A SAT + Computer Algebra System Verification of the Ramsey Problems *R(3,8)* and *R(3,9)*

by

Conor Duggan

A thesis
presented to the University of Waterloo
in fulfillment of the
research paper requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2024

Abstract

The Ramsey problem $R(p, q)$ asks for the smallest n such that every graph on n vertices must contain either a p-clique or an independent set of size q. We provide the first certifiable proof that $R(3, 8) = 28$, automatically generated by a combination of Boolean satisfiability (SAT) solver and a computer algebra system (CAS). While the $R(3, 8)$ problem was first computationally solved by McKay and Min in 1992, it was not a verifiable proof. This SAT+CAS combination is significantly faster than a SAT-only approach. We prove and verify $R(3, 9) = 36$ combining theory on the structure of graphs and a parallelisable SAT+CAS toolchain. The theory relating to $R(3, 9)$ was developed by Graver and Yackel in 1968 and expanded upon and finally proved by Grinstead and Roberts in 1982, again not a verifiable proof.

The SAT+CAS method that we use for our proof is very general and can be applied to a wide variety of combinatorial problems.

Acknowledgements

The progression of mathematics and science is not a solo endeavour. My research thesis was only possible by cumulative contributions and valuable discussions.

Foremost, I thank my supervisors Professor Vijay Ganesh and Professor Curtis Bright. Their direction and deep expertise over the course of almost two years was invaluable. Each maintain the highest standards in their work and I hope to have inherited some of their dedication to precision.

I owe a massive debt of gratitude to Brian Li. His support, mathematical and in other areas, was invaluable. Discussions with him are always interesting and fruitful.

On a similar note, I am grateful to the contributions of Daksh Segal, Piyush Jha and Jonathon Chung.

Finally, to everyone who stimulated my interest in learning and in mathematics, family, friends, teachers or professors, I would not have started this journey without you.

Dedication

This is dedicated to my parents and to friends. Sometimes a walk with someone makes all the difference.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ramsey Theory was first introduced by Frank P. Ramsey in *On a problem of formal logic* [41]. The theory ensures the existence of ordered substructures given a large enough structure. Given this broad and rather vague description, 'Ramsey type' problems exist in various branches of mathematics [13]. The problem has also been referred to as the 'party problem'; how many guests need to be invited to a party to ensure there are p guests who mutually know one another, or q guests who mutually do not know one another. Ramsey numbers are renowned and challenging problems; only 9 non-trivial Ramsey numbers are known, despite an extensive literature on the topic [40]. Erdős famously quoted in *Scientific American* that if aliens invaded Earth and demanded the value of $R(6,6)$, our resources would be better employed in a pre-emptive strike rather than attempt to find the solution. $R(6,6)$, indeed $R(5,5)$ remains unsolved to this day, over 30 years later. My work focused on the 'classic' Ramsey numbers defined below. Ramsey Theory has applications in communication theory [42], information retrieval problems [45] and decision trees [38].

The Ramsey Theorem states that for every p, $q \in \mathbb{Z}$, there exists an $n \in \mathbb{Z}$ such that every graph of order n, contains a p-clique or an independent set of size q. An m-clique is a complete subgraph of order m. A Ramsey problem is defined as finding the smallest integer n, denoted $R(p,q)$, for some given input p, q. A common, equivelant reformulation is as follows: the Ramsey Theorem states that for every p, $q \in \mathbb{Z}$, there exists an $n \in \mathbb{Z}$ such that any red/blue coloring of the edges of the complete graph of order n, denoted K_n, contains a blue monochromatic p-clique or a red monochromatic q-clique. Throughout this paper, I will use both formulations. A (p,q)-graph is a graph without a p-clique and without an independent set of size q. $(p,q;n)$-graphs and $(p,q;n;e)$-graphs are (p,q)-graphs on n vertices and (p,q)-graphs on n vertices and with e edges, respectively. All graphs are assumed to be simple and undirected, unless stated otherwise.

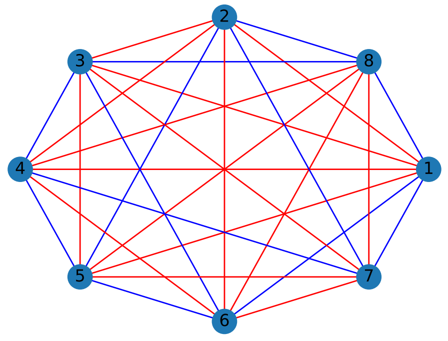

Figure 1.1: A red/blue edge coloring on 8 vertices without a blue 3-clique or red 4-clique, showing $R(3,4) > 8$.

$R(p,q)$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	1	1	1	1	1	1	1	1
2		2	3	4	5	6	7	8	9
3			6	9	14	18	23	28	36
4				18	25	36–40	49–58	59–79	73–106

Table 1.1: Values of some Ramsey numbers, where exact values are unknown, the best known lower and upper bounds are given [40].

Table 1.1 is a list of some Ramsey numbers. $R(p,q) = R(q,p)$, thus some numbers are excluded from the table for clarity. $R(1,q) = 1$ and $R(2,q) = q$ trivially.

A graph on n vertices has $n(n-1)/2 \in O(n^2)$ edges. There are $2^{n(n-1)/2}$ possible edge-colorings. Thus for a graph with 28 vertices, the search space is huge (i.e., there are more than 6×10^{113} possible colorings). Therefore, finding the exact values of Ramsey numbers is a challenging endeavor, and researchers often focus on bounds. Both theoretical and computational methods exist for improving upper or lower bounds. For example, Erdős theoretically proved $2^{k/2} < R(k,k) < 4^{k-1}$ [12] , while $R(4,8) > 57$ [14] was proven by using a satisfiability (SAT) solver to discover a $(4,8)$-graph on 57 vertices. Despite significant research in this area, it was not until 2023 that the exponent's base for the upper bounds of $R(k,k)$ Ramsey numbers was improved, yielding a bound of $(4-\epsilon)^k$, where $k = 2^{-7}$ [8]. In 1992, McKay and Min [35] computationally showed that no $(3, 8)$-graph exists on 28 vertices. Combined with a previous result that $R(3, 8) > 27$ [21], this showed $R(3, 8) = 28$. Contemporary proof techniques relying on computer-assistance necessitate formal verification.

Chapter 2

Boolean Algebra, SAT Solvers and CASs

This section describes the basics of Boolean algebra, SAT solvers and computer algebra systems (CASs). Boolean algebra was introduced by mathematician George Boole [4]. Values can take *True* ($\top$) or *False* ($\bot$). There are three operations, binary operations *AND* $\wedge$ and *OR* $\vee$, and unary operation *NOT* $\neg$ (or negation). A literal is a variable or its negation. A clause is a disjunction of literals: $x_1 \vee x_2 \vee \cdots \vee x_n$. A formula is a combination of literals and operations. A formula is in conjunctive normal form (CNF) if is a conjunction of clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_n$. The formula $(x_1 \vee x_2) \wedge x_3$ evaluates to *True* with $(x_1, x_2, x_3) = (\top, \bot, \top)$. Note this is not a unique solution. In this solution, we say x_1 is assigned *True*. A set of assigned variables is an assignment.

A conflict-driven clause learning (CDCL) satisfiability (SAT) solver is a computer program that takes as input a Boolean formula in CNF, and determines whether there exists an assignment of variables in the input formula such that the formula evaluates to *True*. The formula is called satisfiable (SAT) if so. If there does not exist an assignment of variables such that the formula evaluates as true, then the formula is unsatisfiable (UNSAT). These solvers learn clauses through unit propagation and back tracking. Unit propagation propagates the assignment of single literal through a formula. If no unit clauses exist, a literal is chosen. If a conflict is found, the solver back tracks to learn a clause to block this assignment of clauses. Often multiple variables are propagated before this happens. Consider the following CNF formula:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3 \vee x_4).$$

Suppose we choose $x_1 = \top$ to propagate and then $x_2 = \top$;

$$x_1\ (propagated):\ (\top \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (\bot \vee x_3) \wedge (\neg x_3 \vee x_4),$$

$$x_2\ (propagated):\ (\top \vee \bot \vee x_3) \wedge (\bot \vee \neg x_3) \wedge (\bot \vee x_3) \wedge (\neg x_3 \vee x_4).$$

Tidying this we see:

$$\neg x_3 \wedge x_3 \wedge (\neg x_3 \vee x_4)$$

From this, we can see that x_3 is a conflict. The solver back tracks to find the cause of the conflict, in this case $x_1 \wedge x_2$. The solver learns $\neg(x_1 \wedge x_2)$. CDCL SAT solvers can solve some instances with millions of variables efficiently [16]. However, SAT solvers face challenges when solving combinatorial problems such as the Ramsey problem, in part due to the considerable amount of *symmetry* in the associated search space.

