Magic Labelings on Cycles and Wheels

Andrew Baker and Joe Sawada

University of Guelph, Guelph, Ontario, Canada, N1G 2W1
{abaker04, jsawada}@uoguelph.ca

Abstract. We present efficient algorithms to generate all edge-magic and vertex-magic total labelings on cycles, and all vertex-magic total labelings on wheels. Using these algorithms, we extend the enumeration of the total labelings on these classes of graphs.

1 Introduction

Consider a wireless network in which every device must be able to connect to a subset of the other devices in the network using a unique channel to prevent collisions. One way to create such a channel assignment is to give numeric labels to the devices and channels in such a way that the labels of two devices and the communication line between them sum to a consistent value across every pair of devices in the network. In this case, knowing the labels of the two communicating devices gives the identification number of the communication line between them [1].

This solution is an example of an edge-magic total labeling (EMTL). EMTLs are one application of the "magic" concept of magic squares to graphs. Given a simple undirected graph $G = (V, E)$, let λ be a mapping from the numbers 1 through $|V| + |E|$ to the elements (vertices and edges) of G such that each element has a unique label. An *edge-magic total labeling* is a labeling λ in which the weight of each edge is the same. The weight of an edge is obtained by the sum of the label of the edge and the labels of its two endpoints and denoted by $w(e)$. If the weight is the same for every edge, it is termed the *magic constant* of the labeling, and is given by h. For an example of an EMTL with $h = 20$, see Fig 1(a).

A *vertex-magic total labelling* (VMTL) is a labeling λ in which the weight $w(v)$ of each vertex is the same. The weight of a vertex is obtained by adding the sum of the labels of the incident edges to the label of the vertex itself. If the weight is the same for every vertex in the graph, it is called the magic constant and is given by k. For an example of an VMTL with $k = 20$, see Fig 1(b).

A *totally magic labeling* is a labeling λ which is simultaneously both a vertex-magic total labeling and an edge-magic total labeling. The magic constants h and k are not necessarily equal. The class of totally magic graphs (those which admit a totally magic labeling) is much more restricted than the edge-magic or vertex-magic graphs. Figure 2 gives an example of a totally magic labeling on the cycle C_3. The only known connected totally magic graphs are K_1, K_3, and P_3. There

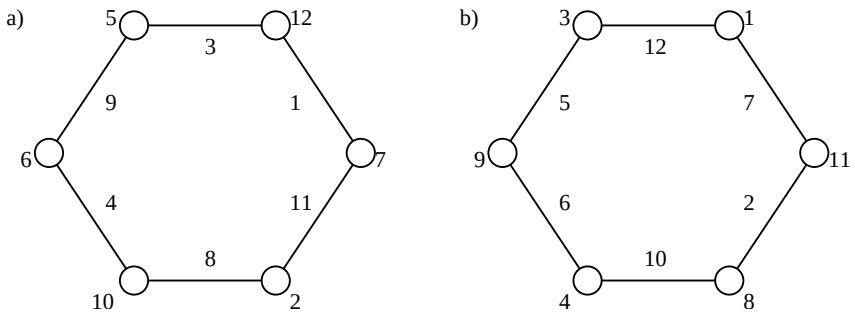

Fig. 1. Two C_6 graphs with corresponding edge-magic and vertex-magic total labelings. a) gives an edge-magic total labeling, and b) gives a vertex-magic total labeling.

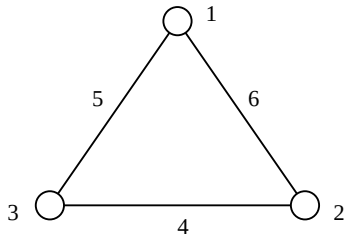

Fig. 2. The graph C_3 with a totally magic labeling. In this case, $h = 9$ (the edge-magic constant), and $k = 12$ (the vertex-magic constant).

are however an infinite number of disconnected totally magic graphs, as any graph consisting of a union of $2n+1$ ($n \geq 0$, $n \in \mathbb{Z}$) disjoint triangles is a totally magic graph [1]. There are additional types of magic labelings described beyond EMTLs, VMTLs and totally magic labelings. For a more complete treatment, see Gallian's dynamic survey [2].

