

Automated Bounds on Recursive Structures

Wayne Goddard

School of Computing, Clemson University

goddard@cs.clemson.edu

Abstract

Tables are the basis for many dynamic programming algorithms on recursive structures such as trees and grids. These tables record the rules for combining structures. We show how such a table can be used to determine the maximum or minimum value of the associated parameter (the extremal function), and implement software that automatically determines the extremal function from the table, and outputs a proof thereof. We also produce software that automatically generates a candidate table from a black-box oracle for the parameter. We use the software to solve some old and new problems; for example, we obtain the values of domination parameters for some small grids.

1 Overview

Recursive structures such as trees and grids lend themselves to dynamic programming. Indeed, many algorithms to calculate parameters on specific graph families utilize this technique. We consider here parameters that can be calculated on recursive structures using “Wimer” tables (see below). To simplify the presentation we will assume that the recursive structures in question are from some graph family, such as trees or series-parallel graphs, but the ideas apply to any recursive structures that admit Wimer tables.

The main idea is simple: that the table can be used to calculate the extremal function of the associated parameter. That is, one can derive an expression that gives the minimum or maximum value of the parameter over all graphs of order n in the family. Indeed, we show that the ideas can be automated so that the expression, as well as a proof of the expression, can be generated by computer. We also introduce a new way to generate the tables. The focus of the research is on providing actual software, though as proof-of-principle we use the software to prove several new results.

1.1 Tables

In order to make the ideas concrete, here is an example table. In the original form, Wimer et al. [11] envisaged a table as based on a partition of the set \mathcal{T} of rooted marked trees, where a

marked tree is a tree where some subset of the vertices are marked. The table records the rules for combining trees: which class the combined tree lies in. The combination operation is to take two rooted trees and add an edge between the roots of the trees, making the root of the parent subtree the overall root (see Figure 1). Some of the classes of the partition are designated as *valid*.

EXAMPLE. [11] A **dominating set** in a graph is a set of vertices such that every vertex not in the set is adjacent to a vertex of the set. In the table, class 1 is a set dominating all but the root, class 2 is a dominating set that includes the root, and class 3 is a dominating set that doesn't include the root, and class 0 is all other sets. Classes 2 and 3 are the valid ones.

		<i>child</i>		
		1	2	3
<i>parent</i>	1	–	3	1
2	2	2	2	2
3	–	3	3	3

The notation – means that the combination is not valid.

The ideas seem to have been introduced by Wimer et al. [11]. One can use the table to determine the minimum or maximum cardinality of a valid set. That is, once one has the table, one has a linear-time algorithm to compute the value for a graph, provided the recursive decomposition of the graph is known or can also be computed in linear time.

Later Bern et al. [1] showed how to obtain one table from another. Borie et al. [3] showed how a table can be generated from a description of the parameter, and since then, the main line of research has been on showing that, given a description of the sets in some form of logic (such as monadic second order), then one can, at least in theory, directly produce both the table and an algorithm. See, for example, Hagerup [7].

1.2 From Tables to Bounds: Our Results

Now, we observe that the above table actually contains all the information about dominating sets. We need only add the information about the base cases (that the single-vertex tree with unmarked vertex is in class 1 and with marked vertex is in class 2), and which classes represent valid sets (classes 2 and 3). Thus, *the table can be used to answer other questions about the domination number*.

In particular, one can deduce from the table the minimum or maximum value of the parameter on the graphs of a given order. We call this the **extremal function** for the parameter. And one can therefore hope to automate the process. We produce:

Artifact 1 *Software that, on input a table, determines the extremal function for the associated parameter, and produces a full proof of this.*

That is, the software provides the exact expression for the extremal function (the maximum or minimum value of the parameter for a given number of vertices). Further, we obtain a computer proof of correctness of this expression—both of the bound and of the sharpness thereof—which can, in principle, be checked by hand. The software uses ideas from combinatorics and artificial intelligence, as well as some computer algebra.

As evidence of the utility of the software, we apply it to some old and new problems. Using it we determine the minimum value of a new parameter (called the triple response number) in a tree. We also apply this methodology to grids and extend results about some parameters in grids of small height. It is to be noted that we do not have a “by-hand” proof of the larger bounds mentioned above. Also, the application of dynamic programming in grids is not new; see, for example, Livingston and Stout [9].

Along the way to automating the process, we wondered if there were other ways to produce the table without going through a logic formulation. Specifically, given a black-box oracle that tells one whether a set is valid or not, is it possible to induce the table?

The idea is simple: in principle the table can be obtained from examining the infinite matrix of combinations and identifying those rows and columns that are identical (the corresponding graphs behave identically under arbitrary combining). So we implemented software that will induce the partitions and corresponding table, though this table comes without a proof of correctness:

Artifact 2 *Software that, on input an oracle that tests a set for validity, produces a candidate table for that set property.*

Note that the user supplies only the boolean method for validity of marked tree, and an upper bound on the size of the trees to consider. The program generates the classes, calculates the table, produces MetaPost for the representative trees, and L^AT_EX for the table. (See Figure 3 for sample output.) The actual table is guaranteed to be a refinement of this table.

We determined the table for the new parameter (triple response number) mentioned above, and for some of the grid problems mentioned there too. The proof that these tables are complete is not computer generated. However, given some properties of the valid set (for example, that the property is the union of local properties, as it is in domination), it should be possible to produce proofs (though this is not yet implemented).

