Algorithms for Finding Magic Labelings of Triangles

James M. McQuillan\(^1\) and Dan McQuillan\(^2\)

\(^1\)School of Computer Sciences, Western Illinois University, Macomb, IL, USA
\(^2\)Department of Mathematics, Norwich University, Northfield, Vermont, USA

Abstract—We present a new algorithm for finding vertex-magic total labelings of disjoint unions of triangles. Since exhaustive searches are infeasible for large graphs, we use a specialized algorithm designed to find labelings with very restrictive properties, and then attempt to generate other labelings from these. We show constructively that there exists a vertex-magic total labeling, VMTL, for each of the feasible values of \(7C_3, 9C_3\) and 11\(C_3\).

Keywords: graph labeling, algorithm, vertex-magic

1. Introduction

Let \(G\) be a simple graph with vertex set \(V\) and edge set \(E\). A total labeling of a graph is a bijective map \(f: V \cup E \rightarrow \{1, 2, \ldots, |V|+|E|\}\). The weight of a vertex \(v\) incident with edges \(e_1, \ldots, e_t\) is \(wt_f(v) = f(v)+f(e_1)+\cdots+f(e_t)\). The total labeling \(f\) is a vertex-magic total labeling, or VMTL, if the weight of each vertex is a constant. In this case, the constant is called the magic constant of the VMTL. If a graph \(G\) has a VMTL, then \(G\) is called a vertex-magic graph.

Given a VMTL \(f\) of a graph \(G\) with degree \(\Delta\), there is a dual VMTL \(f^*\) of \(G\) in which \(f^*(x) = |V|+|E|+1-f(x)\) for each \(x \in V \cup E\). If the magic constant of \(f\) is \(h\), then the magic constant of \(f^*\) is \((\Delta+1)(|V|+|E|+1)-h\).

The range of possible magic constants for a graph is called the spectrum of the graph. Assume that \(G\) is a 2-regular graph. Suppose further that \(G\) has a VMTL with magic constant \(h\) and with \(n = |V| = |E|\). Then

\[
\frac{5n+3}{2} \leq h \leq \frac{7n+3}{2}.
\]

The integral values in this range are the feasible values for the 2-regular graph \(G\). (See [24] and [34] for a more general discussion.) For a 2-regular graph, the dual of a VMTL with magic constant \(h\) is a VMTL with magic constant \(6n+3-h\).

Problem 1.1: The VMTL problem: given a feasible magic constant \(h\), does there exist a VMTL with magic constant \(h\)?

There have been many exciting general VMTL constructions in recent years, such as [15], [11], and [22]. Some other important papers for labeling regular graphs include [21], [5], [13], [31], [2], [3] and [23]. For 2-regular graphs, Wallis proved in [33] that, for a vertex-magic regular graph \(G\), the multiple graph \(sG\) is also vertex-magic provided that \(s\) is odd or the degree of \(G\) is odd. Gray gave VMTLs of \(C_3 \cup C_{2n}, n \geq 3\) and \(C_4 \cup C_{2n-1}, n \geq 3\) in [14]. In [27], it was shown that, for every \(s \geq 4\) even, \(sC_3\) is vertex-magic. For every \(s \geq 6\) even, \(sC_3\) has VMTLs with at least \(2s-2\) different magic constants. For every \(s\) odd, VMTLs for \(sC_3\) with \(s+1\) different magic constants were also provided. In this paper, we are interested in algorithms that can be used to construct VMTLs in \(sC_3\).

One approach to Problem 1.1 is to design an algorithm to construct VMTLs for certain special types of graphs. This is a part of our strategy as our algorithm is specifically designed for \(sC_3\). An algorithm could be designed in an efficient way using properties of those graph types. For example, in [4], algorithms were designed specifically for finding all VMTLs on cycles and wheels. They provide a table giving the total number of VMTLs on cycles \(C_3\) through \(C_{18}\). Moreover, they give the number of VMTLs for these for a given magic constant. They give a table giving the number of VMTLs for wheel graphs \(W_3, \ldots, W_{10}\). Moreover, they give the number of VMTLs for these for given magic constants. (They also give the number of VMTLs for some of the feasible magic constants for \(W_{11}\).)

