THE DIRECTED RAMSEY NUMBER $R(7)$ IS AT MOST 46

AHMET TAHIR MAT

ABSTRACT. Let $R(k)$ be the least n such that every tournament on n vertices contains a transitive subtournament on k vertices. In 2022 Neiman, Mackey and Heule proved $34 \le R(7) \le 47$. We prove $R(7) \le 46$. The argument looks at the in-degree of a single vertex of a hypothetical counterexample. It uses three computations. (i) Complete lists of the tournaments on 21 to 26 vertices that contain no transitive subtournament on 6 vertices. We obtain them by a SAT-based enumeration in which every job carries a checked LRAT certificate; the lists agree with previously published, uncertified files. (ii) Five new SAT instances on 46 vertices with 504 free arcs each. We solve them by cube-and-conquer; each of the 47,495 leaf cubes is closed by an LRAT proof that is checked while it is produced, and the cube cover is checked as well. (iii) The 28 SAT computations of Neiman, Mackey and Heule. They were published without proofs; we re-ran all of them with the same checked method (32,692 leaf cubes). Every proof was accepted by two checkers, one of which is formally verified. In Lean 4 we derive the theorem from twelve axioms: $R(5) \le 14$ and eleven statements of the computational results above. We list explicitly the parts of the argument that the proof checkers do not cover.

1. INTRODUCTION

A *tournament* is an orientation of a complete graph. It is *transitive* if its vertices can be ordered so that every arc points forward. We write TT_k for the transitive tournament on k vertices. Let $R(k)$ be the least n such that every tournament on n vertices contains TT_k as a subtournament. These numbers go back to Erdős and Moser [7]. The values $R(2), \dots, R(6)$ are $2, 4, 8, 14, 28$; see [14, §2] and the references there, in particular [15] for $R(5) \le 14$ and [16] for $R(6) = 28$.

For $k = 7$, Sanchez-Flores [17] proved $R(7) \le 54$; the lower bound $R(7) \ge 32$ is stated in [10] and [14, §2]. Lidický and Pfender lowered the upper bound to 53 with semidefinite programming [10]. Neiman, Mackey and Heule [14] ("NMH" below) proved $34 \le R(7) \le 47$ with the help of a SAT solver. Lohn, Lambert and Heule [11] later found a further TT_7-free tournament on 33 vertices; the bounds did not change.

NMH's upper bound argument looks at one vertex v of a hypothetical TT_7-free tournament T. Both neighbourhoods of v are TT_6-free. NMH classified the TT_6-free tournaments on 23, 24 and 25 vertices. They then used the SAT solver CaDiCaL [1] to show that certain partial tournaments have no TT_7-free completion. Each such partial tournament consists of a large in-neighbourhood, the vertex v, and a small tournament that must occur in the out-neighbourhood. For $n = 47$ this rules out every in-degree except 22 and 24, and a parity argument rules out those two. For $n = 46$ the in/out-degree pairs $(24, 21)$ and $(23, 22)$ and their reverses survive, and parity says nothing. NMH remark that a catalogue of the TT_6-free tournaments with more than about 20 vertices "could likely be used to further reduce the upper bound on $R(7)$" [14, §6]. We follow that suggestion.

Theorem 1.1. *Every tournament on* 46 *vertices contains* TT_7. *Hence* $34 \le R(7) \le 46$.

Date: Draft, 8 November 2026.

2020 *Mathematics Subject Classification.* Primary 05C55; Secondary 05C20, 68V05, 68V15.

Key words and phrases. Tournament, transitive subtournament, directed Ramsey number, SAT solving, cube-and-conquer, LRAT proof checking.

What the proof uses. The proof (Section 3) excludes every possible in-degree of one vertex. Two catalogue facts drive it. Every TT_6-free tournament on 22 vertices contains a fixed 8-vertex tournament Y_8 that NMH used. Every TT_6-free tournament on 21 vertices contains Y_8, except the copies of one particular tournament, which we call O_{21}^*. With these facts, NMH's 25 instances close the $(23, 22)$ case, and the $(24, 21)$ case reduces to five new SAT instances $\Pi(\mathsf{24X}, O_{21}^*)$, one for each of the five TT_6-free tournaments on 24 vertices. Each new instance has all 46 vertices and 504 free arcs.

We say that an unsatisfiability result is *certified* if the SAT solver emitted a proof in the LRAT format [4] and two checkers accepted it: `lrat-check` from the drat-trim distribution [20], with the patch of Section 5.5, and `cake_lpr` [19], which is formally verified in HOL4 and compiled by the verified CakeML compiler. Every unsatisfiability result used in the proof is certified in this sense:

- the enumeration of the TT_6-free tournaments on 21 to 26 vertices (Section 4), which also re-derives NMH's classifications on 23, 24 and 25 vertices and the uniqueness of the TT_6-free tournament on 26 vertices;
- the five new instances (Section 6.2);
- all 28 computations of NMH that the proof needs (Section 6.3). NMH published their inputs, encoder and solver logs but no proofs.

Every theorem from the literature that the proof uses is re-derived here; for $R(5) \leq 14$ the re-derivation is an exhaustive computation without a proof certificate (Section 4.1). A Lean 4 formalisation proves Theorem 1.1 from twelve axioms, each of which states one of the computational results; it does not check the SAT proofs (Section 7.1). Section 7 lists everything that lies outside the checked part: the mathematical argument where it is not formalised, the encoders, the TT_5-free lists, and our bookkeeping code.

Prior work. A literature and code search carried out on 3 November 2026 covered arXiv, Semantic Scholar and OpenAlex citation data, GitHub, Zenodo, the OEIS and several lists of open problems. Google Scholar, zbMATH and MathSciNet were not searched. The search found no improvement of $34 \leq R(7) \leq 47$ and no earlier statement of the reduction in Section 3. This is a search result, not a claim that nobody has made the observation. The repository of Lohn, Lambert and Heule contains files of TT_6-free tournaments on 20 to 27 vertices [5]; no method or completeness claim is stated for them. Our lists agree with these files for 21 to 26 vertices (Section 4).

Organisation. Section 2 fixes notation. Section 3 proves Theorem 1.1 from a list of named facts. Section 4 describes the enumeration of the catalogues and its certification. Section 5 describes the SAT encodings, the cube-and-conquer runs and the proof checking. Section 6 reports the results. Section 7 describes the Lean formalisation and lists the trust base.

2. Preliminaries

2.1. **Tournaments.** All tournaments are finite. For distinct vertices u, v of a tournament T exactly one of the arcs $u \to v$, $v \to u$ is present; we then say that u *beats* v. For $X \subseteq V(T)$, $T[X]$ is the induced subtournament. T *contains* S if $T[X] \cong S$ for some X. T is TT_k*-free* if it does not contain TT_k. We write $N^+(v)$ and $N^-(v)$ for the out- and in-neighbourhood of v, identified with the subtournaments they induce, and $d^+(v) = |N^+(v)|$, $d^-(v) = |N^-(v)|$. So $d^+(v) + d^-(v) = |V(T)| - 1$. The *reversal* T^R has every arc reversed. A tournament is transitive if and only if it has no 3-cycle.

For tournaments A and B on disjoint vertex sets, $A \Rightarrow B$ is the tournament on $V(A) \cup V(B)$ that contains A and B and in which every vertex of A beats every vertex of B. Let C_3 be the 3-cycle and let RT_5 be the regular tournament on $\mathbb{Z}_5$ in which $i \to i+1$ and $i \to i+2$. Following

NMH we use

$$H_5 = \mathrm{TT}_2 \Rightarrow C_3, \quad G_6 = \mathrm{TT}_1 \Rightarrow \mathrm{RT}_5, \quad H_6 = \mathrm{TT}_3 \Rightarrow C_3, \quad G_7 = \mathrm{TT}_2 \Rightarrow \mathrm{RT}_5.$$

NMH's files call G_7 "W7". The tournament Y_8 is the 8-vertex tournament of [14, Fig. 5]; its adjacency matrix is in Appendix A. It has 15 three-cycles and is not isomorphic to its reversal.

ST_{27} is the Paley tournament on GF(27): $a \to b$ if and only if $b - a$ is a nonzero square. It is TT_6-free. All its subtournaments on 26 vertices are isomorphic, and so are all its subtournaments on 25 vertices (we checked both by computing canonical forms); we call them ST_{26} and ST_{25}. We use NMH's names for the TT_6-free tournaments on 24 and 23 vertices: $\mathsf{24A}, \ldots, \mathsf{24E}$ are the five isomorphism classes of 24-vertex subtournaments of ST_{27}, and $\mathsf{23A}, \ldots, \mathsf{23Y}$ are the 25 classes on 23 vertices (22 subtournaments of ST_{27} and three doubly regular tournaments). The tournament O_{21}^* on 21 vertices is defined by its digraph6 string in Appendix A. It is regular of out-degree 10, has 385 three-cycles and largest transitive subtournament TT_5, is isomorphic to its reversal, and has an automorphism group of order 63. It contains H_6 but neither G_7, Y_8 nor the reversal of Y_8. (None of these properties is used in the proof except that O_{21}^* is a fixed tournament.)