Depending on which labels are assigned to vertices and which to edges, it is possible to achieve labelings with different magic constants on the same graph. A lower bound for a VMTL is obtained by applying the largest $|V|$ labels to the vertices, while an upper bound is found by applying the smallest $|V|$ labels to the vertices. Summing the weights of every vertex in a VMTL gives us $\sum_{v \in V} w(v) = |V|k$. Every vertex label contributes to one weight (the weight of that vertex) while every edge label contributes to two weights (the weights of its two end points). Thus $|V|k = \sum_{v \in V} \lambda(v) + 2\sum_{e \in E} \lambda(e)$. By applying either the $|V|$ smallest or largest labels to the vertices, we can obtain the inequality

$$\frac{13n^2 + 11n + 2}{2(n+1)} \leq k \leq \frac{17n^2 + 15n + 2}{2(n+1)}$$

which gives basic limits on the magic constant of a graph without taking into account the structure of the graph [3]. Once the structure of the graph is taken

into account, additional limits may be found. The set of integers which are delimited by these upper and lower bounds is the *feasible range*. The values which are the magic constant for some VMTL of a graph form the graph's *spectrum*. Therefore the spectrum is a subset of the feasible range.

In this paper we focus on finding all non-isomorphic VMTLs for cycles and wheels. Section 2 presents previous results with respect to vertex-magic total labelings on cycles and wheels. Sections 3 and 4 detail the enumeration algorithms and results for cycles and wheels respectively. Open problems for further research are presented in Section 5.

2 Background

Throughout this paper, we focus primarily on two classes of graph, the cycles and the wheels. The cycle C_n is given by the vertex set $v_1, v_2, \ldots, v_n \in V(G)$, and edge set $e_i \in E(G)$ where for $1 \leq i < n$, $e_i = \{v_i, v_{i+1}\}$ and $e_n = \{v_1, v_n\}$. Cycles are regular graphs (graphs in which every vertex has the same degree) as every vertex has degree 2. The wheels W_n consist of a cycle C_n together with an additional dominating vertex. A *dominating vertex* is a vertex which is adjacent to every other vertex in the graph. Figure 3 shows a sample wheel graph (W_6) and illustrates the naming scheme we will use while discussing parts of a wheel. Except for W_3, the wheels are not regular graphs.

2.1 Cycles

Every vertex-magic total labeling on a cycle (and indeed any regular graph) has a mirrored dual labeling. This property allows us, given an original labeling λ on graph G, to obtain the dual labeling λ' given by $\lambda'(v) = |V| + |E| + 1 - \lambda(v)$ for all vertices $v \in V(G)$ and $\lambda'(e) = |V| + |E| + 1 - \lambda(e)$ for all edges $e \in E(G)$. The resulting magic constant k' is given by $k' = 6n + 3 - k$ for cycles [4]. Consequently, the distribution of VMTLs by magic constant is symmetrical over the feasible range, and the presence of a VMTL achieving a magic constant in the upper half of the feasible range may be inferred by the presence of the dual labeling achieving the corresponding magic constant in the lower half of the feasible range.

Cycles also have a one-to-one correspondence between their edge- and vertex-magic total labelings. To obtain an EMTL λ_e from a vertex-magic total labeling λ_v, set $\lambda_e(v_i) = \lambda_v(e_i)$ and $\lambda_e(e_i) = \lambda_v(v_{(i+1) \bmod |V|})$ [4]. Figure 1 shows this correspondence graphically. Due to this relationship with EMTLs (which were developed earlier than VMTLs), previous work has been done to enumerate the edge-magic (and therefore also the vertex-magic) total labelings on cycles. The cycles C_3 through C_{10} were completely enumerated by Godbold and Slater [5]. We confirm these calculations, and also count the number of VMTLs/EMTLs on the cycles C_{11} through C_{18}.

Godbold and Slater show that a VMTL exists for every feasible magic constant for C_n when $n > 4$ [5]. Our enumeration breaks down the results for cycles by magic constant.

