Furthermore, if G is graceful, we want to know how to label the vertices. A node in a branching tree is active if its corresponding problem has not been either solved or subdivided yet. Let the set A denote the list of currently active nodes. Initially, set $A = \{$an active node corresponds to the original problem$\}$.

Step 2 (branching). If list A is empty, then stop. G is not graceful. Otherwise, select a node j from the active list A according to jumptracking strategy. If its corresponding Problem 3.1 has a feasible solution in which all the integer variables of problem 2.1 have integer values and all nonzero variables of Problem 2.1 have nonzero values, then $N = 0$, a feasible solution of Problem 2.1 is found, the graph G is graceful, and the algorithm is terminated. If the corresponding problem 3.1 has a feasible solution in the current node which is not a feasible solution for Problem 2.1, then go to Step 3.

Step 3 (selecting). Separate the current node into two subproblems according to separation rule described before. In each new node, solve its corresponding problem 3.1. Add the new subproblems to the active list if they have feasible solutions for Problem 3.1. Go to Step 2.

4. Computational results

In this section, our experimental results are summarized. The branching method presented in this paper is coded in the C language and the corresponding relaxation problems in Step 2 of the algorithm are run by OSL V. 3.0 software [6]. All computations were run on a Pentium IV 2500 MHz, 40000 MG H.D. with 256 MG RAM. In the tables shown in this section, the following abbreviations are used for columns: "Graph type" (the class of graphs under consideration), "n" (the number of the vertices of the graph), "m" (the number of the edges of the graph), "No. samples" (the number of samples of the given class of graph generated randomly), "No. var.," (the number of variables in the model), "Graceful?" (is the graph under consideration graceful?), and "Average time" (the average CPU time in seconds required to process the given class of graph). The notations and definitions of different classes of graphs used in this section follow that used in [1, 2]. It should be also noted that the number of constraints in our model is equal to the number of variables.

First, the effects of applying two methods for branching rule in implementation of branching method are reported in Table 4.1. The results show that strategy B is faster than A. In Table 4.2, different classes of randomly generated graphs are examined by the proposed B&B algorithm. According to the result of this table, a correct solution for each

Table 4.1. Comparison of two methods for branching rule in branching method.

Graph type	n	m	No. samples	No. var.	Graceful?	Average time (strategy A)	Average time (strategy B)
Caterpillars	15	14	30	302	Yes	0.01	0.05
Trees	20	19	32	552	Yes	146.02	149.12
Trees	30	29	32	1277	Yes	7656.06	4447.44
Snakes	27	26	20	1028	Yes	4659.00	2828.47
Snakes	28	27	20	1108	Yes	5265.55	3305.55

Table 4.2. The results of our algorithm for different classes of graphs.

Graph type		n	m	No. var.	Graceful?	Average time
Cycles C_n	C_8	8	8	92	Yes	0.00
	C_{10}	10	10	145	No	0.00
	C_{15}	15	15	330	Yes	0.65
	C_{20}	20	20	590	Yes	105.32
	C_{25}	25	25	925	No	3088.00
	C_{30}	30	30	1335	No	4828.21
Snakes P_n	P_5	5	4	27	Yes	0.00
	P_{10}	10	9	127	Yes	0.00
	P_{20}	20	19	552	Yes	91.88
	P_{25}	25	24	877	Yes	2898.11
	P_{30}	30	29	1277	Yes	4452.02
Complete graphs K_n	K_5	5	10	105	No	0.00
	K_8	8	28	792	No	2332.15
	K_{10}	10	45	2035	No	7085.21
Complete bipartite graph $K_{m,n}$	$K_{5,5}$	10	25	655	Yes	1301.02
Wheels W_n	W_7	8	14	218	Yes	0.00
	W_8	9	16	285	Yes	0.00
	W_{10}	11	20	446	Yes	55.50
	W_{15}	16	30	1006	Yes	3358.11
Helms H_n	H_5	11	15	276	Yes	0.00
	H_8	17	24	705	Yes	1585.44
	H_{10}	21	30	1101	Yes	3471.22
Crowns R_n	R_5	10	10	145	Yes	0.00
	R_8	16	16	376	Yes	3.51
	R_{10}	20	20	590	Yes	89.82
	R_{15}	30	30	1335	Yes	4730.00
Generalized Peterson graphs $P(n,k)$	$P(5,2)$	10	15	265	Yes	0.00
	$P(6,3)$	12	15	288	Yes	0.00
	$P(7,3)$	14	21	525	Yes	50.41
	$P(8,4)$	16	20	516	Yes	53.17
	$P(9,4)$	18	27	873	Yes	2063.52
	$P(10,5)$	20	25	810	Yes	2499.68
Product graphs $K_4 \times P_n$	$K_4 \times P_5$	16	36	1396	Yes	5172.14
	$K_4 \times P_{10}$	20	46	2280	Yes	5457.69

Table 4.3. The results of solving the model for different classes of trees.

