Compact Symmetry Breaking for Tournaments

Evan Lohn, Chris Lambert, and Marijn J.H. Heule
Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
{evanlohn, chrislambert, marijn}@cmu.edu

Abstract—Isolators are a useful tool for reducing the computation needed to solve graph existence problems via SAT. We extend techniques for creating isolators for undirected graphs to the tournament (complete, directed) case, noting several parallels in properties of isolators for the two classes. We further present an algorithm for constructing \(n\)-vertex tournament isolators with \(\Theta(n \log n)\) unit clauses. Finally, we show the utility of our new isolators in computations of tournament Ramsey numbers.

Index Terms—Satisfiability, Symmetry-breaking, Directed-graphs, Tournaments, Isolators.

I. INTRODUCTION

In recent years, SAT solvers have been used to solve several difficult combinatorial problems [1]–[3]. However, naive encodings of SAT problems often include undesired symmetries, i.e. certain matching subsets of variables that result in equivalent subproblems when given equivalent assignments. To prove the original formula unsatisfiable, in the worst case a solver must search through all possible symmetric parts of the problem space, which slows the generation of unsatisfiable proofs unnecessarily. Similarly, while the solver tries to find a satisfying assignment, symmetries in the input formula may cause the solver to effectively re-explore the same part of the search space even after proving the lack of a solution in a symmetric part of the problem.

The most common way of reducing the impact of symmetries in a given formula is by adding a set of new clauses called a Symmetry-Breaking Predicate (SBP) to the formula before solving [4]–[6]. The goal of a SBP is to preserve the satisfiability of the formula while removing from consideration any regions of the search space known to be symmetric to other regions. In this work we focus on SBP’s for graph existence problems, which are problems that can be solved by checking if a graph with a particular structure exists. Solving such problems is an active area of research [7]–[9]. A large class of problem symmetries in graph existence problems naturally results from the existence of isomorphic labeled graphs. These symmetries exist independent of any desired graph property related to graph structure. Rather, they occur because SAT solvers must search the space of labeled graphs in order to prove the (non-)existence of an unlabeled graph. A SBP that targets graph isomorphisms is known as an isolator. Isolators that break many symmetries with few clauses are most useful in practice, as SAT solvers generally take longer to solve formulas with more clauses. Such isolators are often described as “short”, “small”, or “compact.”

Prior work has shown that it is possible to generate small isolators for undirected graphs [10]. The present work instead handles the generation of short isolators for tournaments: complete, directed graphs. There are several mathematically interesting questions one can ask about tournaments that motivate the generation of tournament isolators. For example, Sumner’s conjecture and various election models in social choice theory rely on tournament properties [11], [12]. Tournament isolators can also aid in the search for doubly-regular tournaments. Doubly-regular tournaments are a class of tournaments that (among many other properties) can be efficiently transformed to skew-symmetric Hadamard matrices [13], which have a wide array of practical uses. However, the most well-known question about tournament structure is the Tournament Ramsey number problem, an analog to Ramsey numbers [14] that asks the question of “in what size tournament \(n\) must a transitive subtournament of size \(k\) exist.” A (sub)tournament is transitive if it contains no cycles. Calculating the tournament Ramsey number for \(k = 7\) is likely the limit of currently known techniques, and doing so would be impactful for the mathematical community.

The first contribution of this work is the generation of compact tournament isolators that asymptotically match the search space reduction of a perfect isolator. Second, we present a methodology for the generation of compact isolators for small tournaments that extends prior work on undirected tournaments [10]. Finally, we demonstrate the practical usage of our small isolators for finding larger graphs relevant to the search for tournament Ramsey numbers.

II. PRELIMINARIES

We define the following common concepts from SAT literature: A literal is either a variable or a negated variable. We use \(\neg\) to denote negation. A clause is a disjunction of literals. A unit clause (sometimes referred to as simply a unit) is a clause containing exactly one literal. A Conjunctive Normal Form (CNF) formula is a conjunction of clauses. Unless otherwise specified, “formula” refers to “CNF formula.” An assignment \(\alpha\) is a function from variables to truth values (True/False). \(\alpha\) satisfies a formula \(F\) if the boolean function denoted by \(F\) returns True given the inputs specified by \(\alpha\).