2.2. **Pivot instances.** Let X and Z be tournaments. The *pivot instance* $\Pi(X, Z)$ is the partial tournament on $V(X) \cup \{p\} \cup V(Z)$ in which the arcs inside X and inside Z are those of X and Z, $x \to p$ for $x \in X$, $p \to z$ for $z \in Z$, and the $|X|\,|Z|$ pairs $\{x, z\}$ with $x \in X$, $z \in Z$ are *free*. A *completion* orients every free pair. Files and tables name $\Pi(X, Z)$ as `X-p-Z`. Whether $\Pi(X, Z)$ has a TT_7-free completion depends only on the isomorphism classes of X and Z.

Lemma 2.1. *Let T be TT_7-free and $v \in V(T)$. Then $N^-(v)$ and $N^+(v)$ are TT_6-free, and T^R is TT_7-free. If $X' \subseteq N^-(v)$ and $Z' \subseteq N^+(v)$, then $T[X' \cup \{v\} \cup Z']$ is a TT_7-free completion of a copy of $\Pi(T[X'], T[Z'])$.*

Proof. A TT_6 in $N^-(v)$ together with v as its last vertex would be a TT_7; similarly, v placed before a TT_6 in $N^+(v)$ gives a TT_7. Reversal maps transitive tournaments to transitive tournaments. The last claim holds because every vertex of X' beats v and v beats every vertex of Z', and a subtournament of a TT_7-free tournament is TT_7-free. □

Lemma 2.2. *$R(k) \le 2R(k-1)$ for $k \ge 2$.*

Proof. A tournament on $2R(k-1)$ vertices has a vertex w with $d^+(w) \ge \lceil (2R(k-1) - 1)/2 \rceil = R(k-1)$. So $N^+(w)$ contains TT_{k-1}, and w followed by it is a TT_k. □

The next lemma and its proof are from [14, §5.2.2].

Lemma 2.3 (NMH)**.** *Every tournament on at least 20 vertices contains TT_6, H_6 or G_7.*

Proof. We begin by showing that every tournament S on at least 10 vertices contains TT_5, H_5 or G_6. Some vertex v of S has out-degree at least 5; let N be a set of 5 out-neighbours of v. If $S[N]$ is regular, then $S[N] \cong \mathrm{RT}_5$, so $\{v\} \cup N$ induces G_6. (A regular tournament on five vertices is isomorphic to RT_5: if a beats b, c with $b \to c$ and a is beaten by d, e with $d \to e$, then out-degree 2 forces $c \to d$, $c \to e$, $b \to d$ and $e \to b$, and $a, b, c, d, e \mapsto 0, 1, 2, 3, 4$ is an isomorphism onto RT_5.) Otherwise some $u \in N$ beats at least three vertices of N; let M be three of them. If $S[M]$ is transitive, then $\{v, u\} \cup M$ induces TT_5. Otherwise $S[M] \cong C_3$ and $\{v, u\} \cup M$ induces $\mathrm{TT}_2 \Rightarrow C_3 = H_5$.

Now let T have at least 20 vertices. Some vertex w has $d^+(w) \ge 10$. Apply the claim to $N^+(w)$. Since w beats all of $N^+(w)$, w together with a TT_5, H_5 or G_6 in $N^+(w)$ induces TT_6, $\mathrm{TT}_3 \Rightarrow C_3 = H_6$ or $\mathrm{TT}_2 \Rightarrow \mathrm{RT}_5 = G_7$. □

3. The reduction for 46 vertices

We state the facts the proof uses, then the lemmas. Facts 3.1, 3.2 and 3.3 are proved by the enumeration of Section 4; Facts 3.4 and 3.5 are the SAT results of Section 6.

Fact 3.1. Every TT_6-free tournament on 26 vertices is isomorphic to ST_{26}. This is proved in [17]; see also [14, §2]. Our certified enumeration (Section 4) re-derives it.

Fact 3.2. There is no TT_5-free tournament on 14 vertices; that is, $R(5) \le 14$. This is classical [15]; our catalogue of TT_5-free tournaments (Section 4.1) re-derives it.

Fact 3.3 (catalogues)**.** For $n = 21, \dots, 25$ let $\mathcal{C}_n$ be the set of isomorphism classes of TT_6-free tournaments on n vertices.

(a) $\mathcal{C}_{25} = \{\mathrm{ST}_{25}\}$.
(b) $\mathcal{C}_{24} = \{\mathsf{24A}, \dots, \mathsf{24E}\}$.
(c) $\mathcal{C}_{23} = \{\mathsf{23A}, \dots, \mathsf{23Y}\}$.
(d) Every tournament in $\mathcal{C}_{22}$ contains Y_8.
(e) Every tournament in $\mathcal{C}_{21}$ other than O_{21}^* contains Y_8.

Fact 3.4 (NMH's instances, re-certified here)**.** None of the following 28 pivot instances has a TT_7-free completion: $\Pi(\mathrm{ST}_{26}, \mathrm{TT}_5)$, $\Pi(\mathrm{ST}_{25}, H_6)$, $\Pi(\mathrm{ST}_{25}, G_7)$, and $\Pi(\mathsf{23X}, Y_8)$ for $\mathsf{X} = \mathsf{A}, \dots, \mathsf{Y}$.

Fact 3.5 (new instances)**.** For $\mathsf{X} = \mathsf{A}, \dots, \mathsf{E}$, the instance $\Pi(\mathsf{24X}, O_{21}^*)$ has no TT_7-free completion.

In the rest of this section T is a TT_7-free tournament on 46 vertices and v is a vertex of T. So $d^-(v) + d^+(v) = 45$.

Lemma 3.6. $d^-(v) \le 27$.

Proof. By Fact 3.2 and Lemma 2.2, $R(6) \le 28$. $N^-(v)$ is TT_6-free by Lemma 2.1, so it has at most 27 vertices. □

Lemma 3.7. $d^-(v) \notin \{26, 27\}$.

Proof. Suppose $d^-(v) \in \{26, 27\}$. Let $X' \subseteq N^-(v)$ have 26 vertices. $T[X']$ is TT_6-free, so $T[X'] \cong \mathrm{ST}_{26}$ by Fact 3.1. $d^+(v) = 45 - d^-(v) \ge 18$, so by Fact 3.2 $N^+(v)$ contains a TT_5 on some set Z'. By Lemma 2.1, $\Pi(\mathrm{ST}_{26}, \mathrm{TT}_5)$ has a TT_7-free completion. This contradicts Fact 3.4. □

Lemma 3.8. $d^-(v) \ne 25$.

Proof. Suppose $d^-(v) = 25$. Then $N^-(v) \cong \mathrm{ST}_{25}$ by Fact 3.3(a), and $N^+(v)$ is a TT_6-free tournament on 20 vertices. By Lemma 2.3 it contains H_6 or G_7. By Lemma 2.1, $\Pi(\mathrm{ST}_{25}, H_6)$ or $\Pi(\mathrm{ST}_{25}, G_7)$ has a TT_7-free completion, contradicting Fact 3.4. □

Lemma 3.9. $d^-(v) \ne 23$.

Proof. Suppose $d^-(v) = 23$. By Fact 3.3(c), $N^-(v) \cong \mathsf{23X}$ for some $\mathsf{X} \in \{\mathsf{A}, \dots, \mathsf{Y}\}$. $N^+(v)$ is a TT_6-free tournament on 22 vertices, so it contains Y_8 by Fact 3.3(d). Lemma 2.1 gives a TT_7-free completion of $\Pi(\mathsf{23X}, Y_8)$, contradicting Fact 3.4. □

Lemma 3.10. $d^-(v) \ne 24$.

Proof. Suppose $d^-(v) = 24$. By Fact 3.3(b), $N^-(v) \cong \mathsf{24X}$ for some $\mathsf{X} \in \{\mathsf{A}, \dots, \mathsf{E}\}$. $N^+(v)$ is a TT_6-free tournament on 21 vertices. By Fact 3.3(e) there are two cases.

$N^+(v)$ contains Y_8. Let $X' \subseteq N^-(v)$ have 23 vertices. $T[X']$ is TT_6-free, so by Fact 3.3(c) it is isomorphic to $\mathsf{23Z}$ for some letter $\mathsf{Z} \in \{\mathsf{A}, \dots, \mathsf{Y}\}$. Lemma 2.1 gives a TT_7-free completion of $\Pi(\mathsf{23Z}, Y_8)$, contradicting Fact 3.4.

$N^+(v) \cong O_{21}^*$. Then $T = T[N^-(v) \cup \{v\} \cup N^+(v)]$ is a TT_7-free completion of a copy of $\Pi(\mathsf{24X}, O_{21}^*)$, contradicting Fact 3.5. □

Proof of Theorem 1.1. Suppose T is a TT_7-free tournament on 46 vertices and pick any vertex v. Since $d^-(v)+d^+(v) = 45$, one of the two is at least 23. If $d^+(v) \geq 23$, replace T by T^R; it is TT_7-free by Lemma 2.1, and in it v has in-degree $d^+(v)$. So we may assume $d^-(v) \geq 23$. Lemmas 3.6–3.10 exclude every value $d^-(v) \geq 23$. The lower bound $R(7) \geq 34$ is from [14]. □

Table 1 lists which facts each step uses.