1.3 Implementation and Example Recursive Structures

The goal is to have actual working software. So we implemented these ideas for two graph families: trees and grids.

- *Trees.* These are the simplest. We consider the *join* operation of taking the disjoint union of two rooted trees and adding an edge between the roots and making one of these roots the root of the overall tree. We speak of the *parent* and *child* subtrees—the root of the parent subtree becomes the overall root.

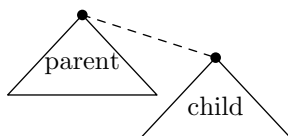


Figure 1: The join of two trees

- *Grids of fixed height.* This is slightly different in that the combining operation is defined only one way: in our case, the combination operation is to form an $h \times (n + 1)$ grid by combining an $h \times n$ parent grid with a child “grid” consisting of only one column. The results are then for the family \mathcal{G}_h of all grids with h rows.

To generate tables for other recursive structures, all that is needed is to provide a list (or generator) of all such structures up to the desired order, and a method that takes two structures and returns the combined structure.

The software is implemented in Java with no attempt (as yet) of efficiency. All the values given in the paper take at most a few minutes on a desktop PC.

2 Automated Bounds

In this section we outline the method of automatically generating sharp bounds from the table.

We present this section in terms of *trees*. The same ideas can be used for other recursive structures. For example, the same software also works for grids of fixed height. We present the discussion in terms of *markings*—a subset of the vertices are marked—and there is some definition of *valid* set under interest. This idea can easily be generalized to multiple sets or colorings, although with some slow down.

The key idea is that the table defines a vector for each rooted (unmarked) tree. Indeed, that is precisely what the original associated algorithm calculates. Then one can ask: what does the set of all such vectors look like?

2.1 Input

We start with specifying the input. First one is given:

- A partition $\mathcal{P} = \{P_0, P_1, \dots, P_k\}$ of the set \mathcal{T} of rooted marked trees where some classes of \mathcal{P} are designated valid, and a table which specifies the classes of combinations. We let π_ℓ be the set of pairs (i, j) such that combining a parent tree of class P_i with a child tree of class P_j produces a tree of class P_ℓ .

In our examples, the class P_0 will always be invalid; indeed it will be the *impossible class*: the markings that no valid tree can contain. (For example, suppose the set of interest is an independent set: all trees with two adjacent marked vertices are in P_0 .) Further we do not specifically list P_0 in the table, as any tree combined with a tree from P_0 is in P_0 .

Now, one is also given a parameter based on the collection of valid trees of \mathcal{P} . Specifically one is given:

- A *function* \oplus which says how the weights of two markings should be combined. In all our examples this is simply addition, since we are looking at the cardinality of the marked set. But one can envisage other functions. One is also given an *operator* Ψ which is either minimum or maximum, depending on whether the parameter f is the minimum or maximum cardinality of a valid set.

Finally, one is given:

- An *operator* Θ which says whether we want the minimum or maximum value of the parameter f .

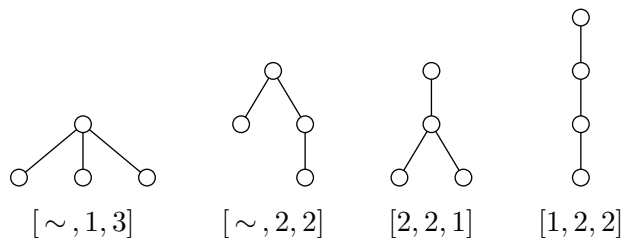
2.2 $\mathcal{E}(T)$ for a tree T

For a given unmarked tree T , we define the k -tuple $\mathcal{E}(T)$:

the vector $\mathcal{E}(T) = [e_1, \dots, e_k]$ is defined as follows. For each class P_i of \mathcal{P} , examine all such markings and determine the best number for that class, where best is determined by the choice of Ψ .

For example, for the example of domination, e_1 is the *minimum* number of vertices in a set that dominates all vertices except the root. It may happen that there is no such marking in a particular class: if so, we write \sim in the vector.

EXAMPLE. There are four rooted (unmarked) trees of order 4. For domination these have vectors as follows:



Now, this vector $\mathcal{E}(T)$ can be calculated recursively from the vectors of the parent and child trees (T_P and T_C). Specifically, if $T = \text{join}(T_P, T_C)$ with $\mathcal{E}(T_P) = \vec{x}$ and $\mathcal{E}(T_C) = \vec{y}$, then

$$\mathcal{E}(T) = \text{join}(\vec{x}, \vec{y}) := [\Psi_{(i,j) \in \pi_\ell} x_i \oplus y_j]_{\ell=1}^k. \quad (1)$$

In this we mean that if either value is \sim then $x_i \oplus y_j = \sim$. Further, values of \sim are ignored when calculating Ψ of a set; if the set is empty or has only \sim values, then $\Psi = \sim$.

At the end, the value of the parameter f is

$$f(T) = \Psi_{i \in \pi} e_i,$$

where $\mathcal{E}(T) = [e_i]$ and π is the set of classes which correspond to a valid marking. (For our example of domination, $\pi = \{2, 3\}$.)

2.3 Sets of vectors

We next consider the set of all vectors for a given order:

$$\mathcal{E}(n) = \{ \mathcal{E}(T) : T \text{ has } n \text{ vertices} \}.$$

Note that $|\mathcal{E}(n)| \leq (n+1)^k$.

Now, every tree on at least two vertices is the join of two smaller trees. By Equation 1, the vector of a tree is the *join* of the vectors of the smaller trees. Thus, the set $\mathcal{E}(n)$ can be determined recursively:

Lemma 1 For $n \geq 2$

$$\mathcal{E}(n) = \{ \text{join}(\vec{x}, \vec{y}) : 1 \leq p < n, \vec{x} \in \mathcal{E}(p), \vec{y} \in \mathcal{E}(n-p) \}.$$

To get it all started, we need to know $\mathcal{E}(1) = \{\mathcal{E}(K_1)\}$. If the parameter of the set is the cardinality thereof, $\mathcal{E}(K_1)$ is $[0, 1, \sim, \dots, \sim]$ for some permutation of the class numbers (where \sim means that there is no valid marking in this class).

Now let $m_f(n)$ denote the **extremal function**: that is, the Θ -value of the function f over all trees of order n . The extremal function can be calculated from $\mathcal{E}(n)$:

$$m_f(n) = \Theta \{ \Psi_{i \in \pi}(z_i) : \vec{z} = (z_i) \in \mathcal{E}(n) \}.$$

2.4 Extremal vectors

It turns out, however, that one does not actually need to calculate all of $\mathcal{E}(n)$. We define a partial ordering \preceq (loses-to) on the vectors of $\mathcal{E}(n)$. Specifically, for vectors \vec{x} and \vec{y} we define $\vec{x} \preceq \vec{y}$ if $x_i \prec y_i$ for all $i \in \{1, \dots, k\}$, where \prec is a total order defined for entries as follows.

Note that each entry is either a nonnegative integer or \sim . We define $x_i \prec y_i$ as follows:

- Suppose $\Theta = \Psi = \max$. Then $x_i \prec y_i$ if either $x_i = \sim$, or if x_i and y_i are both integers and $x_i \leq y_i$.
- Suppose $\Theta = \max$ and $\Psi = \min$. Then $x_i \prec y_i$ if either $y_i = \sim$, or if x_i and y_i are both integers and $x_i \leq y_i$.
- The cases with $\Theta = \min$ are similar.

The idea in the first case is that in calculating the maximum, the larger the better, and something is better than nothing. The idea in the second case is that again the larger the better, but in calculating the join (which uses Ψ), nothing is better than something.

Lemma 2

(a) If $x_i \prec y_i$ and $x_j \prec y_j$, then $x_i \oplus y_i \prec x_j \oplus y_j$.

(b) If $\vec{x} = [x_i] \preceq \vec{y} = [y_i]$, then $\Psi_{i \in \pi} x_i \prec \Psi_{i \in \pi} y_i$.

(c) If $\vec{x}_1 \preceq \vec{x}_2$ and $\vec{y}_1 \preceq \vec{y}_2$, then $\text{join}(\vec{x}_1, \vec{y}_1) \preceq \text{join}(\vec{x}_2, \vec{y}_2)$.

PROOF. (a) Trivial, assuming \oplus is order-preserving.

(b) The result is clear if \vec{x} and \vec{y} have \sim in the same entries. But it is not that much harder to check the claim in general.

(c) This follows from (a) and a generalization of (b). QED

We can now define the set that we are interested in:

Define $\tilde{\mathcal{E}}(n)$ as the \preceq -maximal elements of the set $\mathcal{E}(n)$.

EXAMPLE. Consider the domination number of a tree. If we want the maximum domination number, then $\tilde{\mathcal{E}}(4) = \{[\sim, 1, 3], [\sim, 2, 2]\}$. If we want the minimum domination number, then $\tilde{\mathcal{E}}(4) = \{[\sim, 1, 3], [2, 2, 1], [1, 2, 2]\}$.

It turns out that one can replace $\mathcal{E}(n)$ by $\tilde{\mathcal{E}}(n)$ throughout:

Theorem 1

(a) If $f(T)$ exists for all trees T , then $m_f(n) = \Theta \left\{ \Psi_{i \in \pi}(z_i) : \vec{z} = (z_i) \in \tilde{\mathcal{E}}(n) \right\}$.

(b) $\tilde{\mathcal{E}}(n)$ is the \preceq -maximal elements in $\{ \text{join}(\vec{x}, \vec{y}) : 1 \leq p < n, \vec{x} \in \tilde{\mathcal{E}}(p), \vec{y} \in \tilde{\mathcal{E}}(n-p) \}$.

PROOF. (a) By Lemma 2b, if $\mathcal{E}(T) \preceq \mathcal{E}(T')$, then $f(T) \prec f(T')$. It follows that $m_f(n)$ is achieved for each n by some tree T_n such that $\mathcal{E}(T_n)$ is \preceq -maximal.

(b) Consider any tree T with $\mathcal{E}(T) \in \tilde{\mathcal{E}}(n)$. Then T is the join of two trees, say T_P and T_C . By Lemma 2c, if $\vec{x}_1 \preceq \vec{x}_2$ and $\vec{y}_1 \preceq \vec{y}_2$, then $\text{join}(\vec{x}_1, \vec{y}_1) \preceq \text{join}(\vec{x}_2, \vec{y}_2)$. It follows that the vector $\mathcal{E}(T)$ is achieved by the join of two trees such that $\mathcal{E}(T_P)$ and $\mathcal{E}(T_C)$ are \preceq -maximal. QED

2.5 The general form of $\tilde{\mathcal{E}}(n)$

So the strategy is to calculate $\tilde{\mathcal{E}}(n)$ until one guesses at a general form for it. Note that the set is computable in polynomial-time. Once one discerns the form for $\tilde{\mathcal{E}}(n)$, one then hopes to prove by induction that it has that general form. This gives a proof of the value of $m_f(n)$.

The simplest case is that $\tilde{\mathcal{E}}(n)$ settles down to a finite set. If d is a positive integer, we say that $\tilde{\mathcal{E}}(j)$ is obtained from $\tilde{\mathcal{E}}(i)$ by a **simple shift by d** , if the first set is obtained from the second by adding d to some entries. More precisely, there is a bijection ϕ from $\tilde{\mathcal{E}}(i)$ to $\tilde{\mathcal{E}}(j)$, and 0–1 vectors $\vec{a}_{\vec{x}}$ for each $\vec{x} \in \tilde{\mathcal{E}}(i)$, such that the vectors \vec{x} and $\phi(\vec{x})$ have \sim in the same entries, and $\phi(\vec{x}) = \vec{x} + d \cdot \vec{a}_{\vec{x}}$.

EXAMPLE. For the minimum domination number, $\tilde{\mathcal{E}}(4) = \{[\sim, 1, 3], [2, 2, 1], [1, 2, 2]\}$ and $\tilde{\mathcal{E}}(5) = \{[\sim, 1, 4], [3, 2, 1], [1, 2, 2]\}$. So they are related by a simple shift by 1. The sets $\tilde{\mathcal{E}}(6)$ and $\tilde{\mathcal{E}}(5)$ are related by the same simple shift.

Thus one proceeds until the following situation arises: for some integers R and C , the set $\tilde{\mathcal{E}}(C + R + i)$ is a simple shift of $\tilde{\mathcal{E}}(C + i)$ for all i with $0 \leq i \leq R - 1$. This suggests the hypothesis that the pattern continues. That is, the hypothesis is special small cases for $n < C$ and a general form for $\tilde{\mathcal{E}}(n)$ for $n \geq C$, where the expression depends on n modulo R .

One then needs to verify that the general expression is correct. We use a proof by induction. (However, we conjecture that it is sufficient to just check that the pattern holds for n large enough.) Verifying the inductive step entails looking at all combinations of parent and child: one needs to consider each of parent small, child general form; child small, parent general form; and both parent and child general form.

EXAMPLE. Consider determining the minimum domination number of a tree of order n . (Here $m_f(n) = 1$, with the extremal tree being a star.) The extremal sets can be calculated as:

$$\begin{aligned} \tilde{\mathcal{E}}(1) &= \{[0, 1, \sim]\}, \quad \tilde{\mathcal{E}}(2) = \{[\sim, 1, 1]\}, \quad \tilde{\mathcal{E}}(3) = \{[\sim, 1, 2], [1, 2, 1]\}, \text{ and} \\ \tilde{\mathcal{E}}(n) &= \{[\sim, 1, n-1], [n-2, 2, 1], [1, 2, 2]\} \quad \text{for } n \geq 4. \end{aligned}$$

(The trees giving the vectors in $\tilde{\mathcal{E}}(n)$ are the star rooted at the center, the star rooted at an end-vertex, and the star with one edge subdivided rooted at the end-vertex whose edge was subdivided.)

On the other hand, if we are looking for the maximum value of the domination number, then the sets $\tilde{\mathcal{E}}(n)$ do not settle down to a simple shift.

2.6 Automation

In general, we verify the inductive step by computer. This verification requires some computer algebra. For instance, in the above example, combining the general form $[\sim, 1, p-1]$ with the general form $[n-p-2, 2, 1]$ yields the vector

$$\alpha = [\sim, \min(n-p-1, 3, 2), \min(p+1, p)],$$

which simplifies to $[\sim, 2, p]$, since $n-p \geq 4$.

In general, one needs to show that the vector α is in the *envelope* given by the claimed $\tilde{\mathcal{E}}(n)$. One way to do this is to haul out *linear programming*, and show that there is some β (in this case the vector $[1, 2, 2]$) in the claimed $\tilde{\mathcal{E}}(n)$ such that $\alpha \preceq \beta$. At the same time, one needs to verify that every point in the envelope is in fact achievable. So we have to see whether there is an *integral solution* of $\alpha = \beta$.

3 Example: Maximum Packing Number in Trees

We present first a simple example of the process by reproducing a well-known bound. A *packing* in a graph G is a set S of vertices such that the distance between any two vertices

		child			
		1	2	3	
parent	○ 1	1	3	1	valid
	● 2	2	–	–	valid
	○ ● 3	3	–	3	valid

Figure 2: The table for packing

is at least 3. The maximum cardinality of a packing is denoted $\rho(G)$. Figure 2 gives the table for packing—for each class an example tree is drawn.

Suppose we wanted to know the *maximum value of the packing number of a tree* as a function of the order.

Lemma 3 *The set $\tilde{\mathcal{E}}(n)$ of the \preceq -maximal vectors for packing is given by:*

$\tilde{\mathcal{E}}(1) = \{ [0, 1, \sim] \}$, $\tilde{\mathcal{E}}(2) = \{ [0, 1, 1] \}$, and $\tilde{\mathcal{E}}(3) = \{ [1, 1, 1] \}$;

For $n \geq 4$ and $n \bmod 2 = 0$: $\tilde{\mathcal{E}}(n) = \{ [(n-2)/2, (n-2)/2, n/2], [(n-2)/2, n/2, (n-2)/2] \}$;
and

For $n \geq 5$ and $n \bmod 2 = 1$: $\tilde{\mathcal{E}}(n) = \{ [(n-3)/2, (n-1)/2, (n-1)/2], [(n-1)/2, (n-3)/2, (n-1)/2], [(n-1)/2, (n-1)/2, (n-3)/2] \}$.

PROOF. The proof assumes the table is correct. The data and the proof were generated by computer: the input was the table for packing, the fact that packing number is the *maximum* of a packing and the fact that we wanted the *maximum* value of the packing number. The proof is by induction on n .

In principle, one can verify the proof by hand. It is not hard to do this for the base cases. For the inductive step it is the same idea, just more tedious. For example, consider verifying the inductive step for n even and at least 8. A tree of even order $n \geq 8$ can be obtained by combining two trees of sizes (a) 1 and $n-1$; (b) 2 and $n-2$; (c) 3 and $n-3$; (d) p and $n-p$ where $p, n-p$ are even and at least 4; or (e) p and $n-p$ where $p, n-p$ are odd and at least 5. So one looks at all combinations and vectors, and calculates the resultant vector in each case. For example, in case (e) there are $3^2 = 9$ vector combinations, one of which is:

$$\text{join} \left(\left[\frac{p-3}{2}, \frac{p-1}{2}, \frac{p-1}{2} \right], \left[\frac{n-p-1}{2}, \frac{n-p-3}{2}, \frac{n-p-1}{2} \right] \right) = \left[\frac{n-4}{2}, \frac{n-2}{2}, \frac{n-2}{2} \right].$$

One then determines the \preceq -maximal vectors and checks that the resultant set is what is claimed. QED

From this lemma follows the well-known bound:

Theorem 2 *The maximum value of $\rho(T)$ taken over all trees T of order $n \geq 2$ is given by $\lfloor n/2 \rfloor$.*

PROOF. We need only calculate the parameter $\rho(T)$ for each vector in $\tilde{\mathcal{E}}(n)$ —it is the maximum entry of the vector—and then take the maximum over all vectors. QED

4 Producing the Table

We now consider the process of inducting the table. In principle, one can produce the table using only an oracle for testing whether the marked tree is valid or not. The idea is to define some canonical numbering of the marked trees, and consider the infinite matrix $\mathcal{M} = (m_{ij})$ where m_{ij} says whether combining the i th tree with the j th tree is valid or not. Two marked trees are equivalent iff they have the same rows and columns in \mathcal{M} .

In order to produce a practical program, we simply stop the process at some point. That is, *we consider a finite submatrix of \mathcal{M}* . In particular, we find the equivalence relation on the trees up to some order. This easily, for example, produces the table for independence or domination.

Of course, some behavior might only manifest itself as trees get larger and the first example of a class might be very large. One way to improve on this process is to iterate it. First run the above process to produce an initial partition. Then using this classification, produce another matrix, and run the process again. This idea was needed for the example of triple response number discussed below. Another idea would be to explore only combining the representative trees.

At the end of the process, the software produces a partition and a candidate table. The actual partition and table are a refinement of these.

Note that one cannot be sure of the table without some knowledge of the behavior of the black box—we can never discover whether the behavior suddenly changes at order just over the horizon.

5 Case Study: Triple Response Number in Trees

Consider the following graph parameter. Consider a graph $G = (V, E)$ and two subsets A and R of V with $|R| \geq |A|$, not necessarily disjoint. We say that R *can respond to* A if







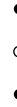

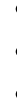




														
	1	2	3	4	5	6	7	8	9	10	11	12	13	
1	-	3	-	-	6	-	-	-	11	-	1	3	6	not
2	4	5	7	8	9	2	5	9	9	5	2	5	9	valid
3	-	6	-	-	11	-	-	-	11	-	3	6	11	not
4	-	10	-	-	12	4	10	12	12	10	4	10	12	not
5	10	9	13	13	9	5	9	9	9	9	5	9	9	valid
6	-	11	-	-	11	-	-	-	11	-	6	11	11	not
7	-	13	7	8	13	7	13	13	13	13	7	13	13	not
8	-	13	8	13	13	8	13	13	13	13	8	13	13	not
9	12	9	13	13	9	9	9	9	9	9	9	9	9	valid
10	-	12	-	-	12	10	12	12	12	12	10	12	12	not
11	-	11	-	-	11	-	-	-	11	-	11	11	11	valid
12	-	12	-	-	12	12	12	12	12	12	12	12	12	valid
13	-	13	13	13	13	13	13	13	13	13	13	13	13	valid

Figure 3: The Triple Response Table

there is an injection ϕ from A to R such that for each $a \in A$, $\phi(a) \in N[a]$. That is, if A is a set of simultaneous attacks, and guards are stationed at R , then one can assign one guard to each attack such that each attack is at the guard or at a neighbor of the guard. Define the parameter $R_3(G)$ as the minimum number of guards such that they can respond to any set of 3 attacks. The 1-attack case is simply the domination number. This parameter was introduced in [2].

We wanted to know the minimum value of $R_3(T)$ over all trees T of order n . Step One is to create the table for trees. The software produces a 13-class table, shown in Figure 3. Note that this is completely automated. All the user supplies is (a) the boolean method that for marked tree says whether guards placement is valid or not, and (b) an upper bound on the size of the tree to consider. The program generates the classes, calculates the table, produces MetaPost for the representative trees, and L^AT_EX for the table.

We have to rely on external verification of the table. In [2] the table and algorithm for R_k in general is given, together with proofs of the validity of the table. (It turns out that the class of the marked tree is determined by its ability to handle fewer attacks and so whether it can give or needs support from the rest of the tree.) Nevertheless, there is some hope that automation might at least assist the verification process.

Now, to determine the minimum value of R_3 for trees, the algorithm proceeds as follows. The software needs as input (a) the table, (b) what classes are in fact valid markings, (c) whether the parameter is a minimize or maximize, (d) whether we are interested in the minimum or maximum value of the parameter, and (e) the value of $\mathcal{E}(K_1)$.

The software produces as output the minimum value of R_3 :

Theorem 3 *The minimum value of R_3 taken over all trees of order $n \geq 4$ is given by*

$$\begin{aligned} (n+2)/2 & \text{ if } n \equiv 0 \pmod{2} \\ (n+1)/2 & \text{ if } n \equiv 1 \pmod{2}. \end{aligned}$$

The software provides a **proof** of this fact that is, in principle, human checkable. This proof gives the form for $\tilde{\mathcal{E}}(n)$: the claim is that the set $\tilde{\mathcal{E}}(n)$ is given by special cases for $n \leq 42$ and a general form for $n \geq 43$. The general form for $\tilde{\mathcal{E}}(n)$ is given by the list given in Figure 4 (by adding $\lceil n/2 \rceil$ to every entry in every vector which is not \sim). Since $\tilde{\mathcal{E}}(n)$ does not become periodic until $n = 43$, and for n odd has 54 elements, implementing this approach by hand is probably impossible.

Furthermore, the software can assist in producing an extremal tree that meets the bound, for a specific value of n . Research is ongoing to automatically generate the general form of extremal trees.

6 Case Study: Generalized Domination in Grids

A lot has been written about domination in (complete) grid graphs. Such results are laborious by hand. For example, the paper by Chang et al. [4] is devoted to the computation of the (ordinary) domination number for the $5 \times n$ and $6 \times n$ grids, while the paper [10] is devoted to the computation of the paired domination number (defined below) for the $2 \times n$, $3 \times n$ and $4 \times n$ grids. Livingston and Stout [9] showed how the ideas of finite automata can be used to compute the values of parameters in grids of fixed height, while Fisher [5] also used dynamic programming to actually calculate the domination of grids of height up to 21.

We show that there is generic software for such investigations, and determine the actual values for several parameters.

We say that a set is (a, b) -dominating if every vertex in the set has at least a neighbors in the set, and every vertex outside the set has at least b neighbors in the set. Ordinary domination is the case $a = 0$ and $b = 1$; **total domination** the case $a = b = 1$; **2-domination** the case $a = 0$ and $b = 2$; and **double domination** the case $a = 1$ and $b = 2$.

1 2 3 4 5 6 7 8 9 A B C D	min over pi	1 2 3 4 5 6 7 8 9 A B C D	min over pi
~.~.~.~.~.~.0.0.2.~.2.3.0.	> 0	~.~.~.~.~.~.1.1.1.2.~.1.3.1.	> 1
~.~.~.~.~.~.0.0.3.~.2.2.0.	> 0	~.~.~.~.~.~.1.1.1.2.~.2.2.2.	> 2
~.~.~.~.~.~.1.0.0.2.~.1.1.1.	> 1	~.~.~.1.1.2.1.~.~.2.1.1.1.1.	> 1
~.~.~.~.~.~.1.1.0.2.~.1.~.0.	> 0	~.~.~.2.0.1.2.~.~.2.2.1.1.3.	> 1
~.~.~.~.~.~.2.0.0.1.~.1.2.1.	> 1	~.1.~.2.1.~.3.3.1.2.2.2.3.	> 1
~.~.~.~.~.~.2.1.0.3.~.2.2.0.	> 0	~.2.~.1.2.~.1.3.2.1.2.1.1.	> 1
~.~.~.~.~.~.3.1.0.2.~.2.3.0.	> 0	~.2.~.1.2.~.2.2.2.1.2.1.2.	> 1
~.~.~.~.0.1.~.~.~.1.0.2.0.3.	> 0	0.1.0.2.1.2.2.2.2.~.1.~.~.	> 1
~.~.0.0.1.0.~.~.~.1.0.1.1.2.	> 1	0.1.1.1.2.1.1.2.2.~.1.~.~.	> 1
~.~.0.0.1.1.~.~.~.2.1.1.1.1.	> 1	1.1.1.1.1.1.1.1.1.2.~.1.~.~.	> 1
~.~.1.0.1.1.~.~.~.1.0.1.1.1.	> 1	1.1.1.1.2.1.1.~.1.~.1.~.~.	> 1
~.~.1.0.1.1.~.~.~.1.1.1.0.1.	> 0	1.2.~.~.2.0.~.~.1.~.2.~.~.	> 1
~.~.1.1.2.~.~.~.1.1.0.0.~.	> 0	1.2.0.~.1.1.2.1.2.~.1.~.~.	> 1
~.~.1.1.2.0.~.~.~.1.0.1.1.1.	> 1	1.2.1.~.1.1.~.~.1.~.2.~.~.	> 1
~.~.1.1.2.1.~.~.~.1.0.1.0.~.	> 0	1.2.1.1.2.1.1.2.2.1.1.2.1.	> 1
~.~.1.1.2.2.~.~.~.1.2.0.0.2.	> 0	1.2.1.2.2.1.2.1.2.2.2.2.2.	> 2
~.~.2.2.3.2.~.~.~.1.0.0.0.~.	> 0	1.2.1.2.2.1.2.2.1.~.2.~.~.	> 1
~.~.3.1.2.3.~.~.~.1.0.1.0.3.	> 0	1.2.1.3.2.1.3.1.2.3.1.3.1.	> 1
~.1.~.1.1.~.1.2.1.1.1.1.1.	> 1	1.2.1.3.2.1.3.3.1.~.1.~.~.	> 1
0.1.0.~.1.0.2.0.1.~.1.~.~.	> 1	1.2.2.~.1.1.~.~.1.~.1.~.~.	> 1
0.1.0.0.1.0.0.2.1.~.1.~.~.	> 1	1.2.2.2.3.0.2.~.1.~.2.~.~.	> 1
0.1.0.0.1.1.1.1.2.~.1.~.~.	> 1	2.1.2.1.2.1.1.2.2.3.1.2.2.	> 1
0.1.0.1.1.0.1.1.1.~.1.~.~.	> 1	2.1.2.2.1.1.2.2.2.2.1.3.3.	> 1
0.1.0.1.1.1.1.1.1.~.0.~.~.	> 0	2.2.2.1.2.2.1.1.2.1.2.2.1.	> 1
0.1.0.1.1.1.1.2.2.1.1.2.1.	> 1	2.2.2.1.2.2.1.3.2.2.2.1.1.	> 1
0.1.0.2.1.0.2.2.1.2.1.3.2.	> 1	2.2.2.1.2.2.2.2.2.2.2.1.2.	> 1
0.1.1.1.2.0.2.~.1.~.0.~.~.	> 0	2.3.1.2.2.1.2.1.2.1.1.2.1.	> 1
0.1.1.3.2.0.3.3.1.~.0.~.~.	> 0	2.3.2.~.1.0.~.~.1.~.3.~.~.	> 1
1.0.1.1.2.1.~.~.~.1.~.0.~.~.	> 0	2.3.2.2.2.1.2.4.2.1.1.1.1.	> 1
1.0.1.1.2.2.2.2.1.~.0.~.~.	> 0	2.3.2.2.2.1.3.3.2.1.1.1.2.	> 1
1.0.2.2.3.0.3.~.1.~.0.~.~.	> 0	2.3.2.2.3.2.2.2.2.2.2.1.1.	> 1
1.1.1.1.1.1.1.1.1.1.1.2.1.	> 1	2.3.2.3.3.0.3.3.1.~.3.~.~.	> 1
1.1.1.1.2.0.1.~.1.~.0.~.~.	> 0	2.3.2.4.3.0.4.4.1.~.2.~.~.	> 1
1.1.1.1.2.1.1.2.1.2.1.1.1.	> 1	2.3.3.~.1.0.~.~.1.~.2.~.~.	> 1
1.2.0.~.1.0.~.2.1.~.0.~.~.	> 0	3.1.3.1.1.2.1.1.2.2.2.2.2.	> 1
1.2.0.2.1.1.2.1.2.1.1.2.1.	> 1	3.1.3.1.2.2.1.~.1.3.2.2.2.	> 1
1.2.0.3.1.0.2.1.1.2.1.3.1.	> 1	3.1.3.1.2.3.1.3.1.3.2.2.2.	> 1
1.2.1.~.2.1.3.1.2.~.1.~.0.	> 0	3.2.2.3.1.2.2.1.2.2.1.3.2.	> 1
1.2.1.1.2.1.0.2.2.1.1.1.1.	> 1	3.2.3.2.2.2.2.2.1.3.2.2.3.	> 1
1.2.1.1.2.1.1.0.2.1.1.1.1.	> 1	3.2.3.2.3.2.2.4.1.4.1.2.3.	> 1
1.2.1.2.1.1.2.3.1.1.2.1.1.	> 1	3.2.3.3.1.2.4.4.1.2.1.2.3.	> 1
1.2.1.2.2.0.2.3.1.2.0.2.2.	> 0	3.4.3.3.4.1.3.3.2.3.1.1.1.	> 1
1.2.1.3.1.0.3.3.1.2.0.2.2.	> 0	4.3.4.3.3.2.3.3.1.4.1.2.3.	> 1
1.2.2.2.1.0.2.3.1.1.1.1.1.	> 1		
2.0.2.2.3.0.2.~.1.~.0.~.~.	> 0		
2.1.2.0.1.1.0.2.1.2.1.1.1.	> 1		
2.1.2.0.1.1.1.1.1.2.2.1.1.2.	> 1		
2.1.2.1.1.2.1.1.1.1.2.1.1.	> 1		
2.1.2.2.1.1.2.0.1.2.1.2.1.	> 1		
2.2.2.2.1.1.2.2.1.1.1.1.1.	> 1		
2.3.1.3.2.1.3.2.1.2.0.3.1.	> 0		
2.3.2.2.3.2.2.1.3.2.2.2.0.	> 0		
3.0.3.1.2.2.2.2.1.3.1.2.3.	> 0		
3.2.3.3.2.2.3.1.2.3.2.3.0.	> 0		

Figure 4: $\tilde{\mathcal{E}}(n)$ for n odd (left) and even (add $\lceil n/2 \rceil$ throughout)

A **paired dominating** set is a dominating set such that the subgraph induced by the set has a perfect matching. For a survey, see the book [8].