We also define several graph-theoretic concepts. A tournament \(G = (V, E)\) is a complete directed graph; more formally, \(\forall (v_1, v_2) \in V \times V, v_1 \neq v_2 \rightarrow ((v_1, v_2) \in E) \oplus ((v_2, v_1) \in E)\) and \(\forall v \in V, (v, v) \notin E\), where \(\oplus\) is the XOR operation. The phrase “\(G\) is an \(n\)-vertex tournament” means \(|V| = n\). Given
an \( n \)-vertex tournament \( G = (V, E) \) and a permutation \( \pi \) on \( V \), 
\( \pi(G) \) is defined as \( \pi(G) = (V, \{ (\pi(v_1), \pi(v_2)) | (v_1, v_2) \in E \}) \) 
and is colloquially referred to as applying \( \pi \) to \( G \). Two \( n \)-vertex tournaments \( G_1, G_2 \) are isomorphic (written \( G_1 \cong G_2 \)) exactly when there exists a permutation \( \pi \) on the vertices of \( G_1 \) 
such that \( \pi(G_1) = G_2 \). When any such \( \pi \) exists, it is referred to 
as an isomorphism between \( G_1 \) and \( G_2 \). The isomorphism class (also, equivalence class) \( I_G \) of a tournament \( G \) is defined 
as \( I_G = \{ G' | G \cong G' \} \). An automorphism \( \pi \) on a tournament \( G \) 
is any permutation \( \pi \) such that \( \pi(G) = G \). The set of 
automorphisms of \( G \) form a group under function composition. 
This group is referred to as \( Aut(G) \).

III. ISOLATOR NOTATION AND CONCEPTS

To search for a tournament \( G \) satisfying some structural property, we define variables with the semantics “edge \( e \) exists in graph \( G \)” for use in a formula \( F \). We say that \( F \) admits a 
graph \( G' \) exactly when there exists a satisfying assignment to the 
conjunction of \( F \) and the set of unit clauses semantically 
implied by the edges of \( G' \). An isolator for \( n \)-vertex tournaments is a formula \( F \) that admits at least one tournament from 
every equivalence class on \( n \)-vertex tournaments. A perfect 
isolator is an isolator that admits exactly one tournament from 
every equivalence class. A perfect isolator \( F \) is optimal if 
there does not exist a perfect isolator with fewer non-unit clauses than \( F \). A compact or short isolator is not rigorously 
defined. Rather, it describes an isolator with few enough non-
unit clauses to be of practical use in solving SAT problems.

In this work, vertices will be denoted with lowercase letters \( a, b, c, \ldots \) or with \( v_1, v_2, \ldots, v_n \) when an ordering of the 
vertices is relevant. Arcs (directed edges) will be referred to 
with \( (u, v) \), meaning “there is an arc from \( u \) to \( v \).” In our 
construction of isolators, each variable is written in the form 
\( uv \) and has the semantics “arc \( (u, v) \) exists in the graph.” Note 
that the literal \( \neg uv \) therefore means “arc \( (v, u) \) exists in the 
graph.”

A. Short Isolator Examples

Consider the following two labeled 3-vertex tournaments.

![Tournaments](image)

These tournaments represent the only two equivalence 
classes for \( n = 3 \) tournaments: a cycle and a transitive 
tournament. While any combination of a cycle and transi-
tive tournament would suffice to represent both equivalence 
classes, the tournaments chosen above have the interesting 
property of sharing two edges \( ab \) and \( bc \) (colored red). This 
property allows us to produce a short formula that admits both 
graphs:

\[
ab \land bc.
\]

This formula admits exactly one of the two labeled cycles 
and one of the six labeled transitive tournaments on 3 vertices,

![Formula](image)

and does so with the fewest possible clauses. Therefore, \( ab \land bc \)
is a perfect, optimal isolator for \( n = 3 \) tournaments.

Figure 1 displays canonical representatives of all 4 isomor-
phism classes for \( n = 4 \) tournaments. We note that once again 
all highlighted edges have the same edge labels across graphs, 
and all permutations of non-highlighted edges are present. So, 
a short formula that admits exactly the set of graphs in the 
figure is

\[
ab \land bc \land cd \land ad.
\]

While the optimal isolators for \( n = 3, 4 \) are comprised 
entirely of unit clauses, this pattern does not hold for \( n = 5 \). 
Table I contains the number of unit and non-unit clauses for 
our isolators on \( n \leq 8 \) vertices.

B. Comparison of undirected graph and tournament isolators

Although the majority of this work focuses on tournament 
isolators, there are many interesting parallels between undi-
rected and tournament isolators. In an undirected context, the 
existence of edge \( (u, v) \) is denoted by the literal \( uv \), while its 
nonexistence is denoted by the literal’s negation \( \neg uv \). Because 
edgeless and complete graphs are isomorphism classes for any 
\( n \) in the undirected case, every clause of an undirected graph 
isolator containing only arc literals must contain at least one 
positive and one negative literal. These two graphs do not exist 
in the case of tournaments; the closest parallel is transitive 
tournaments. Unlike the set of \( n \)-vertex undirected graphs 
which contains exactly one empty graph and one complete 
graph with \( n! \) automorphisms each, there are \( n! \) isomorphic 
transitive tournaments on \( n \) vertices. It is possible to select 
the particular transitive tournament \( TT \) that an isolator admits 
by ensuring that at least one edge from \( TT \) is present in each 
clause of the isolator. A simple way to do so is ensure each 
clause contains at least one edge \( uv \) s.t. \( u < v \) in vertex 
numbering.