TABLE 1. Facts used by each step of the proof of Theorem 1.1.

Step	Case	Facts used
Lemma 3.6	$d^-(v) \geq 28$	Fact 3.2, Lemma 2.2
Lemma 3.7	$d^-(v) \in \{26, 27\}$	Facts 3.1, 3.2; $\Pi(\mathrm{ST}_{26}, \mathrm{TT}_5)$
Lemma 3.8	$d^-(v) = 25$	Fact 3.3(a), Lemma 2.3; $\Pi(\mathrm{ST}_{25}, H_6)$, $\Pi(\mathrm{ST}_{25}, G_7)$
Lemma 3.9	$d^-(v) = 23$	Fact 3.3(c),(d); $25 \times \Pi(\mathsf{23X}, Y_8)$
Lemma 3.10	$d^-(v) = 24$	Fact 3.3(b),(c),(e); $25 \times \Pi(\mathsf{23X}, Y_8)$; $5 \times \Pi(\mathsf{24X}, O_{21}^*)$
Theorem 1.1	$d^-(v) \leq 22$	reversal (Lemma 2.1)

Remark 3.11. The argument needs one vertex and no counting. For $n = 46$ counting alone could not work: in-degree sequences with 23 vertices of in-degree 21 and 23 of in-degree 24, or 23 of each of 22 and 23, sum to $\binom{46}{2} = 1035$.

Remark 3.12. The catalogue is needed in the case $d^-(v) = 24$. By Lemma 2.3, $N^+(v)$ contains H_6 or G_7, but this is not enough: in exploratory runs (not part of the proof) the solver found TT_7-free completions of $\Pi(\mathsf{24X}, H_6)$ for $\mathsf{X} = \mathsf{A}, \mathsf{B}, \mathsf{C}$. We re-checked each of these 31-vertex tournaments independently: it is a completion of the instance and its largest transitive subtournament has 6 vertices.

4. The catalogues and their certification

4.1. **TT_5-free tournaments.** We built the TT_5-free tournaments on $n \leq 14$ vertices by one-vertex extension. Every TT_5-free tournament on $n+1$ vertices minus a vertex is TT_5-free on n vertices. So the list for $n+1$ is obtained from the list for n by adding a vertex in all 2^n ways, keeping the TT_5-free results, and reducing them to isomorphism classes with nauty's `labelg` [13]. The numbers of classes for $n = 1, \ldots, 14$ are

1, 1, 2, 4, 11, 35, 121, 376, 692, 369, 45, 1, 1, 0.

For $n \leq 10$ the lists equal, class by class, the output of nauty's `gentourng` filtered for TT_5. For $n \leq 13$ they equal the TT_5-free files in NMH's repository [6]. The value 0 at $n = 14$ is Fact 3.2. This computation is exhaustive but it is not certified by a proof checker (Section 7).

4.2. **Pivot decomposition.**

Proposition 4.1. *Let T be a TT_6-free tournament on n vertices and let w be a vertex of maximum out-degree d. Then $\lceil (n-1)/2 \rceil \leq d \leq 13$ and $n - 1 - d \leq 13$. Let A and B be the members of the TT_5-free lists isomorphic to $N^-(w)$ and $N^+(w)$. Then T is isomorphic to a completion of $\Pi(A, B)$ in which every vertex has out-degree at most d.*

Proof. The average out-degree is $(n-1)/2$. $N^-(w)$ and $N^+(w)$ are TT_5-free (Lemma 2.1 with 6 in place of 7), so they have at most 13 vertices by Fact 3.2. Relabel T along isomorphisms $N^-(w) \to A$ and $N^+(w) \to B$; the result is a completion of $\Pi(A, B)$, and d bounds all out-degrees because it is the maximum. □

For each n, a *job* is a triple (d, i, j) with $\lceil (n-1)/2 \rceil \le d \le 13$, A_i in the TT_5-free list on $n-1-d$ vertices and B_j in the list on d vertices. The job asks for all TT_6-free completions of $\Pi(A_i, B_j)$ with all out-degrees at most d. There are 1, 46, 414, 3,086, 17,673 and 167,798 jobs for $n = 26, 25, 24, 23, 22, 21$. For example, for $n = 21$ the values $d = 10, 11, 12, 13$ give $369 \cdot 369$, $692 \cdot 45$, $376 \cdot 1$ and $121 \cdot 1$ jobs. For $n = 26$ the bounds force $d = 13$, and there is one TT_5-free tournament on 12 vertices and one on 13, so there is a single job.

4.3. **The formula of a job.** The variables of the job (d, i, j) are one Boolean variable per free pair $\{x, y\}$, $x \in A_i$, $y \in B_j$, true if and only if $x \to y$, plus auxiliary variables. The clauses are:

(i) the TT_6-exclusion clauses of Section 5.1. With $k = 6$ every 6-set has at most $6! = 720$ orderings, below the limit used (5900), so all clauses are of the "direct" kind;

(ii) cardinality constraints in Sinz's sequential-counter encoding [18]: each $x \in A_i$ beats at most $d - 1 - d^+_{A_i}(x)$ vertices of B_j, and each $y \in B_j$ beats at most $d - d^+_{B_j}(y)$ vertices of A_i.

Every TT_6-free completion with all out-degrees at most d satisfies the formula, with the counter variables set to the partial sums (Lemma 5.1 and [18]).

4.4. **Enumeration and certification.** The enumerator (`src/pivenum.cpp`) solves each job with the CaDiCaL library in incremental mode. After each model it checks that the completed tournament is TT_6-free and respects the degree bound, records it, and adds a *blocking clause* that excludes its assignment of the arc variables.

A job is *certified* when the job formula plus one blocking clause per recorded model is unsatisfiable and this is certified by a checked LRAT proof (`scripts/certify_enum.py`). Certification also rechecks, in separate code, that every recorded model is TT_6-free and that it fits its job: a tournament on n vertices with exactly the fixed arcs of $\Pi(A_i, B_j)$ and all out-degrees at most d (`scripts/audit/modelfix.py`, written apart from the enumerator, with its own digraph6 decoder). A job with a model that does not fit is not certified. A certified job therefore has no completion, in the sense of the job, other than the recorded ones.

For every n we used a shortcut (`scripts/certify_first.py`). In a sample of 168 jobs for $n = 21$ it needed 42% of the solver time of the two-step path. Each job formula is solved once with LRAT output and the proof is checked. If the answer is unsatisfiable and the proof is accepted, the job has no models, and this one checked solve is both its enumeration and its certificate. Otherwise the job is enumerated as above and later certified with its blocking clauses. Table 2 gives the number of jobs closed each way; for $n = 21$ they are 167,633 and 165.

Proposition 4.2. *Assume that the* TT_5*-free lists are complete, that every* TT_6*-free completion of a job with out-degrees at most* d *satisfies the job formula, and that each recorded model agrees with the fixed arcs of its job. If every job for* n *is certified, then every* TT_6*-free tournament on* n *vertices is isomorphic to a recorded model.*

Proof. Let T be TT_6-free on n vertices. By Proposition 4.1, T is isomorphic to a completion T' of $\Pi(A_i, B_j)$ with out-degrees at most d, for some job (d, i, j). The arc assignment of T' extends to a model of the job formula. Since the job formula plus the blocking clauses is unsatisfiable, the arc assignment of T' is blocked, so it equals the arc assignment of a recorded model M. T' and M agree on the fixed arcs and on the free arcs, so $T' = M$. □

4.5. **Classes and checks.** The recorded models are reduced to isomorphism classes with `labelg`. Table 2 gives the numbers. The following checks were made on the class representatives (`scripts/stage12_check.py`).

- For each $n = 21, \ldots, 26$, the set of canonical forms equals the set of canonical forms of the file `tt6free`n`.d6` of [5].

- For $n = 26$ there is one class, and it has the canonical form of ST_{27} minus a vertex, that is, of ST_{26}.
- For $n = 23, 24, 25$ the classes equal NMH's: our ST_{25} (from the Paley construction), our 24A, ..., 24E and 22 sub-ST_{27} classes on 23 vertices (matched to NMH's letters by canonical form), the doubly regular 23W and 23X from NMH's repository (originally from McKay's data [12]), and 23Y, which is the quadratic-residue tournament on $\mathbb{Z}_{23}$.
- All 89 classes on 22 vertices contain Y_8.
- All 17,040 classes on 21 vertices except one contain Y_8, and the exception has the canonical form of O_{21}^*.

Containment of Y_8 is decided by a backtracking search for an induced embedding (`src/tourn.py`). Together with Proposition 4.2 these checks give Facts 3.1 and 3.3. Only one property of `labelg` is needed for this: its output is a relabelling of its input. Equal canonical forms then imply isomorphism, and every recorded model is isomorphic to its class representative. The class counts and the comparison with the other catalogues also rely on canonicity.