Graph type	n	m	No. samples	No. var.	Graceful?	Average time
Trees	20	19	30	552	Yes	149.12
Trees	25	24	30	877	Yes	2898.14
Trees	28	27	30	1108	Yes	3827.11
Trees	30	29	30	1277	Yes	4447.25
Trees	35	34	30	1752	Yes	7625.69
Trees	40	39	30	2302	Yes	11321.54

of the test problems was found by our method in a reasonable amount of time. The number of test problems in each of the class of graphs in Table 4.2 is 10. Since the gracefulness of trees is a very important problem, in Table 4.3 the proposed model is applied to different types of large-scale trees generated randomly by "Naughty V. 2.2" program (http://cs.anu.edu.au/~bdm/). It is useful to note that the largest tree known to be graceful has at most 27 vertices [3, 4], but our model can easily solve the graceful labeling for trees with 40 vertices.

For comparison purposes, we have also compared our results with the results presented by Redl in [7]. He has developed two different approaches for graceful labeling problem. In the first approach, he developed an integer programming model and in the second one he applied a constraint programming technique. In the implementation of these two methods, three classes of graphs were tested at most: generalized Peterson graphs, product graphs of the form $K_4 \times P_n$, and double cones. According to his results, the constraint programming approach is more efficient than the integer programming approach. The largest graph tested by his constraint programming approach was the generalized Peterson graphs $P(10,5)$. The CPU time in seconds reported to solve $P(10,5)$ was 10481.40. It should be noted that Redl performed his algorithms on a SUN 220 workstation with two 450 MHz Ultra SPARC CPUs. Although the computer and the operating system reported in [7] are more powerful than our computer system, the computational time of our algorithm for the same test problem $P(10,5)$ is almost five times faster than the constraint programming approach presented in [7]. The average computational time of our algorithm for the class of generalized Peterson graphs is almost 25% faster than the proposed approach in [7]. Moreover, as it can be seen from Table 4.2, our algorithm can be easily applied to different classes of large graphs and provides accurate and efficient results. The results show that mathematical programming is a very powerful technique to solve the graceful labeling problem.

5. Conclusion

The graceful labeling problem is one of the most popular problems in the world of graph theory and discrete mathematics. Despite the large number of papers in this field of graph theory, there are no general techniques for labeling different classes of graphs. A common approach in graph labeling is to provide formulas for gracefully labeling the given graph. In this paper, first we presented a new approach for modeling the graceful labeling problem as a linear programming model. The main goal of this model is to determine how to

label the vertices of different classes of graceful graphs. Then a branching strategy was developed to solve the model. The algorithm has been extensively tested on a set of different classes of randomly generated graphs. The computational results show that the proposed approach can be a very effective tool for finding a feasible solution for the graceful labeling problem. Moreover, our algorithm does not depend on a particular class of graphs and can be easily applied to different types of graphs.

Acknowledgment

We are grateful to the anonymous referees for their constructive comments that improved the quality of this paper.

References

[1] J. Bosák, *Decompositions of Graphs*, Mathematics and Its Applications, vol. 47, Kluwer Academic Publishers Group, Dordrecht, 1990.

[2] K. Eshghi, *Construction of α-labeling of 2-regular graphs with three components*, Ph.D. thesis, University of Toronto, Ontario, Canada, 1997.

[3] J. A. Gallian, *A guide to the graph labeling zoo*, Discrete Appl. Math. **49** (1994), no. 1–3, 213–229.

[4] ———, *A dynamic survey of graph labeling*, Electron. J. Combin. **5** (1998), no. 1, Dynamic Survey 6, 43 pp.

[5] M. T. Hajian, *Dis-equality Constraints in Linear/Integer Programming*, Imperial College, London, 1996.

[6] S. A. OPL 3.0 Reference Manual ILOG, 2000.

[7] T. A. Redl, *Graceful graphs and graceful labelings: two mathematical programming formulations and some other new results*, Tech. Report TR03-01, CAAM Department, Rice University, Texas, 2003.

[8] A. Rosa, *On certain valuations of the vertices of a graph*, Theory of Graphs (Internat. Sympos., Rome, 1966), Gordon and Breach, New York, 1967, pp. 349–355.

Kourosh Eshghi: Department of Industrial Engineering, Sharif University of Technology, Tehran 11365-9414, Iran

E-mail address: eshghi@sharif.edu

Parham Azimi: Department of Industrial Engineering, Sharif University of Technology, Tehran 11365-9414, Iran

E-mail address: p.azimi@ikco.com