

Symmetries in Graceful Trees

Nick Kraayenbrink, Frits de Nijs, and Mladen Vavic

Technical University Delft

{N.Kraayenbrink, F.deNijs, M.Vavic}@student.tudelft.nl

Abstract. We study how to exploit symmetries in the context of the famous graceful tree conjecture. Current work on graceful labelling of graphs focused on breaking the value symmetry and the structural symmetries of a graph. We observed two symmetries within solutions for graceful trees. Based on these new symmetries, we define a stricter version of the graceful tree conjecture. Experimental results show that, for all the trees that we have studied (up to size 23), a graceful labelling exists for this stricter version. Moreover, the computational costs to find a graceful labelling is significantly reduced due to the restrictions. Besides the improved speed, this stricter definition can provide useful insights to develop a general construction method for graceful tree labellings.

1 Introduction

A graph labelling is a mapping that assigns a non-negative integer value to each vertex and each edge in the graph, possibly with some restrictions. A graph labelling is said to be a graceful labelling if for a graph of n edges each vertex is assigned a distinct value from $\{0, \dots, n\}$ in such a way that each edge is assigned a distinct value from $\{1, \dots, n\}$, and the value on every edge e is equal to the absolute difference in the values on the endpoints [1]. The Graceful Labelling problem was first introduced in 1967 by Rosa in an attempt to progress towards solving Ringel's Conjecture, although it was introduced as a β -valuation of a graph. The result of Rosa's work was a search for proof of the stronger Graceful Tree Conjecture: All trees are graceful [2,1,3].

Although the problem of gracefully labelling a graph may look like just another mathematical curiosity with no real-world use, there are a wide range of applications of graceful trees (or graceful graphs in general). Bloom and Golomb describe in [4] several applications of graceful graphs, including in the field of coding theory and the design of optimal circuit layouts. A more recent paper by Basak discusses a use of graceful caterpillar graphs (which is a class of trees) in the field of networking [5].

Our research focuses on detecting patterns in graceful labellings of trees. More specific, we search for patterns that occur in *at least one* graceful labelling in *all* trees within the experimented domain. Based on our observations of these patterns, we present two new conjectures. Both are stonger variants of the Graceful Tree Conjecture. Also, we exploit these patterns in the context of internal symmetries [6] to reduce the computational cost to find graceful labellings.

Although there have been recent attempts to prove the Graceful Tree Conjecture directly [7,8], most research has focused on proving the Graceful Tree Conjecture on

more narrowly restricted classes of trees. A dynamic survey of Gallian [2] provides a comprehensive overview.

There have also been several initiatives to find graceful labellings of trees of a certain size. Aldred and McKay describe in [9] the probabilistic search algorithm they have used to find a graceful labelling for all trees up to 27 vertices. A more recent effort has managed to extend the number of calculated trees to all trees of 35 or less vertices, using a combination of a deterministic backtracking algorithm and probabilistic search in a distributed computing environment [10]. Furthermore, [11] presented a mathematical programming approach which they used to evaluate the gracefulness of a number of trees of 40 vertices.

2 Symmetries

Solving any given problem using just the definition of the problem may take longer than required, and does not necessarily lead to a proper grasp of the problem. By analyzing the problem and its solution space, symmetries may be found which can speed up the process of solving the problem. Aside from providing a speedup, discovering such symmetries may also lead to valuable insights in the nature of the problem. This section discusses two such families of symmetry for the Graceful Labelling problem, and how these symmetries can be broken in order to make use of them when searching for solutions.

2.1 Symmetries Between Solutions

Symmetries between solutions are symmetries that map solutions onto other solutions [6]. These type of symmetries are oftenly part of the problem itself, or of the problem instance. A symmetry between solutions maps any solution onto another solution for the same problem instance.

The Graceful Labelling problem for trees contains two types of symmetries between solutions; value-symmetry and structure-symmetry. For every tree, there exists one value-symmetry, and zero or more structure-symmetries.

Value-symmetry is a symmetry inherent in the Graceful Labelling problem. For every labelling ℓ there exists a symmetric labelling ℓ' for which each vertex is assigned the conjugate of the label it was assigned in ℓ . In other words, if a vertex had the label l_i in labelling ℓ , it would get the label $n - l_i$ in labelling ℓ' . The edge labels are identical for ℓ and ℓ' , as the differences between the labels of the vertices do not change by this transformation.

Structure-symmetry is introduced because of graph isomorphism. This may be more easily explained by way of an example. The tree in Fig. 1 is a complete binary tree of depth 2, which contains three such symmetries. Swapping the vertices labelled 4 and 5 will not change the fact that the labelling is graceful, nor will swapping the labels 1 and 2. The final symmetry comes from the two branches that have the vertex labelled 0 as common parent. When the vertex labelling of these branches is swapped (which is possible since the branches are identical), the complete labelling will remain graceful.