A second strategy for approaching this problem is to convert it into another well-studied problem. This approach is taken in [20], where they convert an instance of the VMTL problem into an instance of the satisfiability of boolean expressions problem. The SAT problem was the first NP-complete problem ([7]) and is fundamental to computer science. Variants of the Davis-Putnam algorithm ([9]), BDD’s ([6]), and BED’s ([11]) are the three main complete algorithms for the SAT problem. There are also many incomplete local search algorithms. GSAT, WalkSAT ([32]), and UnitWalk ([17]) were three of the early local search algorithms. There is a yearly competition to find an efficient SAT solver as part of the annual SAT conference [19]. Given the amount of research that has been done on this problem, it is natural to consider converting an instance of a problem into an instance of the SAT problem.

A third strategy, which is the approach taken here, is to combine mathematical intuition for finding new VMTLs in certain graphs with very focused, specific algorithms to generate labelings with certain properties. With this strategy, labelings might be found that other more general algorithms could not find in a reasonable amount of time. Such labelings might then be used to construct other labelings. This strategy was used in [27] and [18].

As mentioned above, tables are provided in [4] that give the number of VMTLs for \(C_3\) through \(C_{18}\). (They note
Lemma 2.1. We use it to make our algorithm more efficient. In our algorithm, Lemma 2.2 follows immediately from $b$. The common difference of this component is defined to be $d = b_1 - a_1 = b_2 - a_2 = b_3 - a_3$. This concept is a key ingredient in our algorithm. Lemma 2.2 follows immediately from Lemma 2.1. We use it to make our algorithm more efficient.

Lemma 2.2: Suppose that we have a VMTL of $sC_3$ with magic constant $h$, and that a component has common difference $d$ and labels $[a_1, b_3, a_2, b_1, a_3, b_2]$. Then

(i) $b_i = a_i + d$, $i = 1, 2, 3$;
(ii) the weight of each vertex in that component is $h = a_1 + (a_2 + d) + (a_3 + d) = a_1 + a_2 + a_3 + 2d$;
(iii) the vertex sum of that component is $h = 2d$;
(iv) $d = (h - (a_1 + a_2 + a_3))/2$; and
(v) $h - (a_1 + a_2 + a_3)$ is an even integer.

Proof: All of these follow immediately from Lemma 2.1.

While it is trivial to rearrange any of (ii)-(iv) to obtain the others in Lemma 2.2, we prefer to state all of (i)-(v) so that we can refer to them individually. In an algorithm, when presented with three integers as possible labels for the vertices of one component, we can remove that triple from any further consideration if $h$ minus the sum of the three numbers is not even by Lemma 2.2 part (v). If we wanted a component to have a specific common difference, we could also remove a triple from contention for vertex labels of a component if any of $a_1 + d$, $a_2 + d$, $a_3 + d$ had already been assigned to other components by Lemma 2.2 part (i). We use techniques such as these in our algorithm.

Because exhaustive computer searches are only feasible for small $s$, we chose to write computer programs that are specifically designed to look for VMTLs with very restrictive properties. Such a strategy might not lead to any examples of VMTLs, or perhaps it might lead to just a few sporadic examples. On the other hand, it might lead to some important examples that could be used to generate others. For example, consider the following special case of Problem 1.1.

Problem 2.1: Given a feasible magic constant $h$ of $sC_3$, does there exist a VMTL with magic constant $h$ such that the labels $1, 2, \ldots, s_0$ are on vertices of different components, $s_0 \leq s$, and such that the common difference on each of these $s_0$ components is $3s$?

This seems like quite a restrictive problem, but an efficient algorithm can be tailor-made for this problem, and any solution is very valuable when combined with the following lemma, as it could be used to generate many solutions to Problem 1.1, especially if $s_0$ is close to $s$.

Lemma 2.3 ([27] Lemma 3): Let $s$ be a positive integer and let $\lambda_0$ be a VMTL of $sC_3$ with a magic constant of $h$. Assume that one of the components has a vertex labeled 1 and a common difference of $3s$. Then there exists another VMTL $\lambda_1$ of $sC_3$ with a magic constant of $h - 3$.

We gave some solutions to Problem 2.1 in [27, Theorem 4] for $s$ odd, $s_0 = s$, and [27, Theorem 6] for $s \geq 6$ even, $s_0 = s - 2$.