TABLE 2. Enumeration of the TT_6-free tournaments on n vertices. "One solve": jobs closed by a single checked solve (they have no models). "With models": jobs with at least one TT_6-free completion; they were enumerated and then certified with blocking clauses. "Models" counts labelled completions. Solver hours are sums of per-job solver wall-clock times: checked solves, enumeration and certification together.

n	jobs	one solve	with models	models	classes	solver h
26	1	0	1	39	1	0.0
25	46	44	2	78	1	0.1
24	414	404	10	390	5	1.0
23	3,086	3,052	34	1,068	25	4.9
22	17,673	17,611	62	2,098	89	260.4
21	167,798	167,633	165	200,677	17,040	1,223.8
Total	189,018	188,744	274	204,350		1,490.1

5. SAT encodings, cube-and-conquer and proof checking

5.1. **The TT_k-exclusion encoding.** Let P be a partial tournament on vertices $0, \ldots, n-1$: each pair is a fixed arc or free. Our encoder (`src/encode.c`; `src/ttenc.h` implements the same construction for the enumerator) produces the following CNF $F_L(P)$ for TT_k-exclusion, with a parameter L.

- *Arc variables.* One variable x_{ij} per free pair $i < j$, true if and only if $i \to j$.
- *k-sets.* For every k-set S that contains no 3-cycle of fixed arcs, list the orderings of S compatible with the fixed arcs, that is, the transitive orientations of S that agree with them. If there are none, S is skipped. If there are at most L, the set of assignments of the free pairs inside S that make S transitive is reduced to its prime implicants by Quine–McCluskey merging, and one clause excluding each prime implicant is emitted ("direct" clauses). If there are more than L, one "3-cycle clause" is emitted: the disjunction, over the triples of S, of a literal saying that the triple is cyclic.
- *3-cycle literals.* For a triple with no free pair the literal is constant. For a triple with exactly one free pair whose fixed arcs form a path $a \to b \to c$, it is the arc literal for $c \to a$; if the two fixed arcs do not form a path, the triple cannot be cyclic and the literal

is false. For a triple with two or more free pairs it is a fresh variable t_{abc} with the clauses $\neg t_{abc} \vee \neg[u \to w] \vee [w \to y]$ for the three rotations (u, w, y) of each of the two cyclic orientations of $\{a, b, c\}$; clauses that are satisfied by fixed arcs are dropped.

Lemma 5.1 (soundness)**.** *Let T be a TT_k-free completion of P. Set x_{ij} to the orientation of $\{i, j\}$ in T and t_{abc} to true if and only if $\{a, b, c\}$ is a 3-cycle in T. This assignment satisfies $F_L(P)$ for every L. Hence, if $F_L(P)$ is unsatisfiable, P has no TT_k-free completion.*

Proof. A clause $\neg t_{abc} \vee \neg[u \to w] \vee [w \to y]$ can only fail if $\{a, b, c\}$ is cyclic and $u \to w$; then the cycle is $u \to w \to y \to u$, so $w \to y$. A direct clause excludes a set of assignments of the free pairs of S all of which make S transitive, because Quine–McCluskey only merges cubes contained in that set. $T[S]$ is not transitive, so its assignment is not excluded. For a 3-cycle clause, $T[S]$ is not transitive, so it has a cyclic triple. That triple is not fixed-cyclic (such sets are skipped), so its literal is an arc literal or a variable t_{abc}, and it is true. □

Lemma 5.1 is a statement about the construction. That the program implements it is a separate matter (Section 7). The encoder was compared with brute force on 60 random partial tournaments with 8 to 10 vertices, $k \in \{4, 5\}$, up to 13 free pairs and $L \in \{1, 3, 10, 5900\}$. The CNF was evaluated on all 74,944 assignments of the free pairs, with the t-variables set as in Lemma 5.1; it agreed with the brute-force answer every time (`tests/test_encode.py`).

5.2. **The five new instances.** Each instance $\Pi(24\mathsf{X}, O_{21}^*)$ was built from the matrices `data/I24_X.txt` and `data/O21star.txt` (Appendix A) in the vertex order: 24X (vertices 0–23), p (24), O_{21}^* (25–45). It was encoded with $k = 7$ and $L = 1$. Every 7-set with exactly one compatible ordering gets one direct clause; all others get a 3-cycle clause. Each formula has 504 arc variables and 10,836 variables t_{abc}, one for every triple with two vertices on one side of the pivot and one on the other. For 24A there are 2,608,200 direct clauses, 6,355,503 3-cycle clauses and 43,344 definition clauses, 9,007,047 clauses and 208,977,552 literals in all. The other four formulas are of the same size (Table 3). The five formulas were built twice, with two builds of the encoder, with identical SHA-256 values; these are recorded in `data/stage3_manifest.json` and in Appendix A, and the cloud driver refused a formula with any other value.

The soundness of these formulas was checked by a separate program, `src/tri_sound.c` (about 270 lines of C, written apart from `src/encode.c` and `src/cnf_sound.c`). It reads the matrix and the CNF, and gives each auxiliary variable a meaning from the clauses in which it occurs negatively: if these contain, besides it, only arc literals on one triple $\{a, b, c\}$, the variable means "$\{a, b, c\}$ is a 3-cycle"; any other pattern is a failure. Under these meanings it checks that every clause holds in every TT_7-free completion: a clause is accepted if it is satisfied by a fixed arc, is a tautology, holds under every orientation of the free pairs among its vertices, or if every completion that falsifies it contains a TT_7 on some 7-set containing its vertices. It also checks the header. It said SOUND for all five formulas, and the cloud driver refused to solve a formula without this verdict. Its tests compare it with brute force on random partial tournaments, including mutated clauses (`tests/test_tri_sound.py`).

5.3. **NMH's instances and the clause checker.** For the 28 instances of Fact 3.4 we did not use our encoder. We used NMH's encoder `encoding/tour-encode.c` [6], compiled as in their Makefile, on NMH's own input matrices for the 27 instances that have a log, with the limit $5040 = 7!$. With this limit every 7-set gets direct clauses and there are no auxiliary variables. For $\Pi(\mathrm{ST}_{26}, \mathrm{TT}_5)$, for which NMH published no input, the input is ST_{27} minus a vertex, then p, then TT_5. For all 27 logged instances, the header, the number of clauses and the size in bytes of our CNF equal those reported in NMH's CaDiCaL logs. NMH's CNF files are not published, so this is evidence that we solve the formulas NMH solved, not a byte-for-byte comparison. The inputs, the encoder source and all 28 CNFs are pinned by SHA-256 in `data/stage4_manifest.json`; the cloud driver rebuilt every CNF and refused any mismatch.

We do not rely on NMH's encoder for soundness. A separate program (`src/cnf_sound.c`, about 230 lines of C) checks, for a partial tournament P and a CNF F over its arc variables, that every clause of F holds in every TT_7-free completion of P. It accepts a clause C if

(a) some literal of C is true on a fixed arc of P (this covers NMH's unit clauses for the fixed arcs, which must agree with P); or

(b) C contains a literal and its negation; or

(c) after the literals that are false on fixed arcs are removed, let α be the orientation of the remaining free pairs that falsifies C; there is a 7-set S containing every endpoint of α such that every orientation of the free pairs of S that extends α makes S transitive.

It also checks the clause count and the variable range against the header. In case (c) every completion that falsifies C contains a TT_7 on S, so C holds in every TT_7-free completion. Hence, if the checker accepts all clauses and F is unsatisfiable, P has no TT_7-free completion, whatever program produced F. The checker accepted all 28 CNFs with no failure. Its tests (`tests/test_cnf_sound.py`) compare it with brute force on 40 random partial tournaments; of 240 mutated clauses, 42 were unsound by brute force, and the checker accepted none of them.

Finally a structure check (`scripts/stage4_instances.py`) confirms that each input is a pivot instance with exactly the $X \times Z$ pairs free, that X is isomorphic to our own representative of the class named by NMH (from our constructions, Section 4), that the 25 classes $\mathsf{23A}, \ldots, \mathsf{23Y}$ are pairwise non-isomorphic, and that Z is isomorphic to our Y_8, H_6, G_7 or TT_5.

5.4. **Cube-and-conquer.** All 33 instances were solved by cube-and-conquer [9] with the same scripts (`scripts/make_cubes.py`, `scripts/run_cubes.py`).

- *Level* 0. The $K = 10$ arc variables that occur most often in the formula are chosen, and the $2^{10} = 1024$ cubes are all sign patterns on them. In each new instance the ten variables are the arcs between one vertex of $\mathsf{24X}$ and ten vertices of O^*_{21}.
- *Conquer.* A cube c is solved as $F \wedge c$ (F plus one unit clause per literal of c) by CaDiCaL with LRAT output and a time limit that depends on the level.
- *Splitting.* A cube that reaches its time limit is replaced by $2^S = 16$ children: the cube extended by all sign patterns on the $S = 4$ most frequent arc variables that it does not fix yet. The children run at the next level. The last level has no time limit.
- *SAT.* A satisfiable cube would stop the run and its model would be checked independently (`scripts/verify_model.py`). No cube was satisfiable.