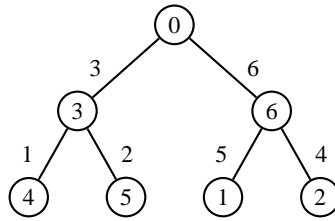


Fig. 1: Example of a tree with three structural symmetries. The shown labelling is one of three non-isomorphic solutions.

These symmetries can be useful, but they may also hinder the fast discovery of a solution. If a solver finds a partial but ungraceful labelling, the partial labellings within the same symmetry group will also not be graceful, therefore reducing the search space of the solver. However when symmetries are broken, the number of solutions to the problem also decreases. Therefore a solver may also find a graceful labelling, but it does not recognize it as such because the labelling is not allowed due to the symmetry breaking constraints. Since the proportion of graceful labellings decreases as the size of the tree increases [12], adding symmetry breaking constraints will in general provide a speedup in solving time.

2.2 Symmetries within Solutions

The concept of using symmetries within solutions (or internal symmetries) to reduce the search space of a problem was introduced by Heule and Walsh in [6]. A symmetry that occurs within a solution is a symmetry that maps a solution onto itself. Internal symmetries that occur in one solution may not occur in other solutions. One objective of this paper is to detect internal symmetries that occur in *all* trees. Exploiting these symmetries can reduce the computational time needed to find a graceful labelling. Also, they may provide some insights how to proof the Graceful Tree Conjecture.

Some symmetries that could occur in a graceful labelling are the assignment of the label zero to a vertex with a special property, or the occurrence of a special ordering in the (vertex or edge) labels. In this paper we explore four such symmetries.

The first candidate vertices that could receive the label zero are the leaf vertices. By labelling a leaf to zero the neighbouring parent vertex must receive the label n , since the edge label n can only occur when its endpoints are labelled 0 and n , and the leaf attaches to only one edge. If the vertex neighbouring the leaf that was set to zero has degree two, its second edge must receive the label $n - 1$ by the same reasoning. This pattern continues until a higher degree vertex is encountered or the end is reached. Therefore setting the leaf that is at the head of a long chain to zero has a cascading effect that sets many other vertices. In the case of a path this effect immediately leads to the only possible solution where a leaf has label zero.

Another possible choice of vertex to set to zero is one of the center vertices. A vertex is considered a center when it has the shortest distance to all other vertices. A tree may have at most two central vertices, however most trees have only one. Unlike the set of

leaves of a tree which may consist of many structurally different choices, the set of tree centers is very small. Therefore intuitively restricting a center to zero reduces the search space more than setting a leaf to zero. This choice also seems promising because Fang [10] reported finding a solution for 99.99% of all trees up to size 35 by starting with a center set to zero using an algorithm with a limited number of iterations.

A third choice of vertex to set to zero is one of the vertices with the highest degree. This is the opposite of setting a leaf to zero in the sense that a leaf vertex has the lowest degree of all vertices in the tree. Setting a highest degree vertex to zero does not imply labellings on other vertices, and a tree may have many highest degree vertices (up to $n - 2$ in case of the path, although not all are structurally different). However due to its less restrictive nature it may prove to be general should the previous symmetries not be.

On the other hand it may be possible to look for symmetries where the labels adhere to a certain order. One such ordering is due to Horton [12], who conjectures that:

Conjecture 1. All trees admit a graceful labelling where every edge label other than n is adjacent to one edge of greater label.

The implication of his conjecture is that there should always be a solution where the edge labels on any path from the vertex with label zero to any other vertex in the tree are in decreasing order. Horton verified this conjecture for all trees up to size 29.

2.3 Symmetry Breaking

Just identifying the symmetries is not enough; constraints need to be added to the problem in order to break these symmetries. These constraints (called symmetry breaking constraints) can usually be implemented in many different ways, of which the optimal method highly depends on the used encoding. The following paragraphs discuss the general idea of the symmetry breaking constraints implemented for the experiments.

We will use the following variables to represent the symmetry breaking constraints. Given a tree $T = (V, E)$, for each $v \in V$ there is a variable L_v with domain $\{0, \dots, n\}$. $L_v = i$ means that vertex v has label i . Also, for each edge $\langle v, w \rangle \in E$ there is a variable $L_{\langle v, w \rangle}$ with domain $\{1, \dots, n\}$. $L_{\langle v, w \rangle} = i$ means that edge $\langle v, w \rangle$ has label i .

Value Symmetry A common way to break the value symmetry is to constrain the label of a certain vertex v by $L_v \leq \lfloor \frac{n}{2} \rfloor$. Aside from having to properly handle trees with n even, a suitable vertex must be chosen. If a ‘wrong’ vertex is chosen, this type of constraint may produce conflicts when also introducing structure symmetry breaking constraints.