3. A magic labeling algorithm for $sC_3$.

In this section, we present an algorithm for generating VMTLs in $sC_3$. We will use the following proposition to make our algorithm more efficient.
Proposition 3.1: A VMTL of $sC_3$ with a magic constant of $h$ has sum of all vertex labels equal to $3s[2(6s + 1) - h]$.

Proof: Let $S_V$ denote the sum of all the vertex labels, and let $S_E$ denote the sum of all the edge labels. Let $n = |V|$ and $m = |E|$. It is easy to check that $S_V + S_E = 1 + 2 + \cdots + (n+m) = (n+m)(n+m+1)/2$ and $S_V + 2S_E = nh$. (For proofs of these, see [27].) Here, $n = m = 3s$. Solving for $S_V$ yields the desired equality.

In Algorithm 3.1 below, we assign labels to some components and then attempt to extend that to a VMTL. Our algorithm has a heavy emphasis on finding labels of vertices in individual components. We make sure that the appropriate and necessary edge labels are available at the same time. Here, we use Lemmas 2.1 and 2.2. If we know the vertex labels and the common differences of some components of a VMTL, we know the edge labels of those components.

For convenience, we use the following term in this situation to refer to the labels assigned to some but not necessarily all of the components. An injective vertex mapping, IVML, of $sC_3$ is an injective map $f$ from a subset $A$ of $V \cup E$ into $\{1, 2, \ldots, |V| + |E|\}$ such that $A$ consists of zero or more components of $sC_3$, and the weight of each vertex in $A$ is a constant. Given a IVML, erasing the vertex and edge labels from one or more components gives an IVML.

We are hoping to go in the other direction. We are hoping to extend some IVMLs to VMTLs. Note that Lemma 2.1 applies to IVMLs as well as VMTLs.

Algorithm 3.1: A VMTL algorithm for $sC_3$

Input:
(a) the value of $s$ that you want to find VMTLs for.
(b) if you are only interested in VMTLs for a particular magic constant $h$, then that value of $h$ is taken as input as well.
(c) any information about restrictions there are on any of the labels that can be used to make the program run faster.

Output: all VMTLs that satisfy the given restrictions.

Our algorithm uses the following procedures:
1. main()

   If a particular value for the magic constant $h$ was not given as input, then we can use a loop to look for VMTLs for all feasible values of $h$. Equation (1) gives the range of feasible values. If we denote by $h_{max}$ and $h_{min}$ the upper and lower bounds in this inequality respectively, then let $h_{mid} = [(h_{min} + h_{max})/2]$. Then the loop could go from $h_{max}$ down to $h_{mid}$. By duality, the VMTLs for the other values of $h$ can be generated later from those. (Note that the algorithm could look for VMTLs for different values of $h$ in parallel.)

For each value of $h$, calculate the sum $k$ that the vertex labels of any VMTL must have according to Proposition 3.1. The main should then call the procedure whichLabelsAreForVertices by a call such as whichLabelsAreForVertices($s, h, \{1, \ldots, 6s\}, 3s, k, 0, 0, 0, \text{null}$); described below.

Note: Three of the arguments for this call are $s$, $h$ and $k$. The argument $\{1, \ldots, 6s\}$ represents the set of all possible labels, and the argument $3s$ represents the size of a set of labels for the vertices. Other arguments are used for the recursive nature of that procedure. They are described in that procedure.

2. procedure whichLabelsAreForVertices(int $s$, int $h$, int[] theNumbersForLabels, int desiredSize, int desiredSum, int sizeSoFar, int sumSoFar, int currentIndexOfNumbersForLabels, int[] soFarForVertices);

This recursive procedure attempts to grow the empty set to sets each having size $desiredSize$ and sum $desiredSum$. For each such set that is found, the procedure lookForVMTLsUsingIt is called in the hope that it can be used to label the vertices of $sC_3$ as part of VMTLs with magic constant $h$ and vertex sum $desiredSum$.

To grow the empty set to such a set of size $desiredSize$ and sum $desiredSum$, we have parameters to keep track of the set of elements we have chosen so far, soFarForVertices (initially null), as well as its size, sizeSoFar (initially 0), and its sum, sumSoFar (initially 0). The parameter currentIndexOfNumbersForLabels (initially 0) keeps track of what element of theNumbersForLabels that we are currently considering to include from soFarForVertices.

This recursive procedure could have multiple base cases. To make this procedure more efficient, one base case could simply return if

\[
\text{length} (\text{theNumbersForLabels}) - \text{currentIndexOfNumbersForLabels} < (\text{desiredSize} - \text{sizeSoFar}).
\]

In such a case, there is no hope of extending soFarForVertices to a desired set because there aren’t enough numbers left to consider in theNumbersForLabels to extend soFarForVertices to a set of size desiredSize.

Another base case could simply return if sumSoFar > desiredSum, as there is no hope of extending soFarForVertices to a desired set because we’ve already exceeded the desired sum.

Another base case could simply return if

\[
\text{currentIndexOfNumbersForLabels} \geq \text{length} (\text{theNumbersForLabels})
\]

as we’ve reached the end of the elements being considered in numbersForLabels in this case.

This recursive procedure should have a base case corresponding to the successful generation of a set with the
The procedure `generateVMTLs` lookForVMTLsUsingIt

To complete this procedure, we need two recursive calls, one of which attempts to use the current element of `theNumbersForVertices` to make a desired set of vertex labels, and the other of which attempts to make a desired set of vertex labels without the current element. The first corresponds to using the current element of `theNumbersForVertices`,

theNumbersForVertices[currentIndexOfChoices].

We recurse with that together with the information in the parameter `soFarForVertices` for the last argument. (We also have to use the appropriate updated values for the sixth and seventh arguments corresponding to size and sum and the eighth argument corresponding to the element of `theNumbersForVertices` that should be considered next.)

The second recursive call corresponds to us not selecting the current element of `theNumbersForVertices`. For this recursive call, no new information is being added to the information in `soFarForVertices`. The eighth argument corresponding to the element of `theNumbersForVertices` needs to be updated.

procedure lookForVMTLsUsingIt(int s, int h, int[] theNumbersForLabels, int[] theNumbersForVertices);

Since we have a specific subset of all possible labels to use for the vertices, the remaining labels must be for the edges. Store these in int[] theNumbersForEdges. Next, call the recursive procedure `generateVMTLs`, with a call such as `generateVMTLs(theNumbersForVertices, theNumbersForEdges, h, s, {false, . . . , false}, null, −1);`.

This procedure will attempt to generate VMTLs using these.

procedure generateVMTLs(int[][] theNumbersForVertices, int[][] remainingNumbersForEdges, int h, int s, boolean[] usedVertexIndex, int[][] ivmlIndices, int[] commonDifferences, int currentIndexOfIVML);

This procedure tries to find VMTLs by labeling one component at a time. The first parameter, `theNumbersForVertices`, gives the set of numbers that is to be used for labeling all the vertices. The parameters `ivmlIndices` and `commonDifferences` keep track of an IVML that we are trying to grow. The parameter `ivmlIndices` keeps track of triples of indices in `theNumbersForVertices`. One triple corresponds to one component, so if we ever get `s` triples we have our vertex labels. From such a triple, the corresponding three elements in `theNumbersForVertices` are currently being used as the vertex labels for one component. The common difference for that component can be found in the appropriate element in `commonDifferences`. From these pieces of information, we have the edge labels for that component.

As we grow an IVML, some labels are used for edges. The second parameter keeps track of the edge labels that have not as yet been used. The first parameter, `theNumbersForVertices`, is the set of all of the labels for the vertices, and the `usedVertexIndex` parameter keeps track of those that have been used so far.

This procedure has a base case corresponding to finding a solution. If the parameter `ivmlIndices` has length `s`, then it corresponds to a VMTL, in which case that information is outputted, and the procedure `returns`. This procedure has a loop to continually look for all possible labelings of one component that could be used to augment the information in `ivmlIndices` and `commonDifferences`, to give a bigger IVML. The corresponding edge labels need to be available. (Recall that Lemma 2.2 part (i) gives us these values.) If they are, then we can recurse on this bigger IVML with all parameters except the first updated. When the loop is finished, this procedure `returns`.

It is important to note that we have several filters in place in the loop above so that (a) we ignore labels of a component that won’t lead to a solution, and (b) so that we ignore labels of a component that won’t lead to a specific type of solution that we are after. Filters of type (a) are used for the purposes of making the algorithm faster. If we exclude filters of the type (b), then our algorithm will find all possible VMTLs for the specified values of `s` and `h`. If we include filter of type (b), we will only be looking for VMTLs with certain properties. We often use filters of both types in order to find certain solutions quickly. We will discuss the use of filters further shortly.

Finally, we can generate other VMTLs from the output of `generateVMTLs` using Lemma 2.3. This can be done by hand (or by another procedure).

Algorithm 3.1 uses several tools so that it is efficient:

(i) In the `main`, our loop considered roughly half of the feasible magic constants. Duality can be used for the others. Parallelism can also be used here.

(ii) In the second procedure, `whichLabelsAreVertices`, we used the results of Proposition 3.1 to restrict the possibilities for sets of numbers to be considered as vertex labels.

(iii) In `generateVMTLs()`, when we looked for a triple to label the vertices of one component, we eliminated possibilities that did not satisfy Lemma 2.2 part (v).
We were able to focus on labeling the vertices because of Lemma 2.2 part (iv). We just needed to make sure that the corresponding edge labels were available when we had labels that we liked for the vertices of a component.

Sometimes, we are interested in finding only labellings in which some components have specific common differences. In this case, Lemma 2.2 part (iv) helps us reduce the number of triples that we need to consider as possible labels for the vertices of a component. In generateVMTLs(), when we looked for a triple to label the vertices of one component, we eliminated possibilities that did not satisfy Lemma 2.2 part (iv) in this situation.

Use Lemma 2.3 to generate many other VMTLs.

In Algorithm 3.1, we use several filters.

In generateVMTLs, when we have an IVML with some components labeled, we try to extend it to VMTLs by labeling an additional component (recursively). Sometimes, a potential labeling of a “next” component can be removed from consideration (“filtered”) because we can tell that it will never lead to a VMTL, or because it will not lead to a specific type of VMTL that we are interested in finding.

If we are given the common differences of any of the components as input, then some triples can be immediately eliminated from consideration as possible vertex labels for a component because of Lemma 2.2 part (iv). If restrictions are given, such as the common difference restriction of Problem 2.1, we might be able to eliminate triples here or at some other point in the algorithm.

4. VMTLs for 7C₃, 9C₃, and 11C₃.

Using Algorithm 3.1, we now answer Problem 1.1 in the affirmative for 7C₃, 9C₃, and 11C₃ for each of their respective feasible magic constants. In order to do so, we answer Problem 2.1 in the affirmative for certain values of h and s₀ for 7C₃, 9C₃, and 11C₃. Then, we use Lemma 2.3.

Theorem 4.1: There exists VMTLs for all of the feasible values of 7C₃.

Proof: Consider the VMTL with components labeled

\[ [1, 37, 15, 22, 21, 16, 36], [2, 39, 12, 23, 18, 33], [3, 42, 8, 24, 21, 29], [4, 40, 9, 25, 19, 30], [5, 38, 10, 26, 17, 31], [6, 41, 20, 13, 34, 27], \text{and} [7, 35, 11, 28, 14, 32]. \]

This VMTL has a magic constant of 74. The first three components have common differences of 3s. Therefore, by using Lemma 2.3 three times, we have VMTLs of 7C₃ with magic constants 71, 68, 65. By duality, there exist VMTLs with magic constants of 55, 58, 61 and 64.

Next, consider the VMTL with components labeled

\[ [1, 37, 14, 22, 16, 35], [2, 38, 12, 23, 17, 33], [3, 42, 20, 11, 34, 28], [4, 40, 15, 18, 26, 29], [5, 36, 6, 31, 10, 32], [7, 39, 9, 25, 21, 27], \text{and} [8, 41, 13, 19, 30, 24]. \]

This VMTL has a magic constant of 73. The first two components have common differences of 3s. By using Lemma 2.3 twice, there exist VMTLs of 7C₃ with magic constants 70 and 67. By duality, there exist VMTLs with magic constants of 56, 59 and 62.

Finally, in [27] it was shown that there exist VMTLs of 7C₃ with magic constants with values 75, 72, 69, 66, 63, 60, 57, 54.

Theorem 4.2: There exists VMTLs for all of the feasible values of 9C₃.

Proof: Consider the VMTL with components labeled

\[ [1, 46, 18, 28, 19, 45], [2, 52, 11, 29, 25, 38], [3, 49, 13, 30, 22, 40], [4, 54, 12, 26, 32, 34], [5, 44, 9, 39, 10, 43], [6, 53, 16, 23, 36, 33], [7, 50, 27, 15, 42, 35], [8, 47, 31, 14, 41, 37], \text{and} [17, 51, 21, 20, 48, 24]. \]

This VMTL has a magic constant of 92. The first three components have common differences of 3s. Therefore, by using Lemma 2.3 three times, we have VMTLs of 9C₃ with magic constants 89, 86, and 83. By duality, there exist VMTLs with magic constants of 73, 76, 79, and 82.

Next, consider the VMTL with components labeled

\[ [1, 47, 19, 28, 20, 46], [2, 50, 15, 29, 23, 42], [3, 51, 13, 30, 24, 40], [4, 49, 9, 36, 17, 41], [5, 52, 16, 26, 31, 37], [6, 45, 10, 39, 12, 43], [7, 53, 14, 27, 33, 34], [8, 54, 22, 18, 44, 32], \text{and} [11, 48, 25, 21, 38, 35]. \]

This VMTL has a magic constant of 94. The first three components have common differences of 3s. By using Lemma 2.3 thrice, there exist VMTLs of 9C₃ with magic constants of 91, 88, and 85. By duality, there exist VMTLs with magic constants of 71, 74, 77, and 80.

Next, consider the VMTL with components labeled

\[ [1, 50, 17, 28, 23, 44], [2, 47, 19, 29, 20, 46], [3, 51, 14, 30, 24, 41], [4, 54, 16, 25, 33, 37], [5, 48, 12, 35, 18, 42], [6, 53, 10, 32, 27, 36], [7, 45, 11, 39, 13, 43], [8, 49, 15, 31, 26, 38], \text{and} [9, 52, 22, 21, 40, 34]. \]

This VMTL has a magic constant of 95. By duality, there exists a VMTL with a magic constant of 70.

Finally, in [27] it was shown that there exist VMTLs of 9C₃ with magic constants with values 96, 93, 90, 87, 84, 81, 78, 75, 72 and 69.

Theorem 4.3: There exists VMTLs for all of the feasible values of 11C₃.
Proof: Consider the VMTL with components labeled
\[
[1, 62, 20, 34, 29, 53], [2, 65, 16, 35, 32, 49],
[3, 61, 19, 36, 28, 52], [4, 57, 22, 37, 24, 55],
[5, 60, 18, 38, 27, 51], [6, 56, 10, 50, 12, 54],
[7, 66, 17, 33, 40, 43], [8, 63, 23, 30, 41, 45],
[9, 59, 15, 42, 26, 48], [11, 58, 14, 44, 25, 47],
[13, 64, 21, 31, 46, 39].
\]

This VMTL has a magic constant of 116. The first five components have common differences of 3s. Therefore, by using Lemma 2.3 five times, we have VMTLs of 119C₃ with magic constants 113, 110, 107, 104 and 101. By duality, there exist VMTLs with magic constants of 85, 88, 91, 94, 97, and 100.

Next, consider the VMTL with components labeled
\[
[1, 59, 22, 34, 26, 55], [2, 60, 20, 35, 27, 53],
[3, 66, 13, 36, 33, 46], [4, 63, 15, 37, 30, 48],
[5, 56, 10, 49, 12, 54], [6, 58, 17, 40, 24, 51],
[7, 65, 19, 31, 41, 43], [8, 57, 11, 47, 18, 50],
[9, 64, 23, 28, 45, 42], [14, 62, 21, 32, 44, 39],
[16, 61, 29, 25, 52, 38].
\]

This VMTL has a magic constant of 115. The first four components have common differences of 3s. By using Lemma 2.3 four times, there exist VMTLs of 111C₃ with magic constants of 112, 109, 106 and 103. By duality, there exist VMTLs with magic constants of 86, 89, 92, 95, and 98.

Finally, in [27] it was shown that there exist VMTLs of 11C₃ with magic constants with values 117, 114, 111, 108, 105, 102, 99, 96, 93, 90, 87 and 84.

References