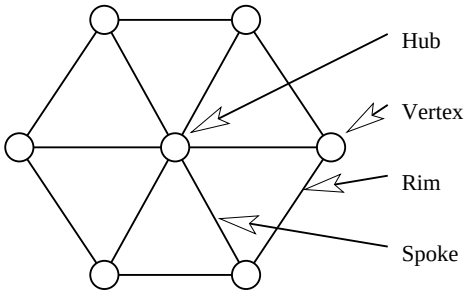

Fig. 3. The wheel graph W_6 demonstrating the naming convention we adopt for the elements of a wheel.

2.2 Wheels

As the wheel W_n consists of a cycle C_n together with a dominating hub vertex, W_n has $n+1$ vertices and $2n$ edges. The vertices v_1 through v_n refer to the vertices of the cycle, with the rim edges r_1 through r_n corresponding to the cycle edges e_1 through e_n. The spoke edges are those which connect the hub to a cycle vertex, and are given by $s_i = \{hub, v_i\}$ for $1 \leq i \leq n$. We demonstrate this naming scheme graphically in Figure 3.

A general conjecture on VMTLs is that having vertices in a graph which differ widely with regard to their degrees prevents that graph from having a vertex-magic total labeling. This holds for wheels, which have a high-degree hub, as shown by MacDougall, Miller and Wallis in [3].

MacDougall *et al.* give two different methods of computing a feasible range for wheel graphs, and the true feasible range is given by the most restrictive maximum and minimum values. In addition to the bounds on the feasible range given earlier, the feasible range on wheels can been further bounded from below by $k \geq \frac{(n+1)(n+2)}{2}$ and above by $k \leq 7n+6$ once you take the structure of the wheel into account. For the wheels W_n with $n > 11$, the minimum magic constant is larger than the maximum magic constant, so no VMTL can exist. MacDougall *et al.* also enumerate the VMTLs on wheels for W_3 (which is also the complete graph K_4), W_4, and W_5 [3]. We extend these results, counting W_6 through W_{10}.

3 Cycle Algorithm

A naïve method to generate all vertex-magic total labelings for a graph is to simply try all $(|V| + |E|)!$ permutations of the mapping of the labels onto the elements of the graph, and check to see if each result is a VMTL. Not only does this rapidly become infeasible on its own, as the size of the cycle increases it will also allow isomorphic copies of the same labeling to be generated independently. As such, every successfully generated VMTL must be compared to every other previously generated VMTL in order to remove duplicate copies.

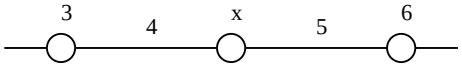

Fig. 4. A partial labeling of a piece of a graph with determined label x.

Our general approach is to apply vertex and edge labels working iteratively around the cycle. In the cycle VMTL generation algorithm, we remove cases of rotational symmetry by assigning the smallest vertex label to v_1, and then handle the reflective symmetry by making sure that $\lambda(e_1) < \lambda(e_n)$. Since v_1 must receive the smallest vertex label, it cannot be larger than $n + 1$ or there will be insufficient labels to label the remaining vertices.

Since we are interested in calculating the number of VMTLs for each magic constant, the algorithm we develop takes both n (the size of the cycle) and k (the magic constant) as input.

We say that a label is a *determined label* if it contributes its value to the weight of a vertex for which every other contributing label is known. Assuming we know the magic constant we are trying to reach, there is only one possible value for the determined label. There are three conditions on a determined label $\lambda(x)$ which allow us to terminate the recursion tree at this node and backtrack. These conditions are:

1. $\lambda(x) < 1$,
2. $\lambda(x) > |V| + |E|$, and
3. $\lambda(x)$ has already been used in this labeling.

Figure 4 gives a partial labeling and illustrates a determined label. In this example, if the desired magic constant is 20, then x must be 11. However, if the desired magic constant is 15, then x would have to be 6. Since 6 has already been used in this labeling, it would not be a valid partial labeling for $k = 15$. If in a cycle we have magic constant k, then $\lambda(e_2) = k - \lambda(v_2) - \lambda(e_1)$. Then, once we know $\lambda(v_3)$ and $\lambda(e_2)$, we are able to determine $\lambda(e_3)$.