Another option is to prevent the adjacency of vertex labels 0 and $n - 1$. Consequently, the vertices with label 1 and n must be connected. This symmetry breaking technique does not constrain the label of a specific vertex or edge in the tree, and is therefore easier to use in combination with structure symmetry breaking constraints. We implement it by adding for each edge $\langle v, w \rangle \in E$ the constraints $L_v = 0 \Rightarrow L_w \neq n - 1$ and $L_w = 0 \Rightarrow L_v \neq n - 1$.

Structure Symmetry A first step in breaking structure symmetries is identifying them. Structure symmetries may occur for equal branches whose roots have a common parent (see Fig. 1), and for those that connect at the same location on each other. An example of the latter type may be seen in Fig. 2, where the two branches l ('left') and r ('right') are identical and connect in their roots l_0 and r_0 . The labels of vertices $l_0 \dots l_3$ may be interchanged with the labels of vertices $r_0 \dots r_3$ due to structure symmetry.

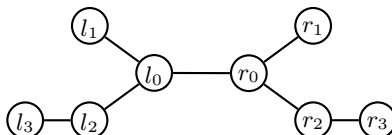


Fig. 2: Example of a tree with equal branches connected to each other. The vertices of the 'left' branch are labelled l_i , the vertices of the 'right' branch are labelled r_i .

In order to break structure symmetry, an order $L_v < L_w$ is imposed on the vertex labels of the roots of the symmetric branches. This order may be chosen arbitrarily. If the identical branches are connected at the same parent, another constraint may be to impose an ordering of the labels of the edges connecting the branches to that parent. This may not produce the same solution, but the symmetry is broken nonetheless. The optimal choice of constraints used depends on the used encoding.

Label Assignment Symmetry When looking for symmetries that assign a specific label i to a subset of vertices $V' \subset V$ or edges $E' \subset E$ a constraint is added to force only solutions that have such a symmetry to be found. We implemented these constraints as $(\bigvee_{v \in V'} L_v = i)$ in case of vertex labels and $(\bigvee_{\langle v,w \rangle \in E'} L_{\langle v,w \rangle} = i)$ in case of edge labels.

Label Order Symmetry To impose an order on the labels assigned in a solution constraints are added to disallow illegal combinations of assignments. For vertices, constraints of the type $L_v < L_w$ is added. For edges, constraints of the type $L_{\langle u,v \rangle} < L_{\langle v,w \rangle}$ are added.

3 Method

To find special properties of Graceful Labellings on Trees a transformation to the Satisfiability (SAT) domain was used. Using the SAT domain for finding Graceful labellings on graphs has a few key advantages; the Satisfiability problem has a large and active research field dedicated to improving solvers, with a yearly competition among the developers to find the fastest solver on random or industrial SAT problems¹. The solvers that compete are publicly available, and as such finding a correct and fast solver is easy.

Another advantage to using SAT is that implementing additional constraints is very easy as any new one can simply be appended as additional clauses to the existing base problem. This allowed us to construct and test new symmetries without requiring any additional implementation glue in the solving mechanism.

¹ <http://www.satcompetition.org/>

A disadvantage of using SAT solvers in a black box approach is that interaction between the program executing the solver and the solver itself has to proceed through relatively slow file I/O. And when trying to find multiple solutions the solver has to be restarted from scratch for every solution, even though the problem itself did not change. A better approach would be to internalize the solver code into the main program and exchange the problem in memory, plus keeping learned blocking clauses in memory when solving the same instance multiple times.

3.1 Graceful Labelling in SAT

How the problem of finding a graceful labelling for a tree is translated to SAT can have a large impact on the time needed to find a solution. In this section two models are explored, the vertex/edge encoding which translates each edge, label pair separately, and the edge-label encoding which links edge labels to vertices.