Computer Algebra Systems (CASs), such as Maple and Mathematica, are storehouses of mathematical knowledge and are widely used to solve a variety of scientific and engineering problems. We use a CAS to dynamically provide mathematical context to the SAT solver in order to break symmetries in the search space associated with an input formula. In particular, we use a CAS to generate blocking clauses that are given to the SAT solver dynamically via a programmatic interface [15]. The clauses block the solver from exploring noncanonical matrix representations of a graph, since they are all isomorphic to one "canonical" representation. An adjacency matrix M of a graph is canonical if every permutation of the graph's vertices produces a matrix lexicographically greater than or equal to M, where lexicographical order is defined by concatenating the above-diagonal entries of the columns of the adjacency matrix starting from the left. We only need to examine the non-isomorphic lexicographically least matrices as the lexicographically least matrix represents its equivelance class under this isomorphism. Blocking noncanonical graphs is achieved using orderly generation [32] and is implemented through the SAT+CAS paradigm [7]. This technique can dramatically prune a formula's search space, since the CAS guides the SAT solver to not only block a noncanonical subgraph but also all its extensions. The SAT+CAS paradigm has been shown to be an effective approach to solving hard combinatorial math problems.

Specific to Ramsey problems, we performed an ablation study which demonstrated that SAT+CAS was 7 times faster than a SAT-only solver (2.3 seconds vs. 17 seconds) at solving $R(3,6) = 18$. The improvement was even more pronounced for $R(3,7) = 23$. These results were derived without any symmetry breaking constraints or other elements of our methodology. Further detail is provided through an ablation study in Section 9.

Chapter 3

Cube and Conquer

Cube and conquer is a parallelisation technique whereby a set of simpler instances are solved, and the aggregate result is equivalent to solving the original instance. Many combinatorial problems have been solved using this technique, *inter alia* Lam's Problem [5] and Pythagorean Triples [25].

Let v_i be a variable appearing in Boolean formula F. Then solving sub-instances $F_1 : F \wedge v_i$ and $F_2 : F \wedge \neg v_i$ is equivalent to solving F. If either F_1 or F_2 are satisfiable, then F is satisfiable. We say v_i was the variable split on. If a sub-instance is still hard to solve, an unassigned variable appearing in the sub-instance can be split on, subject to some selection criteria. This can be repeated subject to some stopping criteria.

Solving time improvement is based on the variables selected and the order in which they are selected. Several existing tools can rank variables to split on, such as March [24] and AlphaMapleSAT [26] . March uses a conflict driven clause learning SAT solver to compute metrics to rank variables to split on. AlphaMapleSAT leverages a Monte Carlo Tree Search to rank variables which may not immediately be optimal, but offer potential improvements after sufficient splitting. Based on experiments run by the AlphaMapleSAT developers, including on the Ramsey $R(3, 8)$ problem, we chose AlphaMapleSAT due to its improved performance.

We used the number of variables assigned as stopping criteria. A variable was split upon, propagated, simplified, and if the total number of assigned variables was greater than a predefined threshold, the cubing stopped for this sub-instance.

Following cubing, the cubes are 'conquered', *i.e.*, they are passed to a solver. Ideally, the solver can solve all cubes. However, when the SAT+CAS could not solve an instance without producing a proof file larger than 7GB, the instance was passed back to be cubed again. 7GB was chosen as files up to this size can fit on 4GB on RAM. This typically corresponded to 8–10 hours of solving. Thus cubing and conquering iterate on hard instances, but not on easier instances.

We used the Python multiprocessing library to facilitate solving many cubes on few CPUs.

Chapter 4

Methodology

Under our methodology, a problem is encoded for a pre-defined n, p, q by deriving a Boolean formula in conjunctive normal form asserting the existence of a (p, q)-graph of order n. The encoding enforces every p-clique to have at least one edge in the opposing (red) color and every q-clique to have at least one edge in the opposing (blue) color, i.e.,

$$\bigwedge_{K_p \subseteq K_n} \bigvee_{e \in K_p} \neg e \wedge \bigwedge_{K_q \subseteq K_n} \bigvee_{e \in K_q} e,$$

where the variable e is assigned True exactly when the corresponding edge is colored blue. We refer to this as the 'basic Ramsey' encoding. A satisfying assignment of the encoding corresponds to finding a (p, q)-graph of order n. Similarly, an unsatisfiable result means no such colorings exist for this particular n, i.e., all colorings contain a blue p-clique or a red q-clique.

In order to break symmetries in the problem, we add constraints that enforce a lexicographic ordering on rows of the graph's adjacency matrix, before using the dynamic symmetry breaking capacities of a CAS. These constraints are a partial static symmetry breaking technique; see Chapter 5 for details. There is an overhead associated with calling the CAS, thus by including these constraints, we reduce the number of CAS calls. This a CNF encoding for (p, q)-graphs on n vertices, and can be passed to a SAT solver or SAT+CAS. We refer to this encoding as the 'standard encoding'. As described in Chapter 5, we experimented with further constraints to improve solve time. For harder instances, we used a parallelisation technique, as described in section Chapter 3. The methodology accepts various n, p, q and does not need different techniques or theories to solve such problems.

Proof correctness is an important step in computer-assisted proofs, in particular for non-existence proofs. Each time SAT+CAS returns UNSAT or SAT, we utilise a modified DRAT-trim

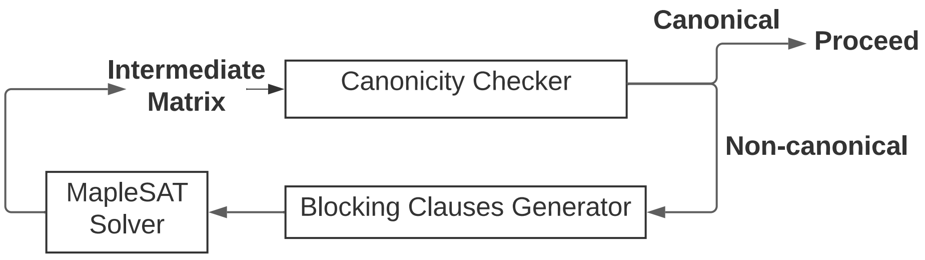

Figure 4.1: A flowchart of the orderly generation algorithm implemented as part of MathCheck's SAT+CAS architecture.

proof checker to verify that the clauses learned by the CAS block non-canonical graphs and the SAT solver's search is exhaustive. This toolchain is known as MathCheck[1].

It was previously known that $R(3, 8) > 27$ [21]. Our methodology produced a satisfying assignment on order 27 for the $R(3, 8)$ problem after 7.5 hours. Graver and Yackel found a cyclic $(3, 9; 35)$-graph, demonstrating $R(3, 9) > 35$ [20]. The $R(3, 9; 35)$ basic encoding has over 70 million clauses. The standard encoding yields a 9GB file. Such a large encoding was not feasible for our tools.

[1]https://github.com/BrianLi009/MathCheck

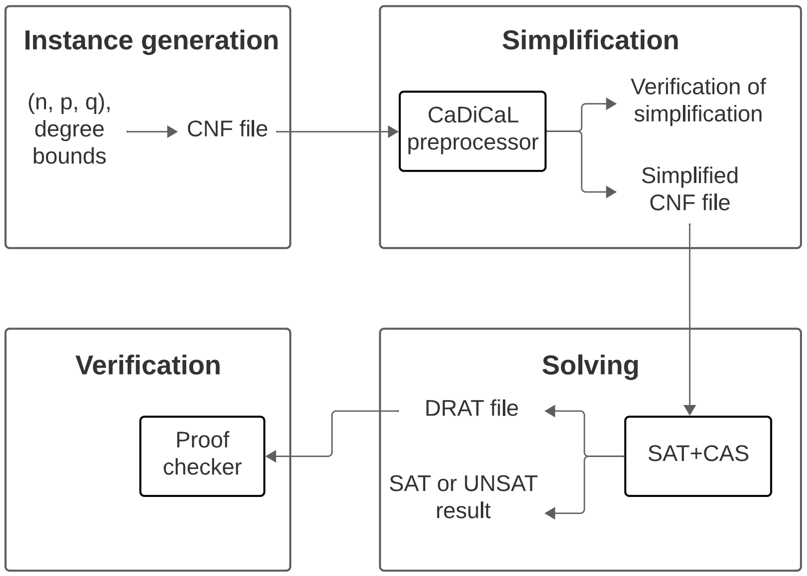

Figure 4.2: Flowchart of pipeline without parallelisation from input parameters to verification.

Chapter 5

Constraints

Additional constraints can be included in an encoding to potentially improve solve times. Additional constraints can arise from theoretical results, or other methods. For example, in deriving $R(4, 8) > 57$, the authors utilised unit clauses which could be iteratively removed if they resulted in a significant number of conflicts [14]. Typically, these constraints narrow the search space or improve propagation speed, although they can also slow down solve times. Additional constraints may use auxiliary variables which increases the number of variables in an encoding. Constraints can be grouped into exhaustive and non-exhaustive categories. Exhaustive constraints do not affect the exhaustive search of a SAT solver, however, non-exhaustive constraints limit the search space. Limiting the search space intuitively should provide a computational benefit, however, it may not be able to prove non-existence. Ultimately, the purpose of additional constraints is to yield a faster result.