Our algorithm aims to obtain determined labels as quickly as possible. If a given label being applied in the algorithm is determined and the partial labeling is infeasible, then the entire computation subtree rooted at that partial labeling can immediately be excluded. Even in the worst-case scenario, where none of the determined labels eliminates a partial labeling, the use of a determined label reduces the branching factor at a position in the computation tree from $1 \leq i \leq |V| + |E|$ to 1.

Before the algorithm itself is called, the global variables n and k are set with the size of the cycle and desired magic constant respectively, and the available list is initialized to every label being currently available. The actual algorithm begins with an initialization phase (by a call to **initializeCycle**()) which sets the labels of a vertex and two edges (v_1, e_1, and e_n). The initialize function then calls **extendCycle(2)** which recursively labels the remaining vertices and edges. Execution completes when there is only one vertex (v_n) remaining without

```
function initializeCycle ()
    for each available label i where i ≤ n + 1 do
        λ(v_1) := i
        avail[i] := false
        for each available label j do
            λ(e_1) := j
            avail[j] := false
            λ(e_n) := k − λ(v_1) − λ(e_1)
            if λ(e_1) < λ(e_n) ≤ 2n and avail[λ(e_n)] then
                avail[λ(e_n)] := false
                extendCycle (2)
                avail[λ(e_n)] := true
            avail[j] := true
        avail[i] := true
```

Fig. 5. Pseudocode for the initialization function for cycles. Global variables n and k are set to the desired values before the initializeCycle function is called.

a label. A linked list of unused labels is maintained at all times. This way, the more complete the partial labeling becomes, the fewer potential labels must be considered for each non-determined element.

The initialization function starts the labeling by attempting every possible label for vertex v_1 and edge e_1. We require that $\lambda(v_1)$ be the minimal vertex label in order to remove rotational symmetry. The maximum possible label for v_1 is $n+1$ due to the fact that since v_1 receives the minimum vertex label, we must retain $n-1$ labels greater than $\lambda(v_1)$ for the other vertices. This determines the label for edge e_n. In order to remove reflective symmetry, we require $\lambda(e_n) > \lambda(e_1)$. The initialization function then calls the extend function with parameter 2. The pseudocode for the initialization function can be found in Figure 5.

The extend method takes a single parameter - the position (t) in the cycle which is to be generated. A single loop applies, in turn, every unused label greater than $\lambda(v_1)$ to vertex v_t. Applying a label to v_t determines the label for e_t. The extend method then calls itself with parameter $t+1$. The recursion terminates when $t = n$. At this point there is only one unlabeled element, v_n, which is obviously determined. If the single remaining label is the required label, then the VMTL is successfully completed and the **Print()** method is called. **Print()** is a generic function which can be used to perform any operation on the completed VMTL. In the case of enumeration, a count of the number of VMTLs is incremented. Figure 6 gives the pseudo-code for the extend function.

In order to obtain results more quickly, the algorithm is parallelized to run on multiple different processors. Each process is given an integer value as a command-line argument which acts as a static value for the first element to be assigned a label. Instead of iterating through all available values, the algorithm simply uses the supplied label. As the runtime increases for larger graphs, the

```
function extendCycle (t)
    if t = n then
        λ(v_n) := k − λ(e_n) − λ(e_{n−1})
        if λ(v_1) < λ(v_n) ≤ 2n and avail[λ(v_n)] then
            Print ()
    else
        for each available label i where i > λ(v_1) do
            λ(v_t) := i
            avail[i] := false
            λ(e_t) := k − λ(v_t) − λ(e_{t−1})
            if 0 < λ(e_t) ≤ 2n and avail[λ(e_t)] then
                avail[λ(e_t)] := false
                extendCycle (t + 1)
                avail[λ(e_t)] := true
            avail[i] := true
```

Fig. 6. Pseudocode for the extend function for cycles. Global variables n and k are set to the desired values before the extendCycle function is called.

Table 1. The total number of non-isomorphic VMTLs for cycle graphs C_n ($3 \leq n \leq 18$).

C_n							
n	Unique VMTLs	n	Unique VMTLs	n	Unique VMTLs	n	Unique VMTLs
3	4	7	118	11	36128	15	74931690
4	6	8	282	12	206848	16	613296028
5	6	9	1540	13	1439500	17	5263250382
6	20	10	7092	14	10066876	18	47965088850

problem is distributed to more processors by supplying two seed values which determine the first two elements to receive labels.

3.1 Results

Table 1 gives the total number of EMTLs/VMTLs on the cycles C_3 through C_{18}, of which C_{11} through C_{18} had not previously been enumerated. Table 2 give the number of unique labelings broken down by magic constant.

4 Wheel Algorithm

Wheel graphs have a clear relationship to the cycles so the algorithm for generating all unique VMTLs on wheel W_n bears a similarity to the algorithm for cycle C_n. However, the extra vertex and additional n edges complicate the process.

Table 2. The number of unique VMTLs for cycle graphs C_3 through C_{10} broken down by magic constant (k). (Note that the duals have not been included.)

C_3		C_4		C_5		C_6	
k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs
9	1	12	1	14	1	17	3
10	1	13	2	15	0	18	1
				16	2	19	6

C_7		C_8		C_9		C_{10}	
k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs
19	9	22	10	24	31	27	125
20	10	23	19	25	43	28	236
21	11	24	57	26	125	29	698
22	29	25	55	27	264	30	1138
				28	307	31	1349

C_{11}		C_{12}		C_{13}		C_{14}	
k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs
29	308	32	1602	34	3809	37	32077
30	711	33	4111	35	10967	38	91866
31	1781	34	10834	36	33951	39	299525
32	3371	35	19183	37	79234	40	576701
33	4945	36	30877	38	139499	41	977354
34	6948	37	36817	39	202253	42	1427929
				40	250037	43	1627986

C_{15}		C_{16}		C_{17}		C_{18}	
k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs
39	63995	42	884789	44	1152784	47	26677502
40	284590	43	2706053	45	8660408	48	104169715
41	889063	44	8685625	46	30280605	49	351608789
42	2332807	45	20266824	47	86881643	50	859974262
43	4402572	46	37574150	48	187828262	51	1815449072
44	7339913	47	59829497	49	336981439	52	3082588134
45	10395599	48	83018416	50	511013242	53	4648495519
46	11757306	49	93682660	51	683131331	54	6154283390
				52	785695477	55	6939298042

As with the cycle algorithm, our wheel VMTL generation algorithm applies vertex and edge labels working iteratively around the edge of the cycle portion of the wheel. We remove rotational symmetry by assigning the smallest spoke label to s_1. As with cycles, reflective symmetry is removed by ensuring that $\lambda(r_1) < \lambda(r_n)$. We use the s_1 instead of the v_1 to remove rotational symmetry for the wheel in order to trim the computation tree of partial labelings which will result in a hub with excessive weight more efficiently. Since s_1 must receive the smallest spoke label, it cannot be larger than $2n + 2$ or there will be insufficient labels to label the remaining spokes.

Determined labels continue to be an asset to remove subtrees of the computation tree. In this case, we require three labels in order to determine a fourth. For example, $\lambda(r_2) = k - \lambda(s_2) - \lambda(v_2) - \lambda(r_1)$.

As in the case of the cycles, before the algorithm itself is called, the global variables n and k are set with the size of the wheel and desired magic constant respectively, and the available list is initialized to every label being currently available. The actual algorithm begins with an initialization phase (by a call to **initializeWheel()**) which labels s_1, e_1, e_n, and v_1 in such a way as to prevent isomorphic labelings from being generated. The initialize function then calls **extendWheel(2)** which recursively labels a spoke, exterior vertex, and rim and calls itself until only s_n, v_n, and the hub remain, which are then labeled by a call to **finalizeWheel()**. Also like the cycle algorithm, a linked list consisting of the unused labels is maintained in order to improve efficiency as the partial labeling becomes more complete.

The initialization function starts the labeling by attempting every possible label for spoke edge s_1 and rim edge r_1. Every possible label greater than $\lambda(r_1)$ is applied to r_n, thus removing reflective symmetry. This determines the label for vertex v_1. The initialization function then calls the extend function with parameter 2. The pseudocode for the initialization function is given in Figure 7.

The extend function takes a single parameter t which gives the position of the wheel currently being expanded and applies every possible label greater than $\lambda(s_1)$ to s_t. The extend function then applies every possible label to v_t which determines the label for r_t. The finalize function is called when $t = n$.

In order to prune the computation tree more effectively, we keep a close watch on the weight of the hub vertex through the variable $hubWeight$. Due to its high degree, its weight can easily exceed the desired magic constant. Every time a label is applied to a spoke, the partial hub weight variable is updated. Once in each iteration of the extend method, we check to ensure that the minimum weight the hub can achieve is less than or equal to the desired magic constant. The minimal weight is given by the partial weight plus the smallest unused label (applied to the hub) and the $n-t$ smallest unused labels which are greater than $\lambda(s_1)$. If the minimal hub weight is larger than the desired magic constant, the partial labeling fails and the next set of labels is considered. The pseudocode for the extend function can be found in Figure 8.

The finalize function tries every available label for s_n which is greater than $\lambda(s_1)$. This determines the labels for both v_n and the hub. If these last labels

```
function initializeWheel ()
    for each available label i do
        λ(s_1) := i
        hubWeight := λ(s_1)
        avail[i] := false
        for each available label j do
            λ(r_1) := j
            avail[j] := false
            for each available label p where p > λ(r_1) do
                λ(r_n) := p
                avail[p] := false
                λ(v_1) := k − λ(s_1) − λ(r_1) − λ(r_n)
                if 0 < λ(v_1) ≤ 3n + 1 and avail[λ(v_1)] then
                    avail[λ(v_1)] := false
                    extendWheel (2)
                    avail[λ(v_1)] := true
                avail[p] := true
            avail[j] := true
        avail[i] := true
```

Fig. 7. Pseudocode for the initialization function for wheels. Global variables n and k are set to the desired values before the initializeWheel function is called.

can be successfully applied, then the **Print()** method is called which increments the number of labelings. Figure 9 gives the pseudocode for the finalize function.

4.1 Results

Table 3 gives the total number of VMTLs on the wheels W_3 through W_{10}, of which W_6 through W_{10} had not previously been enumerated. Results on W_{11} are currently pending completion. Table 4 gives the number of unique labelings broken down by magic constant. Of particular note is the fact that W_9 does not have a VMTL for $k = 58$ even though it is in the feasible range as given by MacDougall, Miller, and Wallis in [3]. Goemans gave a counting argument showing why no VMTL on W_9 can have $k = 58$ after we posed the problem of the missing labeling [6].

5 Conclusion and Open Problems

As there are EMTLs/VMTLs for all cycles C_n with $n \geq 3$, the number of unique labelings on larger cycles remain an open problem. It is desirable, however, to determine a formula which gives the number of EMTLs/VMTLs on a cycle of size n without having to actually count them.

In addition to the wheels, MacDougall, Miller and Wallis present other related classes of graphs which have similar size restrictions [3]. Figure 10 gives

```
function extendWheel (t)
    if t = n then
        finalizeWheel()
    else
        for each available label i where i > λ(s_1) do
            λ(s_t) := i
            avail[i] := false
            hubWeight := hubWeight + λ(s_t)
            potentialHub := the minimum available label
            potentialSpokes := the sum of the n − t smallest available labels > λ(s_1)
            if (hubWeight + potentialHub + potentialSpokes < k) then
                for each available label j do
                    λ(r_t) := j
                    avail[j] := false
                    λ(v_t) := k − λ(s_t) − λ(r_t) − λ(r_{t−1})
                    if 0 < λ(v_t) ≤ 3n + 1 and avail[λ(v_t)] then
                        avail[λ(v_t)] := false
                        extendWheel (t + 1)
                        avail[λ(v_t)] := true
                    avail[j] := true
            avail[i] := true
```

Fig. 8. Pseudocode for the extend function for wheels. Global variables n and k are set to the desired values before the extendWheel function is called.

sample graphs for three of these related classes: fans, t-fold wheels, and friendship graphs.

Acknowledgements

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:`www.sharcnet.ca`).