One consequence of undirected isolators requiring at least 
one positive and one negative literal per clause is that undi-
rected isolators have no unit clauses. However, while negat-
ing all literals in an undirected isolator produces another undi-
rected isolator (because the existence and non-existence of an 
edge is symmetric), there is no direct parallel to be found in 
tournaments as edge directionality does not have this property.

Another interesting difference between undirected graphs 
and tournaments is the low number of isomorphism classes 
for tournaments when \( n \) is small (see table I). Intuitively, 
this happens because it is “easier” for tournaments to be 
isoermorphic. The two options for the edge between vertices
u and v in the undirected case are uv existing or not existing. Crucially, an undirected graph G will never be isomorphic to G' constructed by adding or removing an edge of G, which is an operation that can be seen as “flipping” an edge to its other possibility. However, “flipping” an edge of a tournament T by changing the edge’s direction will produce T' ≃ T iff the two vertices u and v of the flipped edge had the same edges to the rest of the graph (the isomorphism is via the permutation that swaps u and v). Although this discrepancy exists for small n, the numbers of isomorphism classes for undirected graphs and tournaments are remarkably close for larger n (see OEIS A000088, A000561 [15]). Therefore, we expect that perfect, optimal isolators for undirected graphs and tournaments will have similar numbers of clauses for larger n.

C. Arc Literal Numbering

Each uv must be assigned a corresponding integer to conform to the commonly used DIMACS CNF format. To do so, we specify a function idxn(u, v) to map each possible arc (u, v) in an n-vertex tournament to a unique integer identifier. Because exactly one of (u, v) and (v, u) must exist for any two vertices u, v, idx must satisfy idx(u, v) = −idx(v, u). To facilitate isolator comparisons across different n, idx also should satisfy the property that idxn(u, v) = idxn+1(u, v). We therefore drop the subscript n when referring to idxn(u, v) in the future, as its value does not depend on n.

In particular, idx is inductively defined as follows for an n + 1-vertex tournament with vertices v1, v2, ..., vn, vn+1. Let K = n(n−1)/2 be the largest output of idx for an n-vertex tournament (implying base case idx(v1, v2) = 1 when n = 2). Applying idx to each of the arcs (v1, vn+1), (v2, vn+1), ..., (vn, vn+1) yields K + 1, K + 2, ..., K + n, respectively. All arcs not included in this definition are of the form (vw, vu) where w > u, and are defined by the earlier mentioned constraint of idx(u, v) = −idx(v, u).

IV. Unit Clauses

In practice, SAT solvers immediately reduce formulas with unit clauses for n-vertex graphs admits at most 2^2-k equivalence class representatives, so in order to admit at least one member of each equivalence class (by the definition of an isolator), the number of units in an isolator must also be asymptotically upper-bounded by n log n.

In the next section, we provide a procedure that achieves this bound.

B. TT-fixing

In situations where we know that every member of the class of n-vertex tournaments contains a TTk (a transitive tournament of size k), we also know that every equivalence class must contain a member with the tournament fixed in some arbitrary position and orientation (i.e. vertices 1 through k in ascending order). Therefore, any formula that fixes (i.e. asserts the existence of) a TTk on the class of n-vertex tournaments is a valid isolator. Because the remaining subset of n − k non-fixed vertices also forms a tournament, further knowledge about the existence of a transitive tournament within the remaining n − k vertices can be used to fix (via units) another transitive tournament within the n − k vertex subtournament. This procedure can be repeated until all vertices of the original tournament are part of some fixed transitive subtournament. Tournament Ramsey numbers provide exactly the required information about the existence of a transitive subtournament. In fact, tournament Ramsey numbers R(k) (when known) provide the largest TTk guaranteed to exist in a tournament of size at least R(k). Therefore, tournament Ramsey numbers (as well as upper bounds, which exist for arbitrarily large n) can be used to iteratively construct large sets of unit clauses for tournament isolators: we will refer to this process as TT-fixing.