Using additional constraints introduces sources of error. Our code may not perform as expected, or the encoding may be incorrect. For exhaustive constraints, we can confirm that the number of $(p, q; n)$-graphs with and without the additional constraints does not change. However, by definition, this does not hold for non-exhaustive constraints. In the one instance where we used non-exhaustive constraints, we wrote a programme to verify that satisfying assignments had the expected structure.

There is no one method to encode constraints into Boolean logic and the 'best' encoding of constraints is not obvious. We experimented with the Sinz sequential [43] encoding and the Bailleaux and Boufkand totalizer encoding [2] to encode cardinality constraints on the edges. Encoding additional constraints requires clauses, and may introduce new variables. The number of clauses or variables introduced does not necessarily result in a faster or slower solve time.

An important concept of constraint encoding for SAT solvers is *arc-constistency*. Arc consistent encodings allow the removal of values from the domains of variables that can never be part of a satisfying solution.

Symmetry breaking constraints

We encoded partial static symmetry breaking constraints, developed by Codish et al. [10], which enforce a lexicographical ordering on the rows of a graphs adjacency matrix. These block the solver from exploring certain symmetric portions of the search space, before the CAS is called. There is an overhead associated with calling the CAS. Given an adjacency matrix A of a graph, define $A_{i,j}$ as the i^{th} row of A without columns i and j. The clauses enforce that $A_{i,j}$ is lexicographically equal or less than $A_{j,i}$, $1 \leq i < j \leq n$. These clauses introduce $O(n^3)$ auxiliary variables and clauses. Based on our empirical evidence, these constraints provide a significant speed up to solving and were included in all runs.

Edge constraints

These enable searching specifically for $(p, q; n; e)$-graphs. The non-existence of these graphs for particular p, q, n, e is an important result in the original proof for $R(3, 9)$, and we use them in our approach to solve $R(3, 9)$. We experimented with the Sinz sequential counter and Bailleaux and Boufkand totalizer encodings. Suppose we wish to encode between l and u of m variables are $\top$.

Sinz sequential cardinality encoding

Define $k = u+1$. The sequential counter is based on an $m \times k$ matrix A where $a_{i,j}$ is $\top$ if at least j of $x_1, \dots, x_i$ are $\top$ [6]. The relationship between auxiliary variables $a_{i,j}$ and edge variables is given by

$$a_{i,j} \iff (a_{i-1,j} \vee (x_i \wedge a_{i-1,j-1})) \text{ for } 1 \leq i \leq m \text{ and } 1 \leq j \leq k.$$

This formula is encoded to CNF by the following four clauses, $\neg a_{i-1,j} \vee a_{i,j}$, $x_i \vee \neg a_{i-1,j-1} \vee a_{i,j}$, $\neg a_{i,j} \vee a_{i-1,j} \vee x_i$, and $\neg a_{i,j} \vee a_{i-1,j} \vee a_{i-1,j-1}$. Additionally, $s_{0,j}$ will be false for $1 \leq j \leq k$ and that $a_{i,0}$ will be true for $0 \leq i \leq m$. $a_{n,l} = \top$ and $a_{m,k} = \bot$. More variables can in fact be assigned in the matrix, however, the clauses allow these to be quickly assigned if they are unassigned. This encoding uses both $O(mk)$ auxiliary variables and clauses.

Bailleaux and Boufkand totalizer cardinality encoding

In constrast, the totalizer is based on a binary tree. Each node in the tree is assigned a value and a set of unique variables. If a node's value M is greater than 1, the node is given two children nodes a, with value $\lfloor M/2 \rfloor$, and b with value $M - \lfloor M/2 \rfloor$. The root node is assigned a value u. This yields a tree with exactly m leaves. Each leaf is bijectively assigned one of the m variables. The tree root is assigned u auxiliary variables, known as the counting variables. Finally, each remaining internal node with value M is assigned M auxiliary variables, known as the linking variables.

For a non-leaf node r with children a, b, define $R = \{r_1, \ldots, r_m\}$, $A = \{a_1, \ldots, a_{m_1}\}$ and $B = \{b_1, \ldots, b_{m_2}\}$ be the set of variables assigned to r, a and b respectively. The following conjunction of clauses is related to the node r:

$$\bigwedge_{\substack{0 \le \alpha \le m_1 \\ 0 \le \beta m_2 \\ 0 \le \sigma \le m \\ \alpha + \beta = \sigma}} C_1(\alpha, \beta, \sigma) \wedge C_2(\alpha, \beta, \sigma).$$

with the following notations:

$$a_0 = b_0 = r_0 = \top, a_{m_1+1} = b_{m_2+1} = r_{m+1} = \bot,$$

$$C_1(\alpha, \beta, \sigma) = (\neg a_\alpha \vee \neg b_\beta \vee r_\sigma),$$

$$C_2(\alpha, \beta, \sigma) = (a_{\alpha+1} \vee b_{\beta+1} \vee \neg r_{\sigma+1}).$$

$C_1(\alpha, \beta, \sigma)$ is the CNF representation of the relation $\alpha + \beta \le \sigma$ and $C_2(\alpha, \beta, \sigma)$is the CNF representation of the relation $\alpha + \beta \ge \sigma$. Finally, for counting variables c_i, the following clauses defines the bounds,

$$\bigwedge_{1 \le i \le l} (c_i) \bigwedge_{u+1 \le i \le m} (\neg c_i).$$

This encoding uses $O(m \log m)$ new variables and $O(m^2)$ new clauses, thus is independent of u, unlike the Sinz encoding. For larger u, we used the totalizer encoding.

In both encodings, more variables can in fact be assigned $\top$ or $\bot$ in advance, however, the clauses allow these to be quickly assigned if left unassigned. A larger encoding size does not necessarily reduce solve time. We considered using other cardinality constraints. The above two were chosen as they are arc-consistent, whereas some other encodings are not.

Degree constraints

These constraints state that for a (p, q)-graph on n vertices, each vertex v satisfies $n - R(p, q - 1) \leq \deg_b(v) \leq R(p - 1, q) - 1$ where $\deg_b(v)$ is the number of (blue) edges on vertex v [20]. To see the upper bound, suppose a vertex v in a (p, q)-graph has at least $R(p - 1, q)$ neighbours joined with v by blue edges. Then by definition, this neighbourhood contains either a red q-clique or a blue $(p - 1)$-clique. The first is a clear contradiction and the second is too because the blue $(p - 1)$-clique and v form a blue p-clique. The same argument can be used on the red edges to give the lower bound.

Note from this proof that the Ramsey constraints directly imply these constraints. Thus, they add no new information. However, as they improve solver performance, they appear to assist with propagations. As the bounds on these are low, the largest used being 8, we used the Sinz sequential counter.

Triangle constraints

There is an upper bound on the number of monochromatic triangles appearing on an edge in a (p, q)-graph. In fact, these along with the degree constraints, form a family of constraints limiting the number of k-cliques in a (p, q)-graph [11]. For the experiments we ran, the largest bound used was 17, thus we only tested the Sinz cardinality encoding. This was only tested this for $p = 3$. These constraints slowed the total solve time. These constraints yield long clauses, under our encoding. Indeed, the clause length increases with exponentially if we limited the number of k-cliques. Short clauses need less assignments to yield a propagation. We theorise the long clauses under our encoding do not assist the solver.

Maximal clique free constraints

These constraints enforce that graphs are maximally blue p-clique free, that is, if any red edge is recoloured to blue, then a blue p-clique is formed. Indeed, a (p, q)-graph exists iff a maximal p-clique free (p, q)-graph exists. This can be proved trivially. A similar result holds for maximally red q-clique free, although one direction of the equivalence does not necessarily hold if both maximal red and maximal blue statements are true. Unlike the aforementioned constraints, these are non-exhauastive constraints. In order to lend credence to our encoding, we wrote a programme to verify that graphs found with these constraints are maximally blue p-clique free. p was chosen to keep encoding files small.

Two encodings were considered, a direct encoding and an alternate encoding, which uses auxiliary variables. The direct encoding introduces no new variables but is exponential in n^{p-1} in the number of clauses used. Under this encoding, the maximality constraint is a disjuction of conjunctions, each of size $\binom{n-2}{p-2}$ which requires an exponential in $\binom{n-2}{p-2}$ number of clauses to convert to CNF. The alternate encoding uses $O(n^p)$ clauses and new variables. The alternate encoding was implemented by introducing for each edge e auxiliary variables $V_P(e)$ for each p-clique P on e minus e, $V_P(e) \equiv \bigwedge_{e_i \in P(e)} e_i$, and maximality constraints $\neg e \rightarrow \bigvee_{v \in V_P(e)} v$.

Chapter 6

Comparison of *R(3,8)* Methods

In this section, we will describe the method used by McKay and Min, and outline some advantages of our method.

First, the authors use a previous result, that $R(3,8) \in \{28, 29\}$ [21]. They intend to generate graphs on 28 vertices without 3-cliques and without independent sets of size 8. These graphs are equivalent to graphs without blue 3-cliques and without red 8-cliques. If at least one such graph is found, then $R(3,8) = 29$. However, by extending particular graphs, and using graph isomorphism rejection, the authors show that no such graphs exist on 28 vertices. Graph isomorphism takes the usual definition in their paper. Note that isomorphic graphs have the same canonical labeling.

Their method extends graphs without 3-cliques and without independent sets of size t. The authors first outline the following two results for graphs on n vertices that have no 3-cliques and no independent set of size t:

- $n - R(3, t-1) \leq \delta \leq \min(t-1, n/2)$ where δ is the minimum degree of the graph,
- Such graphs can only arise from extension of graphs without 3-cliques, without independent sets of size $t-1$ and on $n - \delta - 1$ vertices.

The first point above is an equivalent statement to the degree constraints used in our encoding.

Thus, the authors must generate all graphs (up to isomorphism), on $\{20, 21, 22\}$ vertices for $t = 7$ to show no such graphs on 28 vertices exist for $t = 8$. This corresponds to generating just under 5.2 million graphs with distinct canonical forms. To generate these graphs, the authors generated the equivalent graphs for $t = 6$. The starting point for this chain of graph generation was $t = 3$ and $1 \leq n \leq 5$. The generation procedure is recursively called and each graph must pass three criteria to be extended. In total, over 5.2 million distinct graphs were generated. After a graph was generated, the authors converted it to canonical form using *nauty* [36] to remove isomorphic graphs. Before non-canonical graph removal, their procedure generated 695 million extensions of graphs without 3-cliques, without independent sets of size 6, and on 15 vertices. The authors' generation procedure is specific to $R(3, q)$ problems. For $R(p, q)$ problems with $p \neq 3$, a modified generation procedure would be required.

McKay and Min validate their computational method by noting the number of graphs generated for different combinations of n and t, for $t \leq 7$, align with previously known values.

In comparison, we argue our method has two key benefits. Primarily, it generates a proof certificate. For $R(3, 8)$ we generated a 31 GB DRAT file which can be independently verified, without relying on the correctness of either the SAT solver or the CAS. Secondly, the SAT+CAS method is simpler to implement. In theory, once the CNF file representing the Ramsey problem has been generated it can be immediately passed to a SAT+CAS solver. We also, as outlined in the Methodology section, include additional constraints such as symmetry breaking constraints and vertex degree constraints. Similarly, the SAT+CAS method has broader applicability as it is independent of p, q, and n, and only needs a CNF encoding of the problem, whereas the McKay and Min extension procedure would need modification for different (p, q) with $p \neq 3$. We note the caveat that technological improvements have certainly played a large role run time differences.

Beyond this specific problem, there are additional benefits to the SAT+CAS method. The SAT+CAS method allows for further constraints, typically derived from theory, to be easily included in the encoding, as outlined in Chapter 5. Further constraints limit the search space and may improve search speed. Although not used in deriving $R(3, 8) = 28$, SAT+CAS is parallelisable through the 'cube and conquer' technique [23]. [1]

[1]We did use parallelised MathCheck on the Ramsey problem on a machine with 8 CPUs. The total computational time to solve and verify was 247 hours and the real time was 30 hours. 89 cubes were generated.

Chapter 7

The search for *R(3,9)*

In 1968, Graver and Yackel [20] improved many Ramsey number bounds, and improved many bounds on the minimum number of edges required in $(p, q; n)$-graphs. By a constructive method, and computer verification by Kalbfleisch, a former student and researcher at University of Waterloo, they determined $R(3, 9) \geq 36$. They developed a method which showed $R(3, 9) \leq 37$ and even further, they proved that a $(3, 9; 36)$-graph must be regular of degree 8. Finally, they noted that if a $(3, 8; 27)$-graph with 80 edges does not exist, then $R(3, 9) = 36$. In 1982, Grinstead and Roberts computationally showed no $(3, 8; 27; 80)$-graphs exist [21]. It was not until 1992 that $R(3, 8) = 28$ was computationally proven by McKay and Min, in theory an easier problem.

Directly applying a SAT+CAS to the $R(3, 9)$ problem is difficult. Firstly, the CNF file is large, 9.2Gb including the static symmetry breaking constraints and degree constraints, and thus a large amount of RAM is needed. As the problem is computationally harder than the $R(3, 8)$ problem, a parallisable approach or other method is needed. Below, we outline the methods of Graver and Yackel and Grinstead and Roberts, which combine theory and computation to show $R(3, 9) = 36$. Where relevant, we outline our use of solving and verifying tools to replicate and verify their results. However, we have not implemented graph gluing into the SAT+CAS method, which is a key piece of Grinstead and Roberts' method, and thus could not directly replicate their approach using SAT+CAS. In this case, graph gluing refers to inserting an $n \times n$ adjacency matrix and an $m \times m$ adjacency matrix along the main diagonal of a larger empty matrix, to form an $(m+n) \times (m+n)$ adjacency matrix. The off-diagonal is then filled in, subject to some constraints (in this case the Ramsey constraints), using an arbitrary method. Graph gluing relies on extending immutable submatrices of an adjacency matrix; however the CAS generates permutations which affect the entire matrix. This could be resolved by restricting the CAS to generate permutations from the stabaliser set of these submatrices. Instead we search for $(3, 8; 27; 80)$-graphs using a parallelised SAT+CAS. Using $R(3, 8) = 28$, which we have computationally verified, the

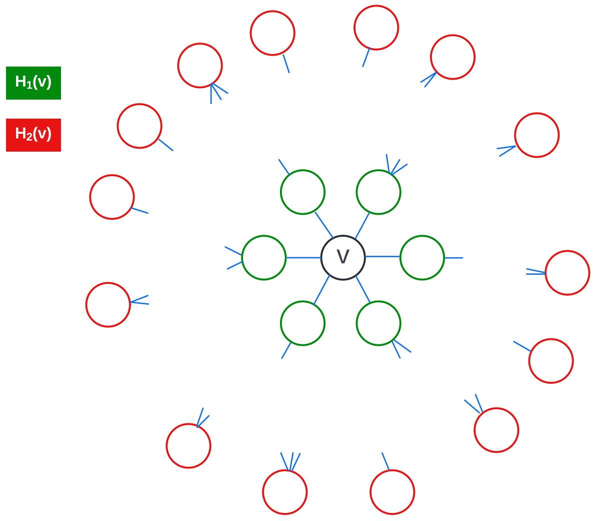

Figure 7.1: Graph with a preferred vertex $v, H_1(v)$ and $H_2(v)$. Note for $p = 3, H_1(v)$ is an independent set.

theoretical results of Graver and Yackel on the structure of $(3, 9; 36)$-graphs are significantly easier to prove.

Lemma 1. *A* $(3, 9; 36)$*-graph contains a* $(3, 8; 27; 80)$*-graph.*

Proof. In a $(p, q; n)$-graph, choose a preferred vertex v of degree d. Define $H_1(v)$ as the graph generated by the neighbourhood of p and $H_2(v)$ as the graph generated by $V(G) \setminus (v \cup H_1)$. In some literatures [17], $H_2(v)$ is referred to as the 'anti-neighbourhood' of v.

Then $H_1(v)$ is a $(p-1, q; d)$-graph and $H_2(v)$ is a $(p, q-1; n-d-1)$-graph. Suppose $H_1(v)$ is not a $(p-1, q; d)$-graph, *i.e.* it contains a $(p-1)$-clique or an independent set of size q. Clearly the latter is a contradiction. If the former, the $H_1(v) \cup v$ is a p-clique. Similarly if there exists an independent set of size $q-1$ is $H_2(v)$, then $H_2(v) \cup v$ is an independent set of size q. In fact, $H_1(v)$ is an independent set if $p = 3$. By definition, v is not joined to $H_2(v)$ by any edge.

Now a $(3, 9; 36)$-graph, by the degree constraints is regular of degree 8, must have 144 edges. As all vertices have equal degree, our choice of v is irrelevant. 8 (blue) edges join v to $H_1(v)$ and due to the regularity, 7 (blue) edges join each vertex in $H_1(v)$ to vertices in $H_2(v)$.

Thus a total of 64 (blue) edges are used. The remaining 80 edges must be in $H_2(v)$, which is a $(3, 8; 27)$-graph. □

Graver and Yackel originally used formula (7.1) below to derive this same result. Our proof above uses the degree bounds of $(3, 9; 36)$, which uses $R(3, 8) = 28$ and was unknown at the time. We will show no $(3, 8; 27; 80)$-graphs exist. Deriving the non-existence of $(3, 8; 27; 80)$-graphs is a computationally hard problem. Below, I outline the approach of the original authors.

The authors of both papers relied on the following definitions:

- $v_i :=$ a vertex of degree $q - 1 - i$,
- $s_i :=$ the number of vertices of degree v_i,
- $e(p, q; n) :=$ the minimum number of edges possible in a (p, q)-graph on n vertices,
- full vertex := a vertex v such that $||H_2(v)|| = e(p, q - 1, n - deg(v) - 1)$,
- $Z(v) := \sum_{v_i \in N(v)} \deg(v_i)$

Note that the maximum degree in a $(3, q)$-graph is $q - 1$, so subscript i can be thought of as the difference between the degree of the vertex and the maximum possible degree in the graph [21]. For a full vertex v of degree k, $Z(v) \geq Z(u)$ for all other vertices u of degree k.

The following lemma was proved by Graver and Yackel and Grinstead and Roberts used a slightly modified version. For our purposes, we will use the latter.

Lemma 2. *Let G be a $(p, q; n; e)$-graph. Let*

$$\phi = ne - \sum_{i \geq 0} (e(3, q - 1; n - v_i - 1) + v_i^2) s_i. \tag{7.1}$$

Then $\phi \geq 0$ and there are at least $n - \phi$ full vertices in G. [1]

Proof. Let $\beta_{ij}(v)$ be the number of vertices in $H_1(v)$ of degree v_j if $\deg(v) = v_i$, 0 otherwise.

Note that $\sum_v \beta_{ij} = \sum_v \beta_{ji}$ as $\sum_v \beta_{ij}$ is the number of edges from vertices of degree v_i to vertices of degree v_j. For a preferred vertex p of degree v_i,

$$||G|| = ||H_2(p)|| + v_i^2 + \sum_{j \geq 0} (i - j) \beta_{ij}.$$

Intuitively, this formula can be explained as

|edges in G| = |edges in $H_2(v)$|+|edges arising by assuming remaining vertices have degree v_i|+

[1]Grinstead and Roberts used Δ instead of ϕ; however, as Δ has a common interpretation in graph theory, I use alternate notation.

$$|\text{add or subtract the edges missing or added from the assumption}|.$$

Summing over all vertices,

$$ne = \sum_v |H_2(v)| + \sum_i (v_i)^2 s_i + \sum_{i \geq 0} \sum_{\deg(v)=v_i} \sum_{j \geq 0} (i-j)\beta_{ij}.$$

For fixed i, j, $\sum_v \beta_{ij}$ has $(i-j)$ as a coefficient, and $\sum_v \beta_{ji}$ has $(j-i)$ as a coefficient. Thus, these terms cancel in our above formula.

By definition, $|H_2(v)| \geq e(p, q-1; n - \deg(v) - 1)$, and this is an equality if v is a full vertex, also by definition.

Thus,

$$\begin{aligned} ne &\geq \sum_{i \geq 0} e(p, q-1; n - v_i - 1)s_i + (v_i)^2 s_i, \\ &= \sum_{i \geq 0} (e(p, q-1; n - v_i - 1) + (v_i)^2) s_i. \end{aligned}$$

This shows $\phi \geq 0$. Further, each vertex which is not full contributes at least 1 to the value of ϕ, so there are at least $n - \phi$ full vertices in G. □

Using this formula, Graver and Yackel improved the bounds on many Ramsey numbers and on the minimum number of edges in Ramsey graphs. Specifically for $R(3,9)$, Grinstead and Roberts used this to derive a sequence of lemmas on the structures of various subgraphs of $(3,8;27;80)$-graphs. They computationally proved these such structures in $(3,8;27;80)$-graphs cannot exist, thus showing $R(3,9) = 36$. We will follow their method and use a SAT+CAS to perform and verify the computational results, where viable.

Lemma 3. $e(3,7;19) = 37$

Proof. We use an encoding which asserts the existence of a $(3,7;19;\leq 36)$-graph. The SAT+CAS returns UNSAT in 249 seconds and verifies in 326 seconds. Using the SAT+CAS, we find an assignment of a $(3,7;19;37)$-graph in 32 seconds which verifies in 24 seconds. Edge counts were encoded using the totalizer counter.

□

Grinstead and Roberts proved this result by first proving a $(3,7;19;36)$-graph must contain either a full vertex of degree 3 or a full vertex of degree 4. They computationally showed this does not occur by gluing a graph such a vertex with known $(3,6;\{14,15\})$-graphs and checking if extensions of this contain triangles or 7-cliques. Graver and Yackel proved $36 \leq e(3,7;19) \leq 37$ using formula 7.1 and constructing a cyclic $(3,7;19;37)$-graph.

Lemma 4. $e(3,7;20) = 44$

Proof. We use an encoding which asserts the existence of a $(3,7;20;\leq 43)$-graph. The SAT+CAS returns UNSAT in 907 seconds and verifies in 2701 seconds. Using the SAT+CAS, we find an assignment of a $(3,7;20;44)$-graph in 62 seconds which verifies in 27 seconds. Edge counts were encoded using the totalizer counter. □

This was proven by Graver and Yackel using (7.1) and a constructive method to find a $(3,7;20;44)$-graph.

Lemma 5. $e(3,7;21) = 51$

Proof. We use an encoding which asserts the existence of a $(3,7;21;\leq 50)$-graph. The SAT+CAS returns UNSAT in 451 seconds and verifies in 897 seconds. Using the SAT+CAS, we find an assignment of a $(3,7;21;51)$-graph in 127 seconds which verifies in 130 seconds. Edge counts were encoded using the totalizer counter. □

Grinstead and Roberts proved this result by first proving a $(3,7;21;50)$-graph must contain either a full vertex of degree 4 with two vertices of degree 4 and two vertices of degree 5 as neighbours. They computationally showed this does not occur, using a similar method to the approach outlined in lemma 3. Graver and Yackel show $50 \leq e(3,7;21) \leq 51$ using formula 7.1 and constructing a cyclic $(3,7;21;51)$-graph.

Lemma 6. *If G is a $(3,7;22;\leq 62)$-graph, then $\delta(G) \geq 5$*

Grinstead and Roberts showed this does not occur by assuming a vertex of degree 4 exists and applying a similar method to the approach outlined in lemma 3. Vertices of degree ≤ 3 are ruled out by the degree bounds.

This could be performed with a SAT+CAS which allowed graph gluings. Otherwise, 22 different encodings could be solved whereby each encoding enforces a different vertex to have degree 4. Based on the run times of the prior lemmas, which are of a similar order, we do not expect this to involve a significant amount of computation.

Lemma 7. *If G is a $(3,8;27;80)$-graph, then $\delta(G) \geq 5$.*

Proof. This is a theoretical proof and relies on the previous lemmas. $\delta(G) \geq 4$ by the degree constraints. Let v be a vertex of degree 4 with two vertices of degree 4 as neighbours, say w_1 and w_2. Then $w_2 \in H_2(w_1)$ and w_2 has degree at most 3 in $H_2(w_1)$. But $H_2(v)$ is a $(3,7;22)$-graph and by the degree constraints, $H_2(w_1)$ has no vertices of degree 3. Thus, each vertex of degree 4 in G has at most one neighbouring vertex of degree 4. Hence, if v is a vertex of degree 4, then $Z(v) \geq 19$, and so $|H_2(v)| \leq 61$. By lemma 6, $H_2(v)$ has no vertex of degree 4, and so G has at most two vertices of degree 4. Using formula and 7.1 and counting the edges and vertices in G,

$$\phi = 2160 - 86s_0 - 80s_1 - 76s_2 - 76s_3 \geq 0,$$

$$160 = 7s_0 + 6s_1 + 5s_2 + 4s_3,$$
$$27 = s_0 + s_1 + s_2 + s_3.$$

By taking cases on $s_3 = 2$ and $s_3 = 1$ (in this case, s_3 corresponds to the number of vertices of degree 4), we reach contradictions. □

Lemma 8. *If G is a $(3,8;27;80)$-graph, then G contains either a full vertex of degree 6 with all neighbours of degree 6, or a full vertex of degree 5 with fours neighbours of degree 6 and one neighbour of degree 5.*

Proof. We use the system of inequalities given in the proof of lemma 7 and set $s_3 = 0$, yielding:

$$s_2 = s_0 + 2,$$

$$s_1 = 25 - 2_s0,$$
$$\phi = 8 - 2s_0 \geq 0.$$

Hence, $s_0 \leq 4$ and $2 \leq s_2 \leq 6$. Thus, $s_0 \leq 4$ and $2 \leq s_2 \leq 6$. As $e(3,7,20) = 44$ and $e(3,7,21) = 51$, if $\deg(v) = 5$ then $Z(v) \leq 29$ and if $deg(v) = 5$ then $Z(v) \leq 36$. This implies the number of edges between vertices of degrees 5 and 6 is at least as great as the number of edges between vertices of degrees 6 and 7.

Case 1: $s_2 \leq 4$ Each vertex of degree 5 has at most four neighbours of degree 6, and no vertex of degree 6 is adjacent to a vertex of degree 7 but not a vertex of degree 5. Thus, there are at most 16 vertices of degree 6 adjacent to a vertex of degree 5 or 7. By solving for s_1, we see there are at least 21 vertices of degree 6, hence there is a vertex of degree 6 with neighbours all of degree 6.

Case 2: $s_2 = 5$ If no vertex of degree 6 with neighbours all of degree 6, then degree 6 vertex has at least one neighbour of degree 5. As there at 19 vertices of degree 6, there are at least 19 edges between vertices of degree 5 and 6. There are only 5 vertices of degree 5, so some vertex v has at least 4 neighbours of degree 6. As $Z(v) \leq 29, v$ must have exactly four neighbours of degree 6 and one neighbour of degree 5.

Case 3: $s_2 = 6$ In this case, $\phi = 0$, so every vertex is full by lemma 2. If v has degree 7, then $Z(v) = 43$ as $e(3, 7, 19) = 37$. v must have at least one neighbour of degree 7. By the inequalities, there are four vertices of degree 7. Hence, there are only four possibilities for the subgraph generated by these vertices as there can be no triangles. Let v be a vertex of degree 7 which has i neighbours of degree 7. Then v has $(i - 1)$ neighbours of degree 5 as $Z(v) = 43$. Thus in each of the four possibilities, there are at most four edges between vertices of degree 5 and 7. As there are six vertices of degree 5, there must be a vertex w of degree 5 not adjacent to any vertex of degree 7. As $Z(w) = 29$ as $e(3, 7, 21) = 21$, w has four neighbours of degree 6 and one neighbour of degree 5. □

Lemma 9. *No* $(3, 8; 27; 80)$*-graph exists.*

Proof. Again, Grinstead and Roberts applied a similar method to the approach outlined in lemma 3, where the glued graphs are the graphs generated by the full vertices described in lemma 8 with either $(3, 7; 20; 44)$-graphs or $(3, 7; 21; 51)$-graphs. □

There are 19 canonical $(3, 7; 20; 44)$-graphs and $(3, 7; 21; 51)$-graphs [18]. We used a SAT+CAS to find all such graphs. Using a SAT+CAS which allows graph gluing methods, lemma 9 could also be solved and verified in this manner.

Grinstead and Roberts estimated 5×10^{10} machine operations and 2.5×10^4 seconds of computation, but note the time could be further improved with machines with more efficient bit-string operations. Computations were performed on a Honeywell Level 66 computer.

We apply a parallelised SAT+CAS to an encoding asserting the existence of a $(3, 8; 27; 80)$-graph to prove directly lemma 9. In doing so, we do not rely on the prior lemmas. However, the trade-off for this is an expected increase in computation time.

Chapter 8

PySMS

Independently, another research group developed a similar tool, PySMS [29]. This uses the SAT solver CaDiCaL [3] in conjunction with a non-CAS method to check the canonicity of partial solutions, known as SAT Modulo Symmetry (SMS). The developers also included a Python tool to build CNF files for various common graph constraints. However, this does not include the partial static symmetry breaking constraints described previously. These constraints were not used in conjunction with SMS during our testing using this tool. The authors use a different definition of canonicity which concatenates across the rows in the upper diagonal part of a graph's adjacency's matrix, as opposed to the MathCheck method which uses the columns in the upper diagonal. By using columns, the first i columns, $1 \leq i \leq n$, form the upper diagonal of an adjacency matrix, which is subgraph of a graph on n vertices. Blocking a non-canonical partial solution corresponds to blocking all extensions of this subgraph.

Results given using SMS can also be verified using a DRAT proof checker. Learned symmetry breaking clauses can be verified in a similar fashion to verification using a SAT+CAS and accounting for the change is canonicity definition. This is performed by checking each blocking clause corresponds to a non-canonical matrix. This is performed in the following fashion:

1. A blocking clause is converted to the partially assigned adjacency matrix A it blocks. Unassigned entries are given a unique value.

2. The permutation q applied during solving to check canonicity is applied to the adjacency matrix, resulting in matrix Q.

3. The partially assigned matrices are compared to determine if Q is in fact lexicographically lesser than A. It is important to account for the unassigned entries. If for some i, j, $a_{i,j} \neq$

$q_{i,j}$, where $a_{i,j} \in A, q_{i,j} \in Q$ both unassigned, and all relevant previous entries of the matrices were unable to determine the lexicographical ordering of A and Q, then q is not a witness not the non-canonicity of A, and the verification fails.

Chapter 9

Ablation Study

We compare elements of the encoding on $R(3,7;23)$ and $R(3,8;28)$ which are exhaustive searches. The times presented here represent the solve times summed with simplification times, if any. Times associated with instance generation are excluded. Verification times are excluded, however, times spent during solving to write to a proof file are included.

These were performed on an Intel E5-2683 v4 running at 2.1GHz. From Table 9.1, the SAT+CAS significantly outperforms the SAT-only solver. For the easier instance, $R(3,7;23)$, it appears a simpler encoding is the fastest method to solve the instance. However, this did not hold for the harder instance. We note that as the easier instance is solved very quickly, variations of a couple of seconds, perhaps due to external factors, change the interpretation of the fastest method.

Further constraints were experimented with for MathCheck – maximum clique free and tri-

Instance	**SAT**	**MathCheck**	**MathCheck+ DC**	**MathCheck, reduced simp**
R(3,7;23)	$>$ 86,400 s	18 s	23 s	17 s
R(3,8;28)	$>$ 100 hr	69 hr	57 hr	$>$ 100 hr

Table 9.1: CPU time of different techniques, *s* stands for seconds, and *hr* stands for hours. The headings of the table are: *SAT* uses a SAT-solver on the standard encoding, *MathCheck* uses MathCheck on the standard encoding, *MathCheck+DC* includes degree constraints to the standard encoding, *MathCheck, reduced simp* uses 10k conflicts for simplification of the standard encoding, rather than the full 100k

Instance	**SMS**	**SMS+ DC**	**SMS+ MCF**	**SMS+ DC+ MCF**
R(3,7;23)	17 s	12 s	14 s	20 s
R(3,8;28)	28 hr	31 hr	24 hr	18 hr

Table 9.2: CPU time of different techniques, *s* stands for seconds, and *hr* stands for hours. The table headings are: *SMS* uses the SMS solver on the basic encoding, *SMS+ DC* uses the SMS solver on the basic encoding with degree constraints, *SMS+ MCF* uses the SMS solver on the basic encoding with maximal p-clique free constraints, *SMS+ DC+ MCF* uses the SMS solver on the basic encoding with degree constraints and with maximal p-clique free constraints.

angle constraints, see Chapter 5. However, these did not improve solve times, nor using them in conjunction with each other or the degree constraints, and are thus excluded from the table.

Table 9.2 shows the results of a similar study using SMS. As the maximum p-clique free constraints were beneficial for solving $R(3, 8)$, they are included in the table. For the easier instance, it appears adding further constraints is not that beneficial, although as noted previously, variations of a couple of seconds, perhaps due to external factors, change the interpretation of the fastest method.

Chapter 10

Related work

In this section, we will examine the approaches of some other papers using SAT solvers to find Ramsey numbers. Note that only the first two papers below tackles the same class of Ramsey problem as our paper, and the others are variations.

In April 2024, Thibault and Brown formally proved $R(4,5) = 25$ using a SAT solver [17]. This result was originally proved by McKay and Min in 1992 [37] using unverified computational methods. Thibault and Brown method is based on the method of McKay and Min, although used an interactive theorem prover, generalisations and a SAT solver.

The method relies on gluing various $(3,5;n_1)$-graphs with various $(4,4;n_2)$-graphs. These graphs are known, as mentioned in McKay and Min's paper. Thibault and Brown first define generalisations, which are coloured graphs with some edges uncoloured. A (p,q)-graph generalisation with one uncoloured edge represents the (p,q)-graph where the uncoloured edge can be coloured red or blue. Thibault and Brown constructed exact covers for the sets of $(3,5;n_1)$-graphs and $(4,4;n_2)$-graphs. A set of generalisations $\mathcal{G}*$ is an exact cover for a set of $(p,q;n)$-graphs $\mathcal{G}$ if $\mathcal{G} = \bigcup_{G* \in \mathcal{G}*} G*$. By constructing exact covers for $(3,5;n_1)$-graphs and $(4,4;n_2)$-graphs, and gluing generalistions, rather than graphs, the authors reduced the graph gluing required. For example, there are 1,449,166 $(4,4;12)$-graphs but their cover construction method yielded 26,845 generalisations. The cover construction method for $(p,q;n)$-graphs begins by selecting a graph currently not covered. An edge is randomly selected to be uncoloured, subject to criteria to ensure the cover does not include non-$(p,q;n)$-graphs, and each generalisation is not too large. The authors later developed a simplicity heuristic to estimate how simple two glued graphs would be to solve. This heuristic proved reasonably accurate and the authors used this to select edges that result in low simplicity scores, rather than random selection. The graph gluing problems were solved using a SAT solver, which took over 900 days computational time. The authors estimate 44 years computational time without using generalisations.

In 2013, a soft-constraint approach was used which improved the lower bound of $R(4, 8)$ from 56 to 58 [14]. Soft constraints refer to additional constraints passed to the SAT solver. If the solver returns UNSAT, the soft constraints are scored, and a proportion of the highest-scoring soft constraints are removed. Scores are proportional to the number of direct or indirect conflicts a soft constraint causes. The authors note that different proportions were better for different problems, but found 20% to be a reasonable default value.

In deriving their result, the authors used two sets of soft clauses, zebra soft-constraints (Z-SC) and unit soft-constraints (U-SC). Z-SC are new propositional variables z_d. These variables result in a striped pattern in the adjacency matrix of any SAT result, hence their name. They are defined as $e_{ij} \equiv z_d$ $(0 \leq i < j < n, j - i = d)$ for $1 \leq d \leq n - 1$. Assignment for an e_{ij} can then quickly propagate to e_{kl} where $k - l = j - i$. Thus only $n - 1$ variables need to be determined, as opposed to the $n(n - 1)/2$ variables used to color a graph. We note that unless all soft constraints are removed, the search is not exhaustive. U-SC refers to unit clauses. In the authors' case, they chose unit clauses corresponding to the coloring of a graph without a blue 4-clique and without a red 7-clique on 48 vertices. In their result, the authors note that the removal of the U-SC did not occur, and the remaining edges were colored entirely by Z-SC.

In 2015, the tri-color Ramsey problem $R(4, 3, 3)$ was found, with heavy use of a SAT solver [9]. The tri-color Ramsey problem $R(4, 3, 3)$ searches for the smallest $n \in \mathbb{Z}$ such that any colouring of K_n contains a blue 4-clique, a red triangle or a green triangle. Similar to a (p, q)-graph, define a (p, q, r)-graph as a complete coloured graph without a blue p-clique, a red q-clique or a green r-clique. They use a symmetry-breaking technique [10] and *nauty* [36] to reduce the number of graphs generated. There are two key intermediate results in the authors' approach:

- if a $(4, 3, 3)$-graph exists on 30 vertices, each vertex must be incident to 13 blue edges, 8 red edges, and 8 green edges (denoted as $\langle 13, 8, 8\rangle$ regular)
- finding all 3-colorings on 13 vertices without any monochromatic triangles, i.e., all $(3, 3, 3)$-graphs of order 13

Based on prior theory, the authors knew that on 30 vertices such a $(4, 3, 3)$-graph must be regular in one of the following combinations:

$$\langle 13, 8, 8\rangle, \langle 14, 8, 7\rangle, \langle 15, 7, 7\rangle,$$
$$\langle 15, 8, 6\rangle, \langle 16, 7, 6\rangle, \text{ or } \langle 16, 8, 5\rangle$$

They instantiated a problem by taking valid assignments of combinations of the above $\langle a, b, c\rangle$-regular graphs and inserting them along the main diagonal of a 30×30 matrix. This represents a partial coloring of K_{30} in 3 colors. The remaining edge colors were determined using a SAT solver. After a total run time of over 350 hours, all combinations returned UNSAT except for $\langle 13, 8, 8\rangle$.

To find all $(3, 3, 3)$-graphs of order 13, the authors used a SAT solver in conjunction with degree sequences and degree matrices. A degree sequence of an undirected graph is a non-increasing sequence of its vertex degrees. A degree matrix of a graph of order n and in k-colors is an $n \times k$ matrix where entry (i, j) is the number of j-colored edges on vertex i. The list of all possible degree sequences was shortened to 280 using a tri-color vertex degree constraint. Each degree sequence was encoded into CNF form. This resulted in a set of sub-problems that were easier to solve. 80 of the sequences were valid left-most columns for a degree matrix representing a $(3, 3, 3)$-graph on 13 vertices. The sequences were extended to represent degree matrices, totalling 11,933 degree matrices. Each matrix was combined with their tri-color Ramsey encoding on 13 vertices and passed to a SAT solver to yield 999 matrices for a $(3, 3, 3)$-graph on 13 vertices. The degree sequence and degree matrix section took over over 500 hours of computational time. Using these final matrices, a SAT solver, and *nauty* for graph isomorphism checking, the authors generated all 78,892 distinct canonical form $(3, 3, 3)$-graphs on 13 vertices. Thus combining these with various pairs of canonical graphs on 8 vertices to form partial solutions, they use a SAT solver to show no $(4, 3, 3)$-graphs on 30 vertices exist. The total solving time for this was 128.31 years (running in parallel on 456 threads).

The lower and upper bounds on directed Ramsey graphs were both recently improved to 34 and 47 respectively for $R(7)$ using a SAT solver [39]. For the purposes of this paper, we will only describe details and differences of 'directed' Ramsey numbers where necessary. We define a tournament as an orientation of the complete graph, such that for all pairs of distinct vertices u and v, exactly one of the edges uv or vu is in the tournament. A sub-tournament on k vertices is called transitive if it is a sub-graph of a tournament and for all vertices u, v, and w, the existence of edges uv and vw implies the existence of edge uw, denoted TT_k. The authors described several encodings of the directed $R(7)$ problem, such as employing self-subsuming resolution. They noted this encoding performed better, likely as it maintained arc consistency. To find the lower bound, the authors used a SAT solver. However, the upper bound required considerably more theory and computation. The authors used graph theory techniques to split the problem into several cases based on the degrees of the vertices. Most cases are solved through the use of a SAT solver. All TT_6-free tournaments on order 23, 24, and 25 vertices were cataloged. Along with a SAT solver, they used these to show that no extensions exist without a TT_7 on 47 vertices.

Chapter 11

Formal Verification and Result

The value of the Ramsey number $R(3,8) = 28$ is concluded by obtaining an UNSAT result on the encoding asserting the existence of a 28-vertex $(3,8)$-graph and a SAT result on the encoding asserting the existence of a 27-vertex $(3,8)$-graph. $R(3,8)$ on 28 vertices was found to be UNSAT after 57 hours. A 31 GB DRAT file was generated and verified in 89 hours.

The $R(3,8;27;80)$ problem was solved using the MathCheck parallelised pipeline. This generated 13.7k cubes. Each cube returned UNSAT. A combined 245 days in CPU time were spent cubing, solving and verifying. With parallelisation, this was approximately 2 days. The combined proof files were 1.1TB. This result, combined with aforementioned theory, and the known result $R(3,9) > 35$ yields $R(3,9) = 36$.

All run times in this paper were performed on a CPU with an Intel E5-2683 v4 running at 2.1GHz.

Correctness of proofs is a standing problem in the field of computer-assisted proofs—especially for exhaustive searches. For example, recent work uncovered consistency issues in previous computational searches on Lam's problem—highlighting the difficulty of relying on special purpose search code for nonexistence results [5]. Thus, the correctness of the result is crucially dependent on the encodings and computational tools we use. Results given by the SAT+CAS paradigm can be verified, as the method generates certificates allowing an independent third party to certify the SAT solver's search is indeed exhaustive and also that the learned clauses provided by the CAS are correct. Thus, one only needs to trust the correctness of the proof verifier, rather than the SAT solver or the CAS. Typically proof verifiers are much simpler pieces of software that can be formally checked [33]. Verification was performed using the DRAT-trim proof checker [44] slightly modified to support the addition of trusted clauses [33]. MapleSAT generates a DRAT (deletion, reverse asymmetric tautology) proof file consisting of the clauses learned by the solver. The