Table 1: Vertex-edge encoding

Required clauses	Range	Meaning
$(x_{v,0} \vee \dots \vee x_{v,n})$	$v \in V$	vertex v has at least one label
$(y_{\langle v,w \rangle,1} \vee \dots \vee y_{\langle v,w \rangle,n})$	$\langle v,w \rangle \in E$	edge $\langle v,w \rangle$ has at least one label
$(\bar{x}_{v,i} \vee \bar{x}_{w,i})$	$v, w \in V, v < w, 0 \leq i \leq n$	at most one vertex has label i
$(\bar{y}_{\langle v,w \rangle,j} \vee \bar{y}_{\langle v',w' \rangle,j})$	$\langle v,w \rangle, \langle v',w' \rangle \in E,$ $\langle v,w \rangle < \langle v',w' \rangle, 1 \leq j \leq n$	at most one edge has label j
$(\bar{x}_{v,i} \vee \bar{x}_{w,j} \vee y_{\langle v,w \rangle, i-j })$	$\langle v,w \rangle \in E,$ $0 \leq i, j \leq n, i \neq j$	if vertex v has label i and vertex w has label j then edge $\langle v,w \rangle$ has label $ i-j $
Redundant clauses	Range	Meaning
$(\bigvee_{v \in V} x_{v,i})$	$i \in \{0, \dots, n\}$	at least one vertex has label i
$(\bigvee_{\langle v,w \rangle \in E} y_{\langle v,w \rangle,j})$	$j \in \{1, \dots, n\}$	at least one edge has label j
$(\bar{x}_{v,i} \vee \bar{x}_{v,j})$	$v \in V, 0 \leq i < j \leq n$	vertex v has at most one label
$(\bar{y}_{\langle v,w \rangle,j} \vee \bar{y}_{\langle v,w \rangle,i})$	$\langle v,w \rangle \in E, 1 \leq i < j \leq n$	edge $\langle v,w \rangle$ has at most one label
$(\bar{x}_{v,i} \vee \bar{y}_{\langle v,w \rangle,j} \vee x_{w,i+j} \vee x_{w,i-j})^*$	$\langle v,w \rangle \in E,$ $0 \leq i \leq n, 1 \leq j \leq n$	if vertex v has label i and edge $\langle v,w \rangle$ has label j then vertex w has label $i \pm j$

* In case $i + j > n$, literal $x_{w,i+j}$ is omitted. Also, in case $i - j < 0$, literal $x_{w,i-j}$ is omitted.

Vertex-edge encoding. For encoding a tree $T = (V, E)$ the most straightforward model introduces Boolean variables $x_{v,i}$ for every $v \in V, i \in \{0, \dots, n\}$. If $x_{v,i}$ is set to true, it means that vertex v has label i . Also, we need Boolean variables $y_{\langle v,w \rangle,j}$

for every $\langle v, w \rangle \in E, j \in \{1, \dots, n\}$. If $y_{\langle v, w \rangle, j}$ is set to true, it means that edge $\langle v, w \rangle$ has label j . The translation therefore introduces $(n + 1)^2 + n^2$ variables.

Using these variables we can encode the graceful labelling problem with the clauses shown in Table 1. Notice that we partition the clauses in *required* and *redundant*. The required clauses represent the compact translation using the selected variables. Several studies have shown that adding redundant clauses can improve the solving time of satisfiability solvers [13,14]. There are approximately the same number of redundant clauses as required clauses. We also experienced that the addition of these redundant clauses improved the performance. The encoding in Table 1 can also be used for graphs in general. In that case only the first type of redundant clauses should be omitted.

The size of the encoding can be reduced by using auxiliary Boolean variables. The four types of at-most-one (AMO) constraints (both required and redundant) are encoded using $O(n^3)$ clauses. This can be reduced to $O(n^2)$ in the following way. Instead of directly translating $\text{AMO}(l_1, \dots, l_k)$ to $(\bar{l}_i \vee \bar{l}_j)$ for $1 \leq i < j \leq k$, we apply until fix-point: there is a constraint $\text{AMO}(l_1, \dots, l_k)$ with $k > 4$, replace it by $\text{AMO}(l_1, l_2, l_3, a) \wedge \text{AMO}(\bar{a}, l_4, \dots, l_k)$ using auxiliary variable a . This method reduces the size of the translation significantly. Yet the size of the whole encoding remains $O(n^3)$.

Edge-label encoding. As an alternative for the vertex-edge translation discussed above, an edge-label translation was constructed which is inspired by the CSP model for graceful trees introduced by [15]. This encoding uses the same x variables used in the vertex-edge encoding, but replaces the y variables, by Boolean variables $z_{i,j}$ for $i \in \{0, \dots, i\}$, $j \in \{1, \dots, n - i\}$. If $z_{i,j}$ is set to true, it means that there is an edge $\langle v, w \rangle \in E$ such that vertex v (or w) has label i and vertex w (or v) has label $i + j$. In other words, $z_{i,j}$ to true means that the edge with label j is attached to the vertex with label i .

Symmetry breaking clauses. The symmetries between solutions are broken in the same way for both proposed translations. As discussed in the last section we break the value symmetry by adding the constraint $L_v = 0 \Rightarrow L_w \neq n - 1$. The clause representation of this constraint is $(\bar{x}_{v,0} \vee \bar{x}_{w,n-1})$. The structure symmetries are broken with constraints of the type $L_v < L_w$. This is translated as a conjunction of $(\bar{x}_{v,i} \vee \bar{x}_{w,j})$ for $0 \leq j \leq i \leq n$.

We exploit the internal symmetries as follows. In case a vertex v is forced to have label 0, a unit clause $(x_{v,0})$ is added. Forcing the edge order is far more tricky and we need a different approach for each of the translations. Recall that the order was forced using constraints of the type $L_{\langle u,v \rangle} < L_{\langle v,w \rangle}$. First consider the vertex-edge translation. Here, we can add a conjunction of $(\bar{y}_{\langle u,v \rangle, i} \vee \bar{y}_{\langle v,w \rangle, j})$ for $1 \leq j \leq i \leq n$.