TT-fixing is best understood via a small example like figure 2. For an arbitrary 16-vertex tournament G, R(5) = 14 implies that G must contain a TT5 as a subtournament. Therefore, the
arcs between vertices \(v_1\ldots v_5\) can be “fixed” to be a transitive tournament by generating all unit clauses corresponding to those arcs. However, the remaining \(16 - 5 = 11\) vertices of \(G\) also form an arbitrary subtournament \(G'\) on 11 vertices. Because \(R(4) = 8\), a \(TT_4\) is guaranteed to exist in \(G'\), so we can add unit clauses corresponding to the specific location of that \(TT_4\)'s existence in vertices \(v_6\ldots v_9\). The repetition of this procedure down to 1 or 0 remaining vertices is \(TT\)-fixing.

C. \(TT\)-fixing gives \(\Theta(n \log n)\) units

Let \(\text{units}(n)\) be the function that returns the number of units that can be added to an isolator when using the \(TT\)-fixing method on \(n\)-vertex tournaments. Our goal is to prove a lower bound on \(\text{units}(n)\). Unfortunately, exact tournament Ramsey numbers are non-trivial to calculate (only up to \(R(6) = 28\) is known). However, from Erdős and Moser we have that \(R(k) \leq 2^{k-1}\) [18], i.e. that a \(TT_k\) must exist when considering any tournament on \(2^{k-1}\) or more vertices. Erdős and Moser’s bound can thus be used with \(TT\)-fixing to lower-bound \(\text{units}(n)\).

We claim that \(\text{units}(n) \geq \sum_{i=1}^{n} \frac{1}{2} \lceil \log_2(i) \rceil\). We proceed via induction, with step \(n\) depending on step \(n - k\), with \(k = \lceil \log_2(n) \rceil + 1\). The proposition is true for \(n = 1\) because \(0 \geq 0, n = 2\) because \(1 \geq 0.5\). By definition of \(TT\)-fixing, for a graph with \(n\) vertices we have

\[
\text{units}(n) = \frac{k(k-1)}{2} + \text{units}(n-k). \tag{1}
\]

By the inductive hypothesis,

\[
\text{units}(n-k) \geq \sum_{i=1}^{n-k} \lceil \log_2(i) \rceil / 2. \tag{2}
\]

Next, we have that

\[
k \frac{k-1}{2} = k \lfloor \log_2(n) \rfloor / 2 = \sum_{i=n-k+1}^{n} \lfloor \log_2(i) \rfloor / 2 \geq \sum_{i=n-k+1}^{n} \lceil \log_2(i) \rceil / 2. \tag{3}
\]

Combining lower bounds (2) and (3) for the terms of eq. (1) completes the proof:

\[
\text{units}(n) = \frac{k(k-1)}{2} + \text{units}(n-k) \geq \sum_{i=n-k+1}^{n} \lceil \log_2(i) \rceil / 2 + \sum_{i=1}^{n-k} \lceil \log_2(i) \rceil / 2 \tag{4}
\]

\[
= \sum_{i=1}^{n} \lceil \log_2(i) \rceil / 2. \tag{5}
\]

This inequality result directly implies the asymptotic \(n \log n\) bound, because \(\log_2(n!) \in \Theta(n \log n)\).

D. Practical vs Theoretical \(TT\)-fixing units

We first note a useful recurrence relation on tournament Ramsey numbers: \(R(k) \leq 2R(k-1)\).

\(\textbf{Proof}\). Consider an arbitrary vertex \(v\) in an arbitrary tournament \(G\) on \(2R(k-1)\) vertices. \(v\) must have either an out-degree or an in-degree of at least \(R(k-1)\). In either case, consider the subset of at least \(R(k-1)\) vertices pointed to/at by \(v\). This subset must contain some \(TT_{k-1}\) as a subgraph by definition of \(R(k-1)\). However, \(v\) points to or at all vertices in this \(TT_{k-1}\), which demonstrates that a \(TT_k\) comprised of the \(TT_{k-1}\) vertices and \(v\) exists in \(G\).

In Figure 3 the bottom two lines depict the strict lower bound used in the \(n \log n\) units proof (blue), as well as the
actual number of units TT-fixing would provide if we only used the $R(k) \leq 2^{k-1}$ bound from the proof (red). Above that (orange) is the number of units TT-fixing provides given the best currently known Ramsey number bounds. The best known bound on $R(7)$ is $34 \leq R(7) \leq 47$ [19], so the black line describes the best case for how many unit clauses TT-fixing could provide if $R(7) = 34$ was proven. The recurrence relation $R(k) \leq 2R(k-1)$ is what allows even improvements to small Ramsey number bounds to impact the efficacy of TT-fixing for large $n$.