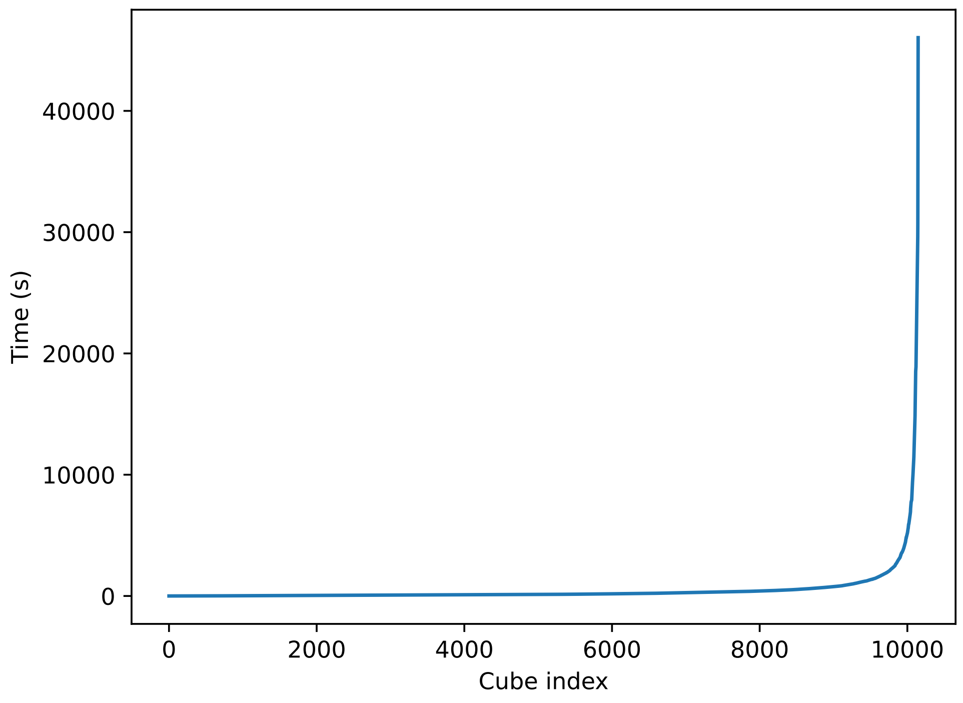

Figure 11.1: Solve time for the 10.1k cubes solved by SAT+CAS. Unsolved cubes were further cubed. Cubes solved during simplification are excluded.

proof checker then verifies whether each clause can be derived from the previous clauses using logically consistent rules. The CAS-derived blocking clauses are verified by evidencing that the clause blocks graphs whose adjacency matrices are not canonical. This is verified by checking a permutation applied to the blocked graph's corresponding adjacency matrix produces a matrix smaller in lexicographical order. The permutation is derived by the CAS during solving and recorded as a witness for the trusted clauses in the DRAT proof.

Chapter 12

Conclusion

Using a CAS, we significantly improve the efficiency of a SAT solver on Ramsey problems and provide the first independently-checkable proof of the result $R(3,8) = 28$ of McKay and Min. We prove and verify $R(3,9) \leq 36$, first solved by Grinstead and Roberts, using a parallelised SAT+CAS, and had a lesser reliance on theory.

SAT+CAS has been demonstrated to be an effective problem solving and verifying tool for Ramsey problems. When combined with domain knowledge to reduce Ramsey problems, the search space can be reduced and thus the effectiveness of SAT+CAS is improved. Future applications of SAT+CAS to Ramsey problems include an automated verification of $R(4,5)$, or solving the unknown values of $R(3,10)$ or $R(4,6)$.

References

[1] Erika Ábrahám. Building bridges between symbolic computation and satisfiability checking. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 1–6, 2015.

[2] Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. volume 2833, pages 108–122, 09 2003.

[3] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[4] George Boole. *The mathematical analysis of logic*. Philosophical Library, 1847.

[5] Curtis Bright, Kevin K H Cheung, Brett Stevens, Ilias Kotsireas, and Vijay Ganesh. A SAT-based resolution of Lam's problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3669–3676, 2021.

[6] Curtis Bright, Jürgen Gerhard, Ilias Kotsireas, and Vijay Ganesh. Effective problem solving using sat solvers. 06 2019.

[7] Curtis Bright, Ilias Kotsireas, and Vijay Ganesh. When satisfiability solving meets symbolic computation. *CACM*, 65(7):64–72, June 2022.

[8] Marcelo Campos, Simon Griffiths, Robert Morris, and Julian Sahasrabudhe. An exponential improvement for diagonal ramsey. *arXiv preprint arXiv:2303.09521*, 2023.

[9] Michael Codish, Michael Frank, Avraham Itzhakov, and Alice Miller. Computing the Ramsey number $R(4,3,3)$ using abstraction and symmetry breaking. *Constraints*, 21:375–393, 2016.

[10] Michael Codish, Alice Miller, Patrick Prosser, and Peter J Stuckey. Constraints for symmetry breaking in graph representation. *Constraints*, 24:1–24, 2019.

[11] David Conlon, Jacob Fox, and Benny Sudakov. Recent developments in graph Ramsey theory. *Surveys in combinatorics*, 424(2015):49–118, 2015.

[12] Paul Erdős. Some remarks on the theory of graphs. 1947.

[13] P. Erdös and A. Hajnal. Ramsey-type theorems. *Discrete Applied Mathematics*, 25(1):37–52, 1989.

[14] Hiroshi Fujita, Miyuki Koshimura, and Ryuzo Hasegawa. SCSat: a soft constraint guided SAT solver. In *Theory and Applications of Satisfiability Testing–SAT 2013: 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings 16*, pages 415–421. Springer, 2013.

[15] Vijay Ganesh, Charles W. O'Donnell, Mate Soos, Srinivas Devadas, Martin C. Rinard, and Armando Solar-Lezama. Lynx: A programmatic SAT solver for the RNA-folding problem. In *SAT 2012*, pages 143–156. Springer Berlin Heidelberg, 2012.

[16] Vijay Ganesh and Moshe Y. Vardi. On the unreasonable effectiveness of SAT solvers. *Beyond the Worst-Case Analysis of Algorithms*, page 547–566, 2021.

[17] Thibault Gauthier and Chad E Brown. A formal proof of $R(4,5) = 25$. *arXiv preprint arXiv:2404.01761*, 2024.

[18] Jan Goedgebeur and Stanisław P Radziszowski. New computational upper bounds for ramsey numbers r (3, k). *arXiv preprint arXiv:1210.5826*, 2012.

[19] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[20] Jack E. Graver and James Yackel. Some graph theoretic results associated with ramsey's theorem. *Journal of Combinatorial Theory*, 4(2):125–175, 1968.

[21] Charles M Grinstead and Sam M Roberts. On the Ramsey numbers $R(3,8)$ and $R(3,9)$. *Journal of Combinatorial Theory, Series B*, 33(1):27–51, 1982.

[22] Marijn J. H. Heule. *Without Loss of Satisfaction*, page 4–14. Springer Nature Switzerland, 2023.

[23] Marijn J H Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *Haifa Verification Conference*, pages 50–65. Springer, 2011.

[24] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding cdcl sat solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing*, pages 50–65, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[25] Marijn JH Heule, Oliver Kullmann, and Victor W Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 228–245. Springer, 2016.

[26] Piyush Jha, Zhengyu Li, Zhengyang Lu, Curtis Bright, and Vijay Ganesh. Alphamaplesat: An mcts-based cube-and-conquer sat solver for hard combinatorial problems, 2024.

[27] Daniela Kaufmann and Armin Biere. Improving AMulet2 for verifying multiplier circuits using SAT solving and computer algebra. *International Journal on Software Tools for Technology Transfer*, 25(2):133–144, 2023.

[28] Markus Kirchweger, Tomáš Peitl, and Stefan Szeider. Co-certificate learning with SAT modulo symmetries. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, August 2023.

[29] Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 34:1–34:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[30] Donald Knuth. *The TEXbook.* Addison-Wesley, Reading, Massachusetts, 1986.

[31] Leslie Lamport. *LATEX — A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[32] Zhengyu Li, Curtis Bright, and Vijay Ganesh. An SC-Square approach to the minimum Kochen–Specker problem. In *SC-Square@FLoC 2022*, 2022.

[33] Zhengyu Li, Curtis Bright, and Vijay Ganesh. A SAT solver and computer algebra attack on the minimum Kochen–Specker problem. *arXiv preprint arXiv:2306.13319*, 2023.

[34] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *SAT 2016*, pages 123–140, 2016.

[35] Brendan D McKay and Zhang Ke Min. The value of the Ramsey number $R(3, 8)$. *Journal of Graph Theory*, 16(1):99–105, 1992.

[36] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[37] Brendan D. McKay and Stanislaw P. Radziszowski. $R(4, 5) = 25$. *Journal of Graph Theory*, 19(3):309–322, 1995.

[38] Shlomo Moran, Marc Snir, and Udi Manber. Applications of Ramsey's theorem to decision tree complexity. *Journal of the ACM (JACM)*, 32(4):938–949, 1985.

[39] David Neiman, John Mackey, and Marijn Heule. Tighter bounds on directed Ramsey number $R(7)$. *Graphs and Combinatorics*, 38(5):156, 2022.

[40] Stanisław Radziszowski. Small Ramsey numbers. *The electronic journal of combinatorics*, 1000:DS1–Aug, 2011.

[41] F. P. Ramsey. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 01 1930.

[42] Fred S Roberts. Applications of Ramsey theory. *Discrete applied mathematics*, 9(3):251–261, 1984.

[43] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *International conference on principles and practice of constraint programming*, pages 827–831. Springer, 2005.

[44] Nathan Wetzler, Marijn J H Heule, and Warren A Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *SAT 2014*, pages 422–429. Springer, 2014.

[45] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM (JACM)*, 28(3):615–628, 1981.

[46] Edward Zulkoski, Vijay Ganesh, and Krzysztof Czarnecki. MathCheck: A math assistant via a combination of computer algebra systems and SAT solvers. In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 607–622. Springer, 2015.