Due to the absence of y variables we cannot add these clauses to the edge-label translation. To this encoding we added the following clauses. For $0 \leq i \leq n$, $(\bar{x}_{v,i} \vee \bar{x}_{w,i+j} \vee \bar{z}_{i,k})$ with $1 \leq j < k \leq n - i$ and $(\bar{x}_{v,i} \vee \bar{x}_{w,i+j} \vee \bar{z}_{i-k,k})$ with $1 \leq j \leq n - i$, $j < k \leq i$ and $(\bar{x}_{v,i} \vee \bar{x}_{w,i-j} \vee \bar{z}_{i,k})$ with $1 \leq j \leq i$, $j < k \leq n - i$ and $(\bar{x}_{v,i} \vee \bar{x}_{w,i-j} \vee \bar{z}_{i-k,k})$ with $1 \leq j < k \leq i$. Notice that the number of additional clauses is $O(n^4)$, thus larger than the original translation. This explains why forcing this symmetry is not useful in practice for this encoding.

Table 2: Edge-label encoding

Required clauses	Range	Meaning
$(x_{v,0} \vee \dots \vee x_{v,n})$	$v \in V$	vertex v has at least one label
$(z_{0,j} \vee \dots \vee z_{n-j,j})$	$1 \leq j \leq n$	edge label j is used at least once
$(\bar{x}_{v,i} \vee \bar{x}_{w,i})$	$v, w \in V, v < w, 0 \leq i \leq n$	at most one vertex has label i
$(\bar{x}_{v,i} \vee \bar{x}_{w,i+j} \vee z_{i,j}) \wedge$ $(\bar{x}_{v,i+j} \vee \bar{x}_{w,i} \vee z_{i,j})$	$\langle v, w \rangle \in E,$ $0 \leq i \leq n, 0 \leq j \leq n - i$	if vertex v (or w) has label i and [ht] vertex w (or v) has label $i + j$ then variable $z_{i,j}$ must be true
$(\bar{x}_{v,i} \vee \bar{z}_{i,j} \vee$ $\bigvee_{\langle v,w \rangle \in E} x_{w,i+j})$	$v \in V,$ $0 \leq i \leq n, 0 \leq j \leq n - i$	if vertex v has label i and variable $z_{i,j}$ is true then a vertex w connected to vertex v has label $i + j$
Redundant clauses	Range	Meaning
$(\bigvee_{v \in V} x_{v,i})$	$i \in \{0, \dots, n\}$	at least one vertex has label i
$(\bar{x}_{v,i} \vee \bar{x}_{v,j})$	$v \in V, 0 \leq i < j \leq n$	vertex v has at most one label
$(\bar{z}_{i,j} \vee \bar{z}_{k,j})$	$1 \leq j \leq n, 0 \leq i < k \leq n - j$	edge label j is used at most once

3.2 Experiment Set-up

To conduct the various experiments a main program written in JAVA performed the following steps:

1. Construct a new tree using the NEXTTREE algorithm [16].
2. Compute the experiment specific constraints for this tree.
3. Encode the tree to SAT using one of the two models and the specific constraints, write the result to file.
4. Run one of the solvers on the file and measure time taken to return a result.
5. Verify the result is either unsat or a graceful labelling.
6. If the result was unsat or the desired number of solutions was reached, go to step 1.
7. Else, append the solution as a restriction to the file and go to step 4.

When this program was run on multiple machines or cpu cores each instance skipped a uniform number of trees between solving by looping on step 1. For every tree the number of solutions found and total time taken were recorded in one separate file per instance to be combined and analyzed later. All experiments were performed on 3.2 Ghz dual core machines.

Because the solvers are applied as black box it was required to find the best combination of solver and encoding by benchmarking each option on a number of problem instances. An initial benchmark was run on 250 trees of 14 vertices selected uniformly using the full encodings (both required and redundant clauses), for which the results are shown in Table 3. From this benchmark it is apparent that PICOSAT, MINISAT and MANYSAT performed best, however it should be noted that in this test MANYSAT used both cores. This meant that when running a program instance per core either PICOSAT or MINISAT would be able to solve trees twice as fast as MANYSAT.

Table 3: Average solving time (ms) of 250 trees with 14 vertices.

Additional Constraints	Solver	Vertex/Edge	Edge-label
None	PICOSAT	15	29
	PRECOSAT	44	173
	MINISAT	19	21
	MANYSAT	26	27
	MARCH-HI	469	3708
	MARCH-NN	457	3856
Edge Order + Structure Symmetry breaking	PICOSAT	17	45
	PRECOSAT	32	139
	MINISAT	14	19
	MANYSAT	21	26
	MARCH-HI	92	1569
	MARCH-NN	99	1795

As the initial benchmark was not conclusive a second benchmark was run on 500 trees of 25 vertices selected uniformly using only the fastest solvers. From the results in Table 4 it was observed that PICOSAT scaled better for larger trees, and that the full vertex/edge encoding performed better than the full edge-label encoding. Therefore, all experiments were run on the PICOSAT solver, and the trees were encoded using the vertex/edge encoding.

Table 4: Average solving time (ms) of 500 trees with 25 vertices.

Additional Constraints	Solver	Vertex/Edge	Edge-label
None	PICOSAT	124	369
	MINISAT	197	1130
Edge Order + Struct. Symm. breaking	PICOSAT	119	922
	MINISAT	202	465

4 Results and Exploration of Symmetries

With the method and best combination of solver and encoding described in 3.2 it is possible to look for the possible properties considered in section 2.2. Because ordering the labels requires a starting point from which the order is imposed, we first look at the possibility of assigning specific vertices the label 0, before considering the edge ordering.

4.1 Center / Leaf to 0

The first two proposed internal symmetry constraints, setting a leaf to 0 or setting a center to 0 presented counterexamples for quite small trees. The smallest tree that does not allow setting its central vertex to zero has 6 vertices and is shown in Fig. 3a. The first tree that does not allow a graceful labelling for each of its leaf vertices where the

selected leaf is set to zero has 7 vertices and is shown in Fig. 3b. The fact that these two trees are very similar is not a coincidence. If we apply the value symmetry to the partial labelling of tree Fig. 3b we obtain a situation where the leaf is labelled with 6 and the center is labelled with 0. At this point we obtain the exact same problem as when setting the center to zero in Fig. 3a.

This transformation can be applied whenever a leaf can not be set to zero and still admit a graceful labelling, which implies that we can use the impossibility of setting a leaf to 0 in a tree of size n as proof that its parent can not be set to 0 in the tree of size $n - 1$ obtained by removing the restricted leaf. Or inversely, if we have a tree of size n which contains a vertex that can not be set to 0 and still admit a graceful labelling, it is possible to construct a tree of size $n + 1$ for which a leaf can not be set to 0 by appending a new leaf vertex to the vertex that can not be set to 0. In other words, the function $f_{\text{leaf} \neq 0}(x)$ that computes the number of trees of x vertices that have a leaf that can not be set to 0 is strictly increasing.

In terms of average computational time needed to compute one solution, constraining one of the leaves to 0 proved to increase the time needed by an increasing factor as shown in Table 6. However under this constraint every computed tree did have at least one solution. On the other hand constraining one of the centers to 0 actually decreased the time needed to find one solution on average, despite containing costly unsatisfiable instances for trees where the center could not be set to 0. In the end, neither constraint can be used in general as a result of the counterexamples and the slowdown in case of the leaves.

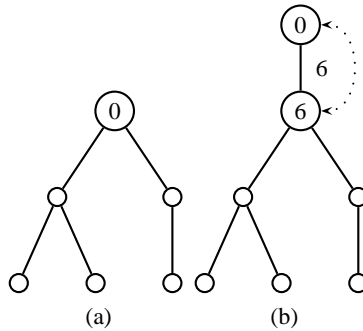


Fig. 3: Partially labelled trees that cannot be extended to a graceful labelling. (a) has its center vertex constrained to 0, (b) has a leaf vertex constrained to 0.

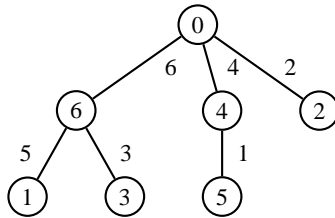


Fig. 4: Example of a graceful labelling where a vertex with the highest degree was given the label 0. By applying value symmetry, the other vertex with a degree of 3 is labelled 0 in a graceful labelling.

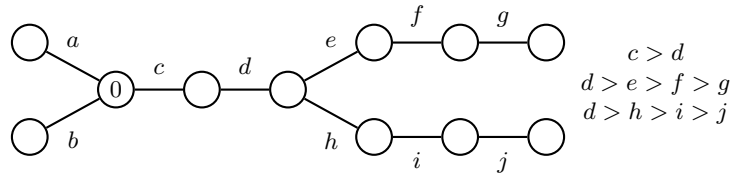


Fig. 5: Tree which does not allow an edge order constraint labelling with the given vertex labelled 0 as a starting point.

4.2 Highest degree to 0

Since setting the center to zero gave much better results than setting a leaf to zero, and since the center of a tree will always have a higher degree than one of the leaves it seemed possible that always selecting the vertex with the highest degree could provide even better results. For this constraint only vertices with the most neighbours are allowed to have label 0. Under this constraint the time to find a graceful labelling was reduced by a factor 5 on average for trees of 15 vertices (Table 6). Further reduction is obtained by applying the tree symmetry as the constraints do not clash (Fig. 4).

Unlike the constraints forcing a leaf or center to 0, forcing a highest degree vertex to have label 0 did not result in unsolvable trees.

Conjecture 2. All trees admit a graceful labelling where any vertex with the highest degree is assigned label 0.

4.3 Strictly decreasing edge labels

An algorithm for finding a Graceful Labelling of a Tree first presented by Horton [12] and later adapted by Fang [10] tries to construct a graceful labelling by appending the highest not yet placed edge label on an edge connected to the cluster of vertices and edges that have been labelled in previous steps. By starting with an initial vertex set to zero, the algorithm constructs a labelling with a strictly decreasing order on edge labels from the initial vertex outward. Horton conjectures that all trees admit such a labelling, and has verified this result up to trees of 29 vertices. Fang later found that by starting with a center set to zero more than 99.99% of all trees of size 30 can be gracefully labelled by this algorithm.

Implementing this property as an additional constraint for the Vertex-Edge encoding of the problem, using only the *required* set of clauses, gave a speed up of a factor 15 for trees of size 15, compared to using only symmetry breaking for symmetries between solutions. This factors increases as the tree size increases. See Table 6 for the full set of results². The experiment has been partially repeated and extended using the complete Vertex-Edge encoding, of which the result can be seen in Table 7.

During the experiments, it became apparent that a single specific class of trees did not always allow a graceful labelling with the the imposed structure. Paths did not always allow its center vertices (which are if a path's highest degree: 2) to be set to 0, if the edge label order was also to be constrained. In fact, paths seem to be the most restrictive

² 35 machines have been used to solve the problem instances, of the same type as those used for the results in Table 3.

when it comes to allowing a vertex with the highest degree to be set to 0, taking into account the edge label ordering constraint based on that vertex. Table 5 shows the vertex positions of the ‘illegal’ vertices for the paths of size 7 to 23. Paths that are not shown allow every vertex to be set to 0.

Table 5: Paths that do not allow all max degree vertices to be set to zero when using the edge order constraint. Marked vertices can not be set to zero, taking into account the edge label ordering.

Number of Vertices	Disallowed vertices
7	
13	
14	
15	
16	
17	
18	
19	
21	
22	

The path of seven vertices proved to be the most restrictive, only allowing the vertex immediately next to the leaf to be labelled zero under these restrictions. Unfortunately, selecting the highest degree vertex furthest from the center is also not possible in general, as the tree in Fig. 5 does not allow such a labelling. On the other hand, paths always have at least one solution where a highest degree vertex is set to zero and the edge labels are in decreasing order. Such a solution can be constructed by following the pattern displayed in Fig. 6.

Therefore, it is possible to ignore paths and instead focus on selecting the correct highest degree vertex as starting point for all other trees. As setting the center to zero proved beneficial to the solver runtime it may be possible to always select the highest degree vertex that is nearest to the center of the tree to be the zero vertex of the decreasing edge order labelling. Implementing this constraint proved to always be possible for all trees up to 20 vertices which leads us to the following conjecture.

Conjecture 3. All trees, except for paths, admit a graceful labelling where any vertex with the highest degree nearest to the center is assigned label 0, and where the edges are labelled in a strictly decreasing order from the vertex with label 0.

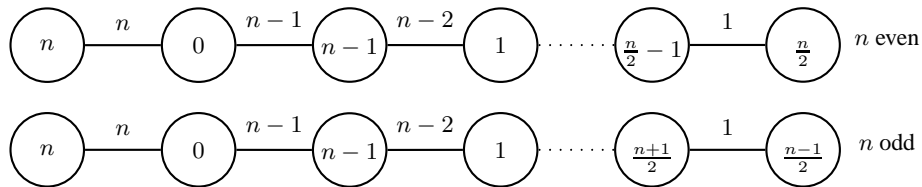


Fig. 6: Construction method for path graceful labelling with highest degree zero and decreasing edge labels.

Table 6: Average runtime (ms) to find one solution under specific constraints, using the vertex-edge encoding with only the *required* clauses. ‘w/o’ indicates the results are from a benchmark *without* structure symmetry breaking, ‘w/’ indicates a benchmark *with* structure symmetry breaking.

Vertices	Trees	Default		Leaf 0	Center 0	Max Degree 0		Edge Order and Max Degree 0
		w/o	w/	w/o	w/o	w/o	w/	w/
4	2	45	18	20	8	9	23	18
5	3	9	20	28	13	39	36	17
6	6	29	23	19	44	29	30	35
7	11	34	27	31	33	45	45	45
8	23	32	29	32	40	31	29	48
9	47	41	47	61	41	35	42	61
10	106	49	46	74	48	44	43	74
11	235	54	41	141	39	38	31	88
12	551	123	71	372	80	65	43	70
13	1,301	297	157	1,158	151	117	69	57
14	3,159	887	367	3,520	494	246	139	53
15	7,741	2,699	834	11,559	1,558	513	273	62
16	19,320	6,539	2,015	–	3,760	1,151	523	77
17	48,629	11,484	4,287	–	5,860	3,651	1,102	97
18	123,867	–	–	–	–	–	–	117
19	317,955	–	–	–	–	–	–	156
20	823,065	–	–	–	–	–	–	221
21	2,144,505	–	–	–	–	–	–	314
22	5,623,756	–	–	–	–	–	–	461
23	14,828,074	–	–	–	–	–	–	675

Table 7: Selective benchmark on 15 machines, using the full vertex-edge encoding. Each machine solved a maximum of 2000 trees for each tree size, evenly spaced along the NEXTTREE index-spectrum.

Vertices	Trees	Default		Leaf 0	Center 0	Max Degree 0		Edge Order and Max Degree 0
		w/o	w/	w/o	w/o	w/o	w/	w/
14	3,159	38	41	40	42	40	38	40
15	7,741	40	43	43	41	40	40	41
16	19,320	47	52	52	54	61	47	49
20	30,000	–	–	–	–	–	141	103
25	30,000	–	–	–	–	–	723	241
30	30,000	–	–	–	–	–	517	671

5 Conclusion

We used satisfiability programming to explore a number of internal symmetries of graceful labellings on trees. In doing so, it was discovered that not every tree allows a graceful labelling where the center or a specific leaf vertex is set to 0. No such counterexamples were found when setting any highest degree node to 0, nor when imposing a strictly decreasing order on the edge labels from any highest degree vertex nearest to the center set to 0 for non-path trees. This lead us to stating two new, stricter Graceful Tree Conjectures. It was also shown that adding these constraints gave a significant reduction in computational time needed to find a solution.

As the vertex selection and edge order constraints appear to be general it could be feasible to implement these constraints into a native algorithm. Such an algorithm will likely be much faster due to not needing the overhead introduced by converting to and from CNF, reading and writing files and from not needing to spawn a new solver process every time. Such an algorithm could then be used in a distributed computing environment to improve the state of the art without needing special attention for false negatives that can occur when using probabilistic algorithms.

Another area of interest is extending mathematical proofs of gracefulness under assumption of Conjecture 3. It might be possible to construct new inductive proofs of gracefulness for classes of trees that have otherwise not yet been proven to always admit a graceful labelling, such as Lobsters.

References

1. Edwards, M., Howard, L.: A survey of graceful trees. *Journal of Mathematics* **1**(1) (2006) 5–30
2. Gallian, J.: A dynamic survey of graph labeling. *The Electronic Journal of Combinatorics* **5** (2009) 1–43
3. Alfalayleh, M., Brankovic, L., Giggins, H., Islam, M.: Towards the Graceful Tree Conjecture: A Survey. *Proceedings of AWOCA (2004)* 239–247
4. Bloom, G., Golomb, S.: Applications of numbered undirected graphs. *Proceedings of the IEEE* **65**(4) (1977) 562–570
5. Basak, A.: MPLS Multicasting Using Caterpillars and a Graceful Labelling Scheme. In: *Proceedings of the Information Visualisation, IEEE Computer Society (2004)* 387
6. Heule, M., Walsh, T.: Symmetry within Solutions. (2010) Accepted for AAAI 2010.
7. Krishnaa, A.: A study of the major graph labelings of trees. *Informatica* **15**(4) (2004) 515–524
8. Gilbert, J.: A complete proof of the Graceful Tree Conjecture using the concept of Edge Degree. *Arxiv preprint arXiv:0709.2201* (2007)
9. Aldred, R., McKay, B.: Graceful and harmonious labellings of trees. *Bull. Inst. Combin. Appl* **23** (1998) 69–72
10. Fang, W.: A Computational Approach to the Graceful Tree Conjecture. *Arxiv preprint arXiv:1003.3045* (2010)
11. Eshghi, K., Azimi, P.: Applications of mathematical programming in graceful labeling of graphs. *Journal of Applied Mathematics* **2004**(1) (2004) 1–8
12. Horton, M.: *Graceful Trees: Statistics and Algorithms*. (2003)
13. Heule, M.J.: Solving edge-matching problems with satisfiability solvers. In: *Proceedings of Logic and Search (LaSh 2008), University of Leuven (2008)* 88–102

14. Heule, M.J.H., Verwer, S.: Using a satisfiability solver to identify deterministic finite state automata. In: BNAIC 2009, BNAIC (2009) 91–98
15. Smith, B.: Constraint programming models for graceful graphs. Principles and Practice of Constraint Programming-CP 2006 (2006) 545–559
16. Wright, R., Richmond, B., Odlyzko, A., McKay, B.: Constant time generation of free trees. SIAM Journal on Computing **15** (1986) 540