In Figure 4, the top (orange) line is the total number of graphs a SAT solver must search in a tournament existence problem in the absence of an isolator. The bottom (red) line is the number of unlabeled tournaments on $n$ vertices; this is the minimum number of graphs that any brute-force solver must search to solve a tournament existence problem. This data was taken from OEIS sequence A000568 [15], which limits the size of $n$ for which we can make this comparison to $n = 19$. The middle (blue) line shows how many graphs are admitted by a TT-fixing isolator using the best known bounds on tournament Ramsey numbers. As $n$ grows large, the gap between the bottom two lines should grow small as per the $n \log n$ units upper bound proof.

E. Undirected Isolators: Clique-fixing

As mentioned earlier, undirected isolators cannot have unit clauses. Therefore, undirected isolators cannot directly benefit from units via TT-fixing. However, a crossover result for undirected graphs does exist for binary clauses that uses the same ideas as TT-fixing: we term this process clique-fixing. Undirected Ramsey number guarantee the existence of a red or blue colored $k$-clique for graphs with more than $R_u(k)$ vertices ($R_u$ used here for undirected Ramsey numbers). Clique-fixing uses the same iterative process as TT-fixing, but generates the following clauses instead of $TT_k$ units:

$$\{r \lor e, \neg r \lor \neg e | e \in Edges(K_k)\}$$

where $r$ is an auxiliary variable representing the concept “the $k$-clique is red” and $Edges(K_k)$ is the set of edge literals for the complete graph on $k$ vertices. We note that these clauses are “almost” units in the sense that after a solver makes a decision about whether to set $r$ to true or false, $\binom{k}{2}$ edges are set by unit propagation. Therefore, clique-fixing steps reduce the search space by half as much as TT-fixing steps do. Although not the focus of this work, it is plausible that a similar asymptotic optimality analysis could be done for clique-fixing given this small discrepancy. However, undirected Ramsey numbers (necessary for clique-fixing) empirically grow much faster than tournament Ramsey numbers (and also theoretically: $R_u(k) \leq 4R_t(k-1)$), so clique-fixing may not be as practically useful as TT-fixing.

V. Perfect, Optimal Isolator SAT encoding

Unit-based techniques scale to arbitrary $n$, and TT-fixing is “asymptotically perfect” in the sense that for large tournaments, no isolator generation technique can provide more than a non-constant factor of search space reduction over TT-fixing. However, no known perfect isolators for $n > 4$ consist solely of unit clauses. Additionally, it can be practically useful to have an optimal perfect isolator for small tournaments to allow searching via SAT solver for only non-isomorphic (sub-)graphs as efficiently as possible. The practical utility of compact perfect isolators is demonstrated in our own experiments in the later “Tournament Ramsey Graphs” section. In the following sections, we describe our technique for creating perfect, optimal isolators for $n \leq 6$.

A. Basic SAT encoding

We re-implemented and modified the perfect isolator encoding for undirected graphs [10] to be used for tournaments. Formally, we encoded the question “Is there a set of $k$ clauses $C_1, C_2, ... C_k$ that is a perfect isolator for $n$-vertex tournaments?” Decoding a solution to this formula allowed us to produce an $n$-vertex isolator with $k$ clauses.

For the $i$th isolator clause $C_i$ and arc literal $l$, we defined the variable $In(C_i, l)$ to represent “$l$ is in $C_i$.” Then, for each tournament $G$ on $n$ vertices, we define variables $Excludes(G, C_i)$ for $1 \leq i \leq k$ to mean “clause $C_i$ does not admit $G$.” This specification is implemented as follows with a Tseitin encoding [20] to handle the equality and conjunctions:

$$Excludes(G, C_i) \leftrightarrow \bigwedge_{l \in A_G} \neg In(C_i, l). \quad (6)$$

Here $A_G$ is the set of arc literals corresponding to the arcs present in graph $G$. We also define the variable $Canon(G)$ for all graphs $G$, meaning “Graph $G$ is the canonical representative of its isomorphism class $I_G$.” We implement this as follows (again using Tseitin):

$$Canon(G) \leftrightarrow \bigwedge_{i=1}^{k} \neg Excludes(G, C_i). \quad (7)$$
Finally, for each isomorphism class \( I \), we add the following clauses representing “exactly one graph in \( I \) is canonical” to the formula for each isomorphism class \( I \):

\[
\text{ExactlyOne}(\{ \text{Canon}(G) | G \in I \})
\]

Here \text{ExactlyOne} is implemented with an At Most One operation via Sinz encoding [21] and an At Least One via disjunction. Therefore, a satisfying assignment to this formula corresponds to a perfect isolator on \( k \) clauses. If the formula is unsatisfiable for \( k \) and satisfiable for \( k + 1 \), then the perfect isolator with \( k + 1 \) clauses is optimal for the \( n \) in question.