References

1. Wallis, W.D.: Magic Graphs. Birkhäuser, New York, NY, USA (2001)
2. Gallian, J.A.: A dynamic survey of graph labeling. The Electronic Journal of Combinatorics **15** (2008) #DS6
3. MacDougall, J.A., Miller, M., Wallis, W.D.: Vertex-magic total labelings of wheels and related graphs. Utilitas Mathematica **62** (2002) 175–183
4. MacDougall, J.A., Miller, M., Slamin, Wallis, W.D.: Vertex-magic total labelings of graphs. Utilitas Mathematica **61** (2002) 3–21
5. Godbold, R.D., Slater, P.J.: All cycles are edge-magic. Bulletin of the ICA **22** (1998) 93–97
6. Goemans, M. personal communication (2008)

```
function finalizeWheel ()
    for each available label i where i > λ(s_1) do
        λ(s_n) := i
        avail[i] := false
        λ(v_n) := k − λ(s_n) − λ(r_n) − λ(r_{n−1})
        if 0 < λ(v_n) ≤ 3n + 1 and avail[λ(v_n)] then
            avail[λ(v_n)] := false
            λ(hub) := k − Σ_{i=1}^{n} λ(s_i)
            if 0 < λ(hub) ≤ 3n + 1 and avail[λ(hub)] then
                Print ()
            avail[λ(v_n)] := true
        avail[i] := true
```

Fig. 9. Pseudocode for the finalize function for wheels. Global variables n and k are set to the desired values before the finalizeWheel function is called.

Table 3. The total number of non-isomorphic VMTLs for wheel graphs W_n ($3 \leq n \leq 8$).

W_n					
n	Unique VMTLs	n	Unique VMTLs	n	Unique VMTLs
3	14	6	859404	9	17804388662
4	2080	7	22063500	10	418858095690
5	31892	8	637402504	11	pending

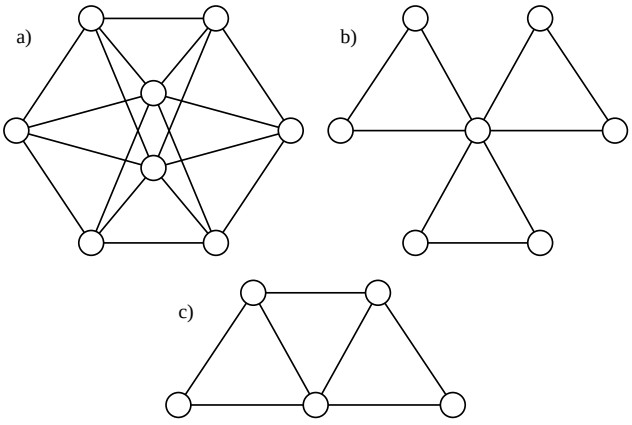

Fig. 10. Examples of graphs in three classes related to the wheels: a) t-fold wheel, b) friendship graph, c) fan.

Table 4. The number of unique VMTLs for wheel graphs W_3 through W_5 broken down by magic constant (k).

W_3		W_4		W_5	
k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs
19	0	26	89	32	239
20	2	27	149	33	1242
21	5	28	522	34	2694
22	0	29	376	35	5180
23	5	30	573	36	7873
24	2	31	211	37	7173
25	0	32	131	38	4124
		33	29	39	2511
				40	776
				41	80

W_6		W_7		W_8	
k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs
39	5978	45	24998	52	795294
40	36945	46	204170	53	7352502
41	76335	47	880257	54	28521585
42	158805	48	2198247	55	64090384
43	173887	49	3637665	56	106131735
44	187409	50	4760707	57	132239986
45	116447	51	4425875	58	133415487
46	77827	52	3384967	59	92798616
47	21793	53	1818749	60	53134373
48	3978	54	646233	61	17008206
		55	81632	62	1914336

W_9		W_{10}		W_{11}	
k	Unique VMTLs	k	Unique VMTLs	k	Unique VMTLs
58	0	66	1739667155	78	pending
59	34364364	67	4780216858	79	pending
60	236314351	68	18515045434	80	pending
61	833847423	69	39874554946	81	pending
62	1846542901	70	75518840087	82	162942689359
63	2996328931	71	84888911188	83	8201853531
64	3821193834	72	90187289669		
65	3553033163	73	60230503071		
66	2649033979	74	33425583234		
67	1364327018	75	9122758622		
68	435740211	76	574725426		
69	33662487				