We consider grids with a fixed number h of rows and with n columns.

EXAMPLE. Consider double domination of grids of height 2. The table is given in Figure 5. Note that the base case of a column of two unmarked vertices lies in the impossible class, and is therefore not shown.















						
	1	–	–	–	3	not
	2	–	–	–	3	not
	3	4	5	6	3	valid
	4	–	–	–	3	not
	5	–	7	8	3	valid
	6	–	9	10	3	valid
	7	–	–	8	3	not
	8	–	–	10	3	not
	9	–	7	–	3	not
	10	–	9	–	3	not

Figure 5: The table for double domination in grids of height 2

Since there is a unique grid for each n , there is only one vector for each n . Thus the process simplifies to generating the vector until a pattern is spotted.

EXAMPLE. Consider determining the double domination number of grids of height 2. This is $n + 1$. (Easy and probably known.)

The vectors are calculated as: $[1, 1, 2, \sim, \sim, \sim, \sim, \sim, \sim, \sim]$ for $n = 1$,

$[\sim, \sim, 3, 2, 3, 3, \sim, \sim, \sim, \sim]$ for $n = 2$, and

$[\sim, \sim, n + 1, n, n + 1, n + 1, n + 1, n + 1, n + 1, n + 1]$ for $n \geq 3$.

Similar ideas allow one to prove results for larger grids. Indeed, the bottleneck lies in determining the table. So specific software was produced to generate the tables in this case.

Theorem 4 *The values of the parameters for small-height grids are given in Figure 6. The function $\varepsilon(a, b)$ is 1 if $n \equiv a \pmod b$ and 0 otherwise. The values for paired domination are obtained from the value for total domination by rounding up to an even number.*

h	Total Dom.	2-Dom.	Double Dom.
2	$2\lceil n/3 \rceil$	n	$n + 1$
3	n	$\lceil 4n/3 \rceil$	$\lceil (3n + 1)/2 \rceil$
4	$2\lceil (3n + 2)/5 \rceil$	$\lceil (7n + 3)/4 \rceil$	$\lceil (19n + 9)/10 \rceil$
5	$\lceil (3n + 2)/2 \rceil + \varepsilon(0, 4)$	$\lceil (15n + 2)/7 \rceil$	$\lceil (7n + 4)/3 \rceil$
6	$2\lceil (6n + 6)/7 \rceil - 2\varepsilon(4, 7)$	$\lceil (28n + 10)/11 \rceil - \varepsilon(6, 11)$	$\lceil (11n + 5)/4 \rceil - \varepsilon(4, 8)$
7	$2n + 2 - \varepsilon(1, 2)$	$\lceil (53n + 10)/18 \rceil - \varepsilon(7, 18)$	$\lceil (19n + 11)/6 \rceil$
8	$2\lceil (10n + 10)/9 \rceil - 2\varepsilon(1, 9) + 2\varepsilon(7, 9)$	$\lceil (10n + 2)/3 \rceil$	$\lceil (18n + 8)/5 \rceil$

Figure 6: Exact values in $h \times n$ grids for n sufficiently large

In all cases the values for $h \geq 5$ are new. (Small values were given in [10, 6].) These results can easily be extended to any grid-like structure, for example, the cartesian product of a path and a fixed graph.

We omit the details.

7 Extensions and Future Research

In theory, one can try this strategy on any recursive structure. Our algorithm requires only an oracle which (a) generates all small examples, (b) combines two examples, and (c) evaluates the example for the given property. Indeed each of these can be implicit: as long as there is a canonical numbering of the family, operation (b) can be implicit. For example, consider series-parallel graphs, or certain zero-one matrices.

We recently extended the software to handle edge parameters on trees. We are currently working on extending the software further. This includes:

- Automatic generation of the general form of extremal graphs.

- Improvements in time complexity.
- Improved handling of envelope considerations.

References

- [1] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8(2):216–235, 1987.
- [2] J. Blair, W. Goddard, S.M. Hedetniemi, S.T. Hedetniemi, F. Manne, and D. Rall. k -response sets. Submitted.
- [3] R.B. Borie, R.G. Parker, and C.A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(5-6):555–581, 1992.
- [4] T.Y. Chang, W.E. Clark, and E.O. Hare. Domination numbers of complete grid graphs. I. *Ars Combin.*, 38:97–111, 1994.
- [5] D. Fisher. The domination number of complete grid graphs. Manuscript.
- [6] S. Gravier. Total domination number of grid graphs. *Discrete Appl. Math.*, 121(1-3):119–128, 2002.
- [7] T. Hagerup. Dynamic algorithms for graphs of bounded treewidth. *Algorithmica*, 27(3-4):292–315, 2000.
- [8] T.W. Haynes, S.T. Hedetniemi, and P.J. Slater. *Fundamentals of domination in graphs*, volume 208 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker Inc., New York, 1998.
- [9] M. Livingston and Q.F. Stout. Constant time computation of minimum dominating sets. *Congr. Numer.*, 105:116–128, 1994.
- [10] K.E. Proffitt, T.W. Haynes, and P.J. Slater. Paired-domination in grid graphs. *Congr. Numer.*, 150:161–172, 2001.
- [11] T. V. Wimer, S. T. Hedetniemi, and R. Laskar. A methodology for constructing linear graph algorithms. *Congr. Numer.*, 50:43–60, 1985.