B. Symmetry Breaking

One symmetry in the above encoding is the order of the isolator clauses, as reordering clauses of an expression in CNF does not affect its satisfying assignments. To break this symmetry, we added clauses that ensured a lexicographic ordering of the clauses in the resulting isolator. For every adjacent pair of clauses \( C_i \) and \( C_{i+1} \), we fixed some ordering of every literal that may appear in them \( l_1, l_2, \ldots, l_n \), and then created variables \( e_0, e_1, \ldots, e_n \) where \( e_j \) represents that clauses \( C_i \) and \( C_{i+1} \) are equivalent when considering only the first \( j \) literals. \( e_0 \) is always true, and to maintain the semantics of the other \( e_j \) we added the clauses

\[
e_j \iff (e_{j-1} \land (\text{In}(C_i, l_j) \iff \text{In}(C_{i+1}, l_j)))
\]

via the Tseitin transformation for every \( 1 \leq i < k \) and \( 1 \leq j \leq n \). Then, we enforced a lexicographic ordering by requiring that for every \( j \) such that \( C_i \) and \( C_{i+1} \) were equal up to \( j \), that if clause \( C_i \) contained \( l_j \) then \( C_{i+1} \) must also contain \( l_j \). Explicitly, we added the following requirement via the Tseitin transformation for every \( 1 \leq i < k \) and \( 1 \leq j \leq n \):

\[
e_{j-1} \land \text{In}(C_i, l_j) \implies \text{In}(C_{i+1}, l_j)
\]

and furthermore we required that \( e_n \) is false to ensure a strict ordering. When searching for an isolator with \( k \) clauses, this reduces the search space by a factor of \( k! \) as only one of the \( k! \) permutations of a given distinct set of clauses will be considered.

There is another symmetry in the vertex labeling. For a given isolator, for each literal \( l \) corresponding to arc index \( I_{ab} \), we can change \( l \) to correspond to arc index \( I_{\pi(a),\pi(b)} \) where \( \pi \) is a permutation of vertex labels. The resulting isolator accepts the same graphs that the original did, but under vertex permutation \( \pi \). To break this symmetry, note that any tournament isolator must admit exactly one transitive tournament. So, we choose to admit only the canonical transitive graph with edges of the form \((v_i, v_j)\), \( i < j \). Note that because every edge in this graph goes from a lower numbered vertex to a higher numbered vertex, the corresponding literals in our encoding are all positive. As such, we know that for any isolator, there is a permuted isolator such that every clause has at least one positive literal in each clause. We may add this to our encoding by requiring for all clauses \( C \)

\[
\forall l \in A_p \quad \text{In}(C, l)
\]

with \( A_p \) being the set of all positive literals. When trying to find an isolator for \( n \) vertices, this reduces the search space by a factor of \( n! \) since the solver is guaranteed to only consider isolators for which the canonical transitive graph is the one described above.

C. Encoding Unit Propagation

Under the encoding described above, our solver finds isolators with many large clauses. However, by applying unit propagation it was often possible to reduce clause sizes. This indicated that not only was the solver generating solutions that needed postprocessing, but candidate isolators that were equivalent under unit propagation were being considered multiple times — a sort of symmetry in this problem. To resolve this, we added variables \( \text{Unit}(l) \) representing “literal \( l \) is a unit clause.” We then required that the isolator be already unit-propagated with respect to these literals by adding the requirement

\[
\neg \text{In}(C, l) \lor \neg \text{Unit}(l)
\]

for all clauses \( C \) and literals \( l \). We also had to account for these units excluding graphs in the \( \text{Canon} \) clauses, which were updated to

\[
\text{Canon}(G) \leftrightarrow \bigwedge_{i=1}^{k} \neg \text{Excludes}(G, C_i) \land \bigvee_{l \in A_G} \neg \text{Unit}(\neg l)
\]

Finally, we considered whether to count these special unit literals towards the clause count in determining isolator optimality. As mentioned in the preliminaries, we chose not to do so. When an isolator with units is used in a SAT solver, the units will be instantly eliminated through unit propagation and thus will reduce the complexity of the resulting problem. Therefore, we consider an optimal isolator to not just have the minimal number of clauses, but the minimal number of non-unit clauses. Since units cannot exist in undirected graph isolators (because an undirected graph isolator must admit both the complete and empty graph), this definition of optimality is consistent with the prior work on the undirected case [10]. Note that we only needed to consider positive unit literals as per the vertex-labeling symmetry breaking, which drastically reduced the search space.

VI. ADDITIONAL ISOLATOR GENERATION TECHNIQUES

The following sections describe several miscellaneous techniques, ranging from practical ways to gain slight improvements on prior techniques to possible directions for future research.

A. Incremental Isolators

Prior work has already shown that any isolator for \( n \)-vertex tournaments is also an isolator for \( n+k \)-vertex tournaments for any positive \( k \), and that combining an isolator on \( m \) vertices with an isolator on \( n \) vertices by applying each isolator to a disjoint subset of vertices creates a new isolator on \( m+n \) vertices [22]. Therefore, it is possible to construct perfect isolators for \( n+k \)-vertex tournaments by adding clauses to
any isolator for $n$-vertex tournaments. In particular, our SAT encoding pipeline had the option to ignore graphs that are not admitted by a given set of units. Including the maximal set of units from an $n$-vertex isolator when generating an encoding for $n+1$-vertex isolators reduces the number of tournaments to generate Canon clauses for by a factor of at least $2^n$ because each isolator has at least the units corresponding to a Hamiltonian path. It is worth noting that we do not have any proofs that any of our non-perfect or non-optimal isolators can be extended to an optimal isolator, even when the isolator being extended from is comprised of only unit clauses. However, extending an isolator from an initial set of units can make searching for compact isolators much more efficient.

The technique of combining isolators is useful for creating compact isolators for large $n$. Although TT-fixing guarantees asymptotic optimality, it does not always add the optimal number of units for small $n$. For example, TT-fixing will generate 9 units when processing 8-vertex (sub)tournaments, while an isolator for $n=8$ with 11 units is possible.

B. Probing

In addition to the SAT encoding approach to isolator generation, we also generated isolators using a method from prior work called “random probes” [10]. On a high level, this approach starts with an empty set of clauses and adds randomly generated clauses that preserve at least one member of each equivalence class until the isolator is perfect. There were only two non-superificial changes needed to adapt the prior work on random probes for undirected graphs to the directed case; allowing unit clauses and allowing clauses with only positive literals. While not guaranteed to generate optimal isolators, the strength of this approach is the relative speed with which isolators are generated. This approach also benefited in efficiency from the technique of disallowing clauses with all negative literals and extending isolators from the unit clauses of smaller isolators.

VII. Results

Our experimental results include the sizes of known perfect isolators for small $n$, as well as experiments showing the practical utility of small $n=6,7$ perfect isolators for solving a tournament existence problem. All results and code are available at https://github.com/evanlohn/digraph_isolators.

A. Experimental Setup

Our SAT-based approach to generating isolators rely on the creation of “map” files: text files associating each tournament of size $n$ with a label representing that graph’s isomorphism class. In order to generate a map file for tournaments on $n$ vertices, we began by enumerating all $2^n(n-1)/2$ graphs of size $n$. We converted each graph into an adjacency matrix and then into the “.d6” format specified in the NAUTY handbook, then fed the resulting graphs into the labcf script bundled with the NAUTY tool for graph isomorphisms [23]. labcf produced a file where each graph was converted to the canonical form used by nauty. We gave each canonical form a unique label and outputted the arc (directed edge) indices of each original graph alongside its canonical form.

B. Small Optimal Isolators

Our SAT encoding allowed us to compute optimal isolators up to $n=6$. The SAT solver CaDiCaL [24] solves the two instances required to prove optimality ($k=6,7$ non-unit clauses) within 24 hours. Figures 1 and 6 graphically display optimal, perfect isolators for $n=4,5$ by displaying a graph from each isomorphism class. Figure 5 presents the same image for one of the 56 isomorphism classes for $n=6$. Most of the structure of these isolators can be seen from their unit clauses, which are depicted via red edges in the figures.

Fig. 5. One of the 56 isomorphism class representatives admitted by a particular isolator for 6-vertex tournaments. Red edges are edges fixed by unit clauses of the isolator.

For $n=7$, solving the SAT instance directly became clearly infeasible (taking several days without any signs of progress). However, random probing allowed us to find a perfect isolator for $n=7,8$. Each probe ran in around 10 seconds when restricted to force a positive literal in each clause with the map file reduced by the unit clauses from the next largest isolator. Table I describes the best (fewest non-unit clauses) isolator found for $1 \leq n \leq 8$. Several thousand probes were required to find our best known isolator for $n=7$, while 2 probes were used to find our $n=8$ isolator (each $n=8$ probe required about 2 days to finish). We note that $n=8$ isolators can have up to 11 unit clauses; the $n=8$ isolator in Table I was the shortest perfect isolator we generated via probing.