5.5. **Streamed LRAT checking.** Text LRAT proofs grew at 2 to 6 megabytes per second of solving in our measurements, so proofs for thousands of solver-hours cannot be stored. Every cube and every enumeration job is therefore checked while it is solved (`scripts/stream_solve.py`). The script creates three named pipes. One carries the formula $F \wedge c$ to CaDiCaL. One carries a second copy of the same formula, written by the same code from the same header, clause file and unit clauses, to the checking program. The third carries CaDiCaL's text LRAT proof to the checking program, `scripts/audit/dualcheck.py`. This program reads each of the two streams once and copies it, unchanged and in lock step, to two checkers that run side by side: the patched `lrat-check` and `cake_lpr` (Section 5.7). The checkers thus read the formula itself; they do not take it from the solver. For each proof the program writes a JSON log with both verdicts, the number of bytes each checker received and the last lines of each checker's output.

The verdict used during the runs was that of `lrat-check`, under a strict rule: exit code 0, a line `c VERIFIED`, and no output containing `FAILED`. The patched `lrat-check` prints `c VERIFIED` only if it derived the empty clause, with hints that end in a conflict, and found no error. A cube is recorded as closed only if CaDiCaL reports unsatisfiable and this verdict is `VERIFIED`. The verdict of `cake_lpr` was logged but did not stop a run, so that a failure of `cake_lpr`, such as an exhausted heap,

could not halt a computation of many hours. It counts as acceptance only if `cake_lpr` exits with code 0, prints exactly the line `s VERIFIED UNSAT` and nothing on its error stream (`cake_lpr` exits with code 0 also when a check fails). After the runs an audit (`scripts/audit/union_audit.py`) matches every leaf with the log of its proof and requires that both checkers accepted it. A proof that only `lrat-check` saw was replaced: the same formula was solved again by CaDiCaL and the new proof was checked by both checkers (`scripts/audit/cake_sample.py`). Section 6 gives the counts. Every proof was discarded after it had been checked. Only the per-cube and per-job records (identifier, literals, level, status, verdict and times) and the logs are kept.

`cake_lpr` keeps the parsed formula in a heap of fixed size, about 11 to 13 bytes per byte of formula text. We used 1024 MiB for the enumeration jobs, 2048 MiB for NMH's instances and 14,336 MiB for the new instances, whose formulas have 1.07 GB.

The `lrat-check` patch and the first run. The unpatched `lrat-check` accepts a final empty-clause step whose hints end without a conflict. For example, it accepts the proof `3 0 0` for the satisfiable formula with the clauses $x_1 \vee x_2$ and $\neg x_1 \vee x_2$. This was already known upstream: it is reported in drat-trim issue #42, with a fix proposed in pull request #43 [8]. Our patch (commit `645ee78` of our repository) rejects an empty clause unless its hints end in a conflict, and treats a hint to a clause index outside the clause table as a hint to a deleted clause instead of reading outside the table. `tests/test_lrat_check.py` checks that the patched checker rejects such bogus proofs and accepts genuine CaDiCaL proofs. `cake_lpr` rejects the example as well. A first complete run of all computations ("run 1", 3–5 November 2026) used the unpatched checker. Its checks covered every lemma of every proof except the final empty-clause step, which therefore rested on CaDiCaL. We repeated every computation with both checkers ("run 2", 5–8 November 2026). All numbers in this paper are from run 2. Run 1 is archived in `results/run1/`; its results agree with those of run 2: the same catalogues for $n = 21, \ldots, 25$, and every instance closed with no satisfiable cube.

5.6. **Cover check.** At the end, `scripts/cnc_summary.py` checks three things for each instance.

(1) *Level-0 cover.* The formula $\bigwedge_c \neg c$ over the 1024 level-0 cubes is unsatisfiable. This is shown by CaDiCaL with a checked LRAT proof, from the cubes alone, without the instance formula.

(2) *Splits.* For every cube with children, the children are exactly the 2^S sign patterns on one common sequence of S variables not fixed by the parent, each appended to the parent's literals.

(3) *Leaves.* Every cube without children is recorded as unsatisfiable with checker verdict `VERIFIED`; no cube is open and none is satisfiable.

It also checks that each level-0 record carries exactly the literals of its cube in the cube file, and that the formula that was solved has the SHA-256 recorded for it. It prints `ALL CLOSED` only if all of this holds.

Lemma 5.2. *If (1)–(3) hold, then F is unsatisfiable.*

Proof. Let β be an assignment. By (1), β satisfies some level-0 cube. If a cube satisfied by β has children, then by (2) β satisfies the child whose sign pattern agrees with β on the split variables. The cubes get longer, so this descent ends at a leaf c satisfied by β. By (3), $F \wedge c$ is unsatisfiable, so β falsifies F. □

Lemma 5.2 does not depend on the time limits, on the depth of the splits, or on the order in which cubes were run. This matters for the run history of the new instances in Section 6.2.

5.7. **Tools.** We used CaDiCaL 3.0.1 [1, 2], built from the release tarball (SHA-256 `0a8ea563…7e11`). The checker `lrat-check` is the file `lrat-check.c` of the drat-trim distribution, with the header "Copyright (c) 2017–2023 Marijn Heule" and "last edit July 2023" (SHA-256 `bf07c2ac…4a95`), with our patch of Section 5.5 (patched file: SHA-256 `6ef54b2c…d1ca`). It was compiled with

`gcc -O2 -std=c99 -DLONGTYPE`. The unpatched file is byte-identical to `lrat-check.c` at commit `28c864a` (21 April 2024) of `https://github.com/marijnheule/drat-trim`, the last commit on `master` that changes it. The second checker, `cake_lpr`, is by Tan, Heule and Myreen [19]; we used its release of 1 November 2026 [3]. The checker is verified in HOL4 and compiled by the verified CakeML compiler; we used the compiler's ARMv8 output `cake_lpr_arm8.S` (SHA-256 `50b63e18…852b`), linked with the C wrapper `basis_ffi.c` (SHA-256 `bd498883…cbbc`) by `cc -O2`. Both values equal those published with the release. The build script checks them and runs a self-test, which includes the bogus proof above. The files we used are byte-identical to those at the release tag `cake_lpr/v1.0.0` (commit `033aad5`). According to the README there, the assembly files were built from HOL4 commit `9783a49` and CakeML commit `665f9b2`. nauty 2.9.3 [13] was used for `labelg` and `gentourng` (tarball SHA-256 `9fc4edae…1b5b`). The scripts are in Python 3. The cloud instances ran Ubuntu 24.04 (arm64). The development machine (Apple M5) had the same versions of CaDiCaL and nauty.

6. Results

6.1. **Catalogues.** All jobs for $n = 21, \dots, 26$ were completed and certified (Table 2). The classes equal the files of [5]; for $n = 23, 24, 25$ they also equal NMH's classes, and for $n = 26$ the one class is ST_{26}. Facts 3.1 and 3.3 hold. All 204,350 recorded models fit their jobs; a separate run of `scripts/audit/modelfix.py` over the final files confirms this.

The enumeration ran on one cloud machine from 11:17 UTC on 5 November to 03:31 UTC on 6 November 2026, without interruption. Checking with both checkers started at 12:56 UTC on 5 November. The 14,125 jobs certified before that (all jobs for $n = 26, 25, 24, 23$ and 10,578 jobs for $n = 22$) were checked by `lrat-check` alone. All of them were solved again afterwards, and both checkers accepted every new proof (`results/cake/enum/`). The other 174,893 jobs were checked by both during the run. So every job has a certificate accepted by both checkers. No proof was accepted by one checker and rejected by the other.

6.2. **The five new instances.** All five instances are `ALL CLOSED` (Table 3). The time limits were 600 s at level 0 and 1800 s at level 1, with no limit at level 2. Of the 5×1024 level-0 cubes, 2,295 were closed directly and 2,825 were split into 16 children each. All 45,200 level-1 cubes were closed within the level-1 limit, so level 2 was never used. In total 47,495 leaves were closed by checked proofs; no cube was satisfiable, no check failed, and no leaf is open. This proves Fact 3.5.

Three level-0 cubes (cube 58 of `24A`, 13 of `24B` and 83 of `24C`) were recorded as unknown after 257, 221 and 525 s, before the limit of 600 s. They were split like the cubes that reached the limit; Lemma 5.2 does not depend on why a cube was split.

Run history. The instances ran on cloud spot machines (Section 6.4) from 5 November to 03:36 UTC on 8 November 2026. A first machine with 96 vCPUs checked with `lrat-check` only, until 13:10 UTC on 5 November; from then on every machine checked with both checkers. `cake_lpr` needed about 14 GiB of memory per worker for these formulas, which limited a 96-vCPU machine to 35 workers. The machines were interrupted 15 times by the provider for lack of capacity. After each interruption the affected instances were resumed on another machine from the records synchronised to the development machine; from 7 November the five instances ran on several smaller machines. They closed between 01:39 and 03:36 UTC on 8 November.

The interruptions do not affect correctness. A cube gets a record only when its run has ended: as unsatisfiable with an accepted proof, or as unknown, and then it is split. Work that was running at an interruption left no record and was redone. Lemma 5.2 needs only the checks (1)–(3), which `cnc_summary.py` applies to the final records, whatever machine produced them. A separately written audit (`scripts/audit/closure.py`) repeats them on the synchronised records and finds the cube trees of all 33 instances closed, with no cube recorded twice. Whether a cube reaches its

time limit depends on the machine and its load, so the leaves differ from those of run 1 (48,995 leaves), which used the same level-0 cubes.

Second checker. Of the 47,495 leaves, 47,236 were checked by both checkers during the run. The other 259 (52, 49, 57, 53 and 48 for 24A, ..., 24E) are level-0 leaves from the first machine. They were solved again, with a `cake_lpr` heap of 16,640 MiB, and both checkers accepted every new proof. The per-leaf audit (Section 5.5) finds every leaf checked by both, and no proof accepted by one checker and rejected by the other. The five level-0 cover proofs were also accepted by both.

TABLE 3. The five new instances $\Pi(\mathsf{24X}, O_{21}^*)$ (46 vertices, 504 arc variables, 11,340 variables in all). "Split vertex" is the vertex of 24X (numbered from 0) whose arcs carry the level-0 split. "L0", "L1": level 0, level 1. "Max s" is the longest single cube in seconds.

Instance	clauses	split vertex	L0 closed	L0 split	L1 cubes	leaves	max s	solver h
`24A-p-021*`	9,007,047	21	460	564	9,024	9,484	1,210	541.3
`24B-p-021*`	8,988,168	22	445	579	9,264	9,709	1,312	571.3
`24C-p-021*`	8,988,168	3	481	543	8,688	9,169	1,188	503.3
`24D-p-021*`	9,007,047	8	452	572	9,152	9,604	1,468	530.4
`24E-p-021*`	9,007,047	20	457	567	9,072	9,529	1,360	528.4
Total			2,295	2,825	45,200	47,495		2,674.7

6.3. **NMH's instances.** All 28 instances are `ALL CLOSED` (Table 4), with 32,692 leaves closed by checked proofs, no satisfiable cube and no failed check. This proves Fact 3.4 without relying on NMH's runs.

Run history. The instances ran on the machine of the enumeration (96 vCPUs), from 03:31 UTC on 6 November to 00:47 UTC on 7 November 2026, without interruption: all 28 side by side with 3 workers each, and time limits $(1800, 3600, 1800, 7200, \infty)$ seconds for levels 0 to 4. The deepest cubes are at level 4 (26 literals). The audit of Section 6.2 finds every cube tree closed.

Second checker. Of the 32,692 leaves, 32,684 were checked by both checkers during the run. For the other eight, all at level 4 (solver times 10,533 to 15,885 s, proofs of 4.3 to 7.0 GB), `cake_lpr` stopped because its heap of 2048 MiB was exhausted; `lrat-check` had accepted them. With that heap `cake_lpr` accepted every other proof of these instances, the largest of 3.6 GB. The eight leaves were solved again, and both checkers accepted the new proofs, `cake_lpr` with a heap of 16,384 MiB. The per-leaf audit finds every leaf checked by both, with no disagreement. The 28 cover proofs were also accepted by both.

6.4. **Resources.** All cloud computations of run 2 ran on Amazon Web Services spot instances in the region eu-north-1, with Graviton4 (m8g, r8g) and Graviton3 (r7g) processors. The enumeration and then NMH's instances ran on one m8g.24xlarge with 96 vCPUs. The new instances used 20 machines, in turn or side by side: the first m8g.24xlarge, then three r8g.24xlarge in turn, and from 7 November r8g.16xlarge, r8g.12xlarge, r8g.8xlarge, r7g.12xlarge and r7g.8xlarge machines (`results/aws_ledger.csv`). Solver time was about 1,490 hours for the catalogues, 2,675 hours for the new instances and 1,266 hours for NMH's instances, about 5,430 hours in all. Solving again the proofs that only `lrat-check` had seen took about 200 more hours (161 for the catalogues, 16 for the new instances and 24 for NMH's). These are sums of per-job or per-cube solver wall-clock times, not CPU-hours: the two checkers ran as separate processes beside each solver. The cost of run 2, metered from public spot prices, was US$88.08; with run 1 (US$39.22) the total was US$127.30. Storage volumes and data transfer are not included. For comparison, NMH report about 8 days of CaDiCaL time on average for their 25 instances with Y_8, on one core each [14, §5.2.3]; their logs give 4,801 hours for the 27 logged instances, without proofs.

TABLE 4. NMH's instances, rebuilt with NMH's encoder and re-run with checked LRAT proofs. "Free": free arcs. NMH columns: CaDiCaL version, run time in hours and conflicts in millions, from NMH's logs (no log was published for $\Pi(\mathrm{ST}_{26}, \mathrm{TT}_5)$). "Splits": cubes that timed out and were split; "depth": deepest level.

			NMH (logs)			this work			
Instance	free	clauses	CaDiCaL	h	10^6 confl.	splits	depth	leaves	solver h
ST26-p-TT5	130	1,099,282	–	–	–	0	0	1,024	6.3
ST25-p-H6	150	1,687,063	1.5.2	23.9	88	1	1	1,039	12.0
ST25-p-G7	175	2,603,275	1.5.2	33.5	96	0	0	1,024	13.8
23A-p-Y8	184	2,353,250	1.3.1	121.0	354	12	4	1,204	56.5
23B-p-Y8	184	2,364,374	1.3.1	79.4	265	14	4	1,234	52.4
23C-p-Y8	184	2,364,374	1.3.1	97.5	308	14	4	1,234	52.2
23D-p-Y8	184	2,353,250	1.3.1	157.1	427	9	4	1,159	55.7
23E-p-Y8	184	2,343,727	1.3.1	203.9	543	13	4	1,219	59.6
23F-p-Y8	184	2,343,727	1.3.1	230.9	609	11	4	1,189	58.8
23G-p-Y8	184	2,343,727	1.3.1	212.2	560	13	4	1,219	59.8
23H-p-Y8	184	2,343,727	1.3.1	249.9	650	14	4	1,234	60.9
23I-p-Y8	184	2,353,250	1.3.1	203.7	495	11	4	1,189	52.7
23J-p-Y8	184	2,353,250	1.3.1	240.0	589	9	4	1,159	51.8
23K-p-Y8	184	2,353,250	1.3.1	138.5	383	13	4	1,219	54.6
23L-p-Y8	184	2,343,727	1.3.1	242.6	618	11	4	1,189	58.1
23M-p-Y8	184	2,364,374	1.3.1	94.3	300	12	4	1,204	50.8
23N-p-Y8	184	2,364,374	1.2.1	235.5	390	12	4	1,204	50.3
23O-p-Y8	184	2,353,250	1.2.1	310.1	460	12	4	1,204	55.1
23P-p-Y8	184	2,364,374	1.2.1	198.8	342	12	4	1,204	50.1
23Q-p-Y8	184	2,353,250	1.2.1	247.8	394	12	4	1,204	54.7
23R-p-Y8	184	2,364,374	1.2.1	158.6	293	12	4	1,204	51.4
23S-p-Y8	184	2,353,250	1.2.1	431.6	643	12	4	1,204	56.8
23T-p-Y8	184	2,364,374	1.2.1	164.9	312	12	4	1,204	51.5
23U-p-Y8	184	2,353,250	1.2.1	242.6	401	12	4	1,204	56.4
23V-p-Y8	184	2,353,250	1.2.1	327.4	483	15	4	1,249	55.9
23W-p-Y8	184	2,196,265	1.2.1	68.1	139	0	0	1,024	10.8
23X-p-Y8	184	2,196,265	1.2.1	72.2	144	0	0	1,024	11.1
23Y-p-Y8	184	2,219,145	1.2.1	14.5	48	0	0	1,024	5.9
Total				4,800.7	10,336	268		32,692	1,265.7

7. TRUST BASE

7.1. **The Lean formalisation.** The directory `r7lean/` of the repository contains a Lean 4 project (Lean and Mathlib v4.36.0-rc2). It proves

```
theorem R7.every_tournament_46_has_TT7 (T : Tournament (Fin 46)) :
    T.ContainsTT 7
```

that is, Theorem 1.1, from Lean's three standard axioms and twelve further axioms. Each of the twelve states one computational result: $R(5) \le 14$ (Fact 3.2); six catalogue statements for $n = 21, \ldots, 26$ (Facts 3.1 and 3.3); and five statements that concrete CNFs are unsatisfiable (Facts 3.4 and 3.5): one for $\Pi(\mathrm{ST}_{26}, \mathrm{TT}_5)$, one each for $\Pi(\mathrm{ST}_{25}, H_6)$ and $\Pi(\mathrm{ST}_{25}, G_7)$, one for the 25 instances $\Pi(\mathsf{23X}, Y_8)$, and one for the five new instances. The project has no unfinished proof (`sorry`) and does not use `native_decide`. The following are proved in Lean's kernel:

- the reduction of Section 3, with Lemmas 2.1–2.3 and the reversal step;

- the soundness of Lean versions of both encoders, NMH's with the limit 5040 and ours with $L = 1$: if the instance has a TT_7-free completion, then the CNF is satisfiable. The CNF keeps only clauses that pass a local check, and the check is proved sound;
- the isomorphisms between NMH's labelled inputs and our representatives, given as explicit permutations and checked by `decide`;
- Lemma 5.2, in general form.

A compiled Lean program evaluates the Lean definitions of the 33 CNFs and writes them in DIMACS format. For every instance the output has exactly the SHA-256, the clause count and the byte count recorded in `data/stage3_manifest.json` and `data/stage4_manifest.json`, and no clause failed the local check. This step trusts the Lean compiler and runtime, an unproved writer of about 80 lines, and `shasum`. The Lean project does not check the SAT proofs, does not apply Lemma 5.2 to the actual cube trees, and does not formalise the enumeration (Propositions 4.1 and 4.2). These enter only through the twelve axioms. A reader of the Lean statement must still check that its definitions mean what is intended.

7.2. **What is not checked.** Two proof checkers, one of them formally verified, cover the unsatisfiability of every formula that the proof uses (Section 5.5). The following parts are *not* checked by them. A reader who accepts these items and the checked computations can accept Theorem 1.1.

T1. The mathematical argument. Lemmas 2.1–2.3, the reduction of Section 3, Propositions 4.1 and 4.2, and Lemmas 5.1 and 5.2. They are short. The reduction, Lemmas 2.1–2.3, Lemma 5.2 and the soundness of the 33 pivot formulas are proved in Lean (Section 7.1). Propositions 4.1 and 4.2 and the soundness of the enumeration formulas are checked by hand only.

T2. Results from the literature. None is used without re-derivation. $R(5) \leq 14$ is re-derived by our TT_5-free lists (item T3), $R(6) \leq 28$ from it by Lemma 2.2, NMH's classifications on 23–25 vertices and the uniqueness of the TT_6-free tournament on 26 vertices (Fact 3.1) by our certified enumeration, and NMH's 28 SAT results by our certified runs. Lemma 2.3 has a short proof, which is also checked in Lean.

T3. The TT_5-free lists. Their completeness (Section 4.1) rests on our extension program and on `labelg`; it is not certified. It was cross-checked against `gentourng` for $n \leq 10$ and against NMH's files for $n \leq 13$.

T4. Soundness of the enumeration formulas. That every TT_6-free completion of a job with out-degrees at most d satisfies the job formula. This is Lemma 5.1 plus the correctness of the sequential counter, *as implemented* in `src/ttenc.h` and `src/pivenum.cpp`. These formulas were not checked clause by clause. The evidence is the brute-force tests, a test that the enumerator's TT_k clauses equal those of `src/encode.c`, the reproduction of the TT_5-free lists for $n = 10, \ldots, 13$ by pivot enumeration, and the exact agreement of the final lists with independent sources.

T5. Enumeration bookkeeping. That the job list is complete and that each blocking clause is built from the recorded model. That each recorded model fits its job (its fixed arcs and the degree bound) is checked by the certification step, with a separately written check (`scripts/audit/modelfix.py`); all 204,350 models fit.

T6. Catalogue analysis. The Y_8-containment search and the identification of O_{21}^* (Python, `src/tourn.py`); and the property of `labelg` that its output is a relabelling of its input.

T7. Soundness of the five new formulas. They were produced by our encoder in its 3-cycle mode ($L = 1$). Two checks that do not use `src/encode.c` cover them: `src/tri_sound.c` (Section 5.2) said `SOUND` for all five, and the Lean encoder, whose soundness is proved, reproduces the five files byte for byte (Section 7.1). It suffices to trust one of the two: `src/tri_sound.c` (tested against brute force), or the Lean compiler, runtime and writer.

With `src/tri_sound.c`, the construction of the instance matrices (`src/instances.py`) from `data/I24_X.txt` and `data/O21star.txt` is also trusted; the Lean route builds the instances from these files itself. Both rely on the identification of these files with 24X and O_{21}^* by canonical form (item T6).

T8. Soundness of NMH's formulas. This rests on the clause checker `src/cnf_sound.c` and on the structure check of the inputs, not on NMH's encoder. The Lean version of NMH's encoder is a second, independent route: it reproduces all 28 files byte for byte, its soundness is proved, and the isomorphisms of NMH's inputs with our representatives are checked in Lean.

T9. The LRAT checkers. Every proof that the results rest on was accepted by both checkers (Section 5.5), so it suffices to trust one of them. `cake_lpr` is verified in HOL4 and compiled by the verified CakeML compiler; its C wrapper (command line, file reads, heap allocation), the C compiler, the operating system and the hardware are not verified. The patched `lrat-check` is an unverified C program; a reader who trusts `cake_lpr` does not need it.

T10. The streaming harness and the bookkeeping. That `scripts/stream_solve.py` gives the checkers the same formula $F \wedge c$ as the solver; that `scripts/audit/dualcheck.py` passes the formula and the proof unchanged to both checkers and reads their verdicts correctly; that `scripts/run_cubes.py` records the literals of the cube that was actually solved; the structural split check and leaf tally in `scripts/cnc_summary.py`, which `scripts/audit/closure.py` repeats independently; and the per-leaf audit `scripts/audit/union_audit.py`, which matches each leaf with the log of its proof by the cube identifier in the name of the proof's working directory, or with a record of `scripts/audit/cake_sample.py`. The level-0 cover itself is checked by LRAT, with both checkers.

T11. Proofs are not kept. Each LRAT proof was checked while it was produced and then discarded. The published record consists of per-cube and per-job verdicts and the per-proof logs of both checkers. An independent re-check means re-running the solver, about 3,940 solver-hours for the 33 pivot instances and about 1,490 for the catalogues.

T12. Faults. The runs used ordinary cloud hardware. A fault in a solver run is caught by the checkers. A checker fault that accepts a wrong proof would have to occur in both checkers on the same proof; no proof was accepted by one checker and rejected by the other, apart from the eight heap exhaustions of `cake_lpr` (Section 6.3). A fault in the bookkeeping is not caught, except where the separately written audits would expose it.

The following are *not* trusted: CaDiCaL (its unsatisfiability claims are checked and its models are re-checked), NMH's encoder, NMH's runs and classifications, and the catalogue files of [5], which serve only as a cross-check.

8. Data and code availability

The code, the input data, the result files and the notes are in a public repository, `https://github.com/papexlabs/r7-tournament-ramsey`. An archived snapshot is deposited at Zenodo with DOI 10.5281/zenodo.24050255. The snapshot also holds the per-proof logs of both checkers (`run2-cakelogs.tar.gz`, 271,883 files), which are too large for the repository. The repository contains, for every enumeration job, the completion record and the certificate record; for every cube, its literals, status, checker verdict and times; the records of the proofs that were solved again; the per-instance summaries; the SHA-256 manifests of NMH's instances and of the five new formulas; the archived records of run 1 (`results/run1/`); and the Lean project. It does not contain LRAT proofs (Section 7, item T11) or third-party files other than the checker sources in `tools/`. NMH's repository and the catalogue files of [5] are fetched by `scripts/fetch_refs.sh`; the SHA-256 values of the files we used are in `refs/SHA256SUMS.txt`.

The SHA-256 values of the input matrices of the five new instances are in Appendix A. The main programs are:

- enumeration: `src/pivenum.cpp`, `src/ttenc.h`, `scripts/pivot_enum.py`, `scripts/certify_enum.py`, `scripts/certify_first.py`, `scripts/stage12_check.py`;
- encoding and checking of formulas: `src/encode.c`, `src/instances.py`, `src/cnf_sound.c`, `scripts/stage4_instances.py`;
- cube-and-conquer: `scripts/make_cubes.py`, `scripts/run_cubes.py`, `scripts/stream_solve.py`, `scripts/cnc_summary.py`, and the cloud drivers in `scripts/cloud/`;
- proof checking and audits: `tools/drat-trim/lrat-check.c` (patched), `tools/cake_lpr/`, `scripts/audit/dualcheck.py`, `scripts/audit/cake_sample.py`, `scripts/audit/union_audit.py`, `scripts/audit/closure.py`, `scripts/audit/modelfix.py`, `src/tri_sound.c`;
- Lean formalisation: `r7lean/`;
- tournament utilities: `src/tourn.py`.

9. Use of AI tools

This work was carried out with an AI assistant, Claude (Anthropic; the model Claude Opus 5.5, used through Claude Code), working under the author's direction. The assistant wrote all of the code in the repository: the enumeration and the encoders, the cube-and-conquer and checking scripts, the cloud drivers, the audit tools and the Lean project. It prepared, ran and monitored the cloud computations, including the resumptions after spot interruptions and the re-checks described in Section 6, and it wrote the working notes and the drafts of this paper. Every commit of the repository names the assistant as co-author.

Separate instances of the assistant served as independent auditors. They wrote the soundness checker `src/tri_sound.c` and the checks `scripts/audit/closure.py` and `scripts/audit/modelfix.py`. One of these audits pointed to the gap in `lrat-check` described in Section 5.5, which had already been reported upstream.

The author chose the problem, approved each computational stage and its budget, made the decisions about the runs (among them to repeat all computations with two proof checkers), and reviewed the results and this text. The correctness of the result does not rest on trust in the assistant beyond the trust base of Section 7: every unsatisfiability claim is checked by two proof checkers, one of them formally verified, and the deduction of Theorem 1.1 from the computational facts is checked in Lean (Section 7.1). The AI system is not an author of this paper. The author takes responsibility for its content.

Acknowledgements

We thank David Neiman, John Mackey and Marijn Heule for publishing their code, inputs and solver logs, and Evan Lohn, Chris Lambert and Marijn Heule for publishing their catalogue files. This work would not have been possible without them. We also thank the developers of CaDiCaL, drat-trim, CakeML and `cake_lpr`, and nauty for making their tools freely available.

Competing interests

The author is the founder of Papex Labs, which operates Papex (`https://papex.org`), the research-record service on which the records of these computations are published.

References

1. Armin Biere, *CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT Competition 2018*, Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions, Department of Computer Science Series of Publications B, vol. B-2018-1, University of Helsinki, 2018, pp. 13–14.
2. Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt, *CaDiCaL 2.0*, Computer Aided Verification (CAV 2024), Part I, Lecture Notes in Comput. Sci., vol. 14681, Springer, 2024, pp. 133–152.
3. *`cake_lpr`: a verified checker for LRAT and LPR proofs*, GitHub repository `https://github.com/tanyongkiam/cake_lpr`, Tag `cake_lpr/v1.0.0` (commit `033aad5`) of 1 November 2026, downloaded on 5 November 2026.
4. Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Jr., Matt Kaufmann, and Peter Schneider-Kamp, *Efficient certified RAT verification*, Automated Deduction (CADE 26), Lecture Notes in Comput. Sci., vol. 10395, Springer, 2017, Introduces the LRAT format, pp. 220–236.
5. *digraph_isolators*, GitHub repository `https://github.com/evanlohn/digraph_isolators`, Directory `tt6free_tournaments`, files committed 25 May 2022; accessed 3 November 2026.
6. *Directed-Ramsey: code, inputs and solver logs accompanying* [14], GitHub repository `https://github.com/neimandavid/Directed-Ramsey`, Branch `master`; accessed 3 November 2026.
7. Paul Erdős and Leo Moser, *On the representation of directed graphs as unions of orderings*, Magyar Tud. Akad. Mat. Kutató Int. Közl. **9** (1964), 125–132, MR 168494.
8. haskiindahouse, *drat-trim issue #42, "`lrat-check` says 'VERIFIED' for a proof that proves nothing (regression #31)", and pull request #43, "`lrat-check`: reject an empty lemma that does not end in a conflict"*, `https://github.com/marijnheule/drat-trim/issues/42` and `https://github.com/marijnheule/drat-trim/pull/43`, 2026, GitHub user `haskiindahouse`; both opened on 29 September 2026. The pull request was open and not merged on 8 November 2026.
9. Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere, *Cube and conquer: guiding CDCL SAT solvers by lookaheads*, Hardware and Software: Verification and Testing (HVC 2011), Lecture Notes in Comput. Sci., vol. 7261, Springer, 2012, pp. 50–65.
10. Bernard Lidický and Florian Pfender, *Semidefinite programming and Ramsey numbers*, SIAM J. Discrete Math. **35** (2021), no. 4, 2328–2344.
11. Evan Lohn, Chris Lambert, and Marijn J. H. Heule, *Compact symmetry breaking for tournaments*, Proceedings of the 22nd Conference on Formal Methods in Computer-Aided Design – FMCAD 2022 (Alberto Griggio and Neha Rungta, eds.), TU Wien Academic Press, 2022, pp. 179–188.
12. Brendan McKay, *Digraphs*, `http://users.cecs.anu.edu.au/~bdm/data/digraphs.html`, Data page, cited in [14].
13. Brendan D. McKay and Adolfo Piperno, *Practical graph isomorphism, II*, J. Symbolic Comput. **60** (2014), 94–112.
14. David Neiman, John Mackey, and Marijn J. H. Heule, *Tighter bounds on directed Ramsey number $R(7)$*, Graphs Combin. **38** (2022), no. 5, Paper No. 156, Preprint arXiv:2011.00683v2 (18 May 2022).
15. K. B. Reid and E. T. Parker, *Disproof of a conjecture of Erdős and Moser on tournaments*, J. Combinatorial Theory **9** (1970), no. 3, 225–238, MR 274328.
16. Adolfo Sanchez-Flores, *On tournaments and their largest transitive subtournaments*, Graphs Combin. **10** (1994), no. 2–4, 367–376.
17. ———, *On tournaments free of large transitive subtournaments*, Graphs Combin. **14** (1998), no. 2, 181–200.
18. Carsten Sinz, *Towards an optimal CNF encoding of Boolean cardinality constraints*, Principles and Practice of Constraint Programming (CP 2005), Lecture Notes in Comput. Sci., vol. 3709, Springer, 2005, pp. 827–831.
19. Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen, *`cake_lpr`: Verified propagation redundancy checking in CakeML*, Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2021), Part II, Lecture Notes in Comput. Sci., vol. 12652, Springer, 2021, pp. 223–241.
20. Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt, Jr., *DRAT-trim: Efficient checking and trimming using expressive clausal proofs*, Theory and Applications of Satisfiability Testing (SAT 2014), Lecture Notes in Comput. Sci., vol. 8561, Springer, 2014, pp. 422–429.

Appendix A. Data

Adjacency matrices have entry 1 in row i, column j if and only if $i \to j$ (vertices numbered from 0). These are the matrices of `data/Y8.txt`, `data/H6.txt` and `data/W7.txt`; Y_8 is the matrix of

[14, Fig. 5].

$$Y_8:\ \begin{smallmatrix} 0&0&1&1&1&1&1&1\\ 1&0&0&0&1&0&1&1\\ 0&1&0&0&1&1&0&1\\ 0&1&1&0&0&1&1&0\\ 0&0&0&1&0&1&0&1\\ 0&1&0&0&0&0&1&1\\ 0&0&1&0&1&0&0&0\\ 0&0&0&1&0&0&1&0 \end{smallmatrix} \qquad H_6:\ \begin{smallmatrix} 0&1&1&1&1&1\\ 0&0&1&1&1&1\\ 0&0&0&1&1&1\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1\\ 0&0&0&1&0&0 \end{smallmatrix} \qquad G_7:\ \begin{smallmatrix} 0&1&1&1&1&1&1\\ 0&0&1&1&1&1&1\\ 0&0&0&1&1&0&0\\ 0&0&0&0&1&1&0\\ 0&0&0&0&0&1&1\\ 0&0&1&0&0&0&1\\ 0&0&1&1&0&0&0 \end{smallmatrix}$$

The tournament O_{21}^*, as labelled in `data/O21star.txt`, in nauty's digraph6 format (the adjacency matrix row by row, six bits per character):

```
&T^}?@Z\GEmsQMySCw[xQdZPLFhCtfDQViPXZB_N|ZQSenHEMsakhTUgkUyOxdqIdxCw]dKX{@}?
```

SHA-256 of the input files of the five new instances:

`data/I24_A.txt`	`7e027dfd8cde62f9b866fb05f8e8225449c3e218dd23efa361fe8e59b3b518d1`
`data/I24_B.txt`	`4bbbdb4138e61a164dd583d4a1c4a750504b54a15fe95a230aac5644c7ba37b3`
`data/I24_C.txt`	`eb94d398672fd48006d1319c01c48c0cdafcb23ebb22bdebe3e111368b4dc87c`
`data/I24_D.txt`	`fb94bdde12356261fc2a698c4fe37b8b64bacdecdcae81f68310a6aa6b7a8f2c`
`data/I24_E.txt`	`73a635a9bd95129cc13d50f4b34e311ad45abf97a405d81995653f045cfe7c35`
`data/O21star.txt`	`1823ac5f5d77c1244622be79d66743100a8306c4fc554b6f3db9f09d1a6f641b`

SHA-256 of the five new formulas, built by `src/instances.py` and `encode` with $k = 7$ and $L = 1$ (`data/stage3_manifest.json`; the files have 1.07 GB each):

`24A-p-O21s.tri.cnf`	`b6a67cd97445f2ce7a5a37c56235158a5dd09f8512febcfc52788f9f054f6970`
`24B-p-O21s.tri.cnf`	`cb1402940fa2ab147762605fbd521f8fbe782c0627c35d437e0c8d8a75d5bdc1`
`24C-p-O21s.tri.cnf`	`8c53b09c90b032e087d88085247e5ad13c4d3c3cc25abec4b00f353af4dd738f`
`24D-p-O21s.tri.cnf`	`0facb5da175e9dd3162085f00fbfaf298e48da737299e42d04b5a5941af8397a`
`24E-p-O21s.tri.cnf`	`94489f5dd54b029e7a96fb545ba1d18cb170576d640ccb03a7800023ea681059`

Papex Labs
Email address: ahmet@papex.org