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Isomorphism classes</th>
<th>Best units</th>
<th>Best non-units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>56</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>456</td>
<td>9</td>
<td>47</td>
</tr>
<tr>
<td>8</td>
<td>6880</td>
<td>10</td>
<td>665</td>
</tr>
</tbody>
</table>

C. Tournament Ramsey Graphs

The known tournament Ramsey numbers are $R(2)=2$, $R(3)=4$, $R(4)=8$, $R(5)=14$, and $R(6)=28$ [25]. Note that in most cases, the next number is two times its predecessor. Recently, the lower and upper bounds for $R(7)$ have been...
improved from $32 \leq R(7) \leq 54$ to $34 \leq R(7) \leq 47$ [19].

The improved lower bound is due to dozens of $TT_7$-free tournament on 33-vertices found using SAT.

McKay [26] extended this set of 33-vertex $TT_7$-free tournaments to 5303 using the following method: generate all 29-vertex subtournaments of known 33-vertex $TT_7$-free tournaments and extend them in all possible ways to 33-vertex $TT_7$-free tournaments. Repeat this procedure until no new 33-vertex $TT_7$-free tournaments are found. Also note that if a tournament has no $TT_k$, then its complement (reversing all arcs) also doesn’t. This can be used to find additional $TT_k$-free graphs as well.

Looking for neighbors and complement graphs is a well-known technique to compute more graphs with a certain property. McKay and Radziszowski used it to compute all known 42-vertex graphs that have no clique of size 5 nor a co-clique of size 5 [27]. They conjecture that this method generated all possible graphs of this type.

For all known Ramsey numbers $R(k)$, there are unique tournaments without a $TT_k$ of size $R(k) - 1$ and $R(k) - 2$. Generalizing this property, if for some $n$ there exists a $k$ with a unique $TT_k$-free tournament on $n$ vertices, then that graph is known as $ST_n$. For example, the unique $TT_6$-free tournaments on 26 and 27 vertices are referred to as $ST_{26}$ and $ST_{27}$ respectively.

Prior to our work, there were 5303 known $TT_7$-free tournaments on 33 vertices, implying that $R(7) \geq 34$. So, either $k = 7$ breaks the pattern of existence of $ST_n$ tournaments, or $R(7) > 34$. We studied the 5303 33-vertex $TT_7$-free tournaments and found that they all have $ST_{26}$ as a subtournament. Moreover, 4952 of them have $ST_{27}$ as a subtournament.

It is the case that any $TT_7$-free tournament on 34 vertices contains at least 1 (up to isomorphism) $TT_7$-free subtournament on 33 vertices. Therefore, enumerating further $TT_7$-free tournaments on 33 vertices is a step towards either finding a $TT_7$-free 34-vertex tournament or proving that no such tournament exists. With this motivation, we explored whether the suite of 5303 was complete or whether there are any other 33-vertex $TT_7$-free tournaments. Our main experimental setup involved finding new members containing $ST_{26}$ but not $ST_{27}$ by solving a CNF formula with a SAT solver, which uses our isolator on 7 vertices. The formula can be described as the union of the following sets of clauses:

1. $\binom{26}{2} = 325$ unit clauses requiring that $ST_{26}$ be present in vertices $v_1$ through $v_{26}$;
2. The perfect isolator for $n = 7$ on the seven remaining vertices $v_27$ through $v_{33}$;
3. A clause blocking each of the 5303 known solutions for each vertex permutation that caused the solution to have $ST_{26}$ in vertices $v_1$ through $v_{26}$ and a graph admitted by the $n = 7$ isolator in vertices $v_27$ through $v_{33}$; and
4. clauses enforcing the “no $TT_7$” condition from [19].

While this formula does not disallow all $ST_{27}$s (i.e. a solution might include an extension from $ST_{26}$ that was not present in the original solution set), it disallows all currently known extensions, including the most common by far 1-vertex extension from $ST_{26}$ to $ST_{27}$. Additionally, the $n = 7$ perfect isolator plays a crucial role for finding new solutions in that without it, the SAT solver could find any tournament equivalent to one of the previously known 33-vertex $TT_7$-free tournaments except for some non-automorphic permutation of the last 7 vertices (which would thus be isomorphic to the previously known solution). All solutions to our formula are non-isomorphic to the original 5303 tournaments.

On the Pittsburgh Supercomputing Center [28], we ran 640 shuffled (clause permuted) versions of the above formula on 640 cores for 6 hours using the Kissat solver [29]. We found three different satisfying assignments. These three solutions represented a single new 33-vertex $TT_7$-free tournament, which is shown in Figure 7. This tournament is special as it is self-complementary; reversing all arcs result in an isomorphic graph. Only a small fraction of tournaments has this self-complementary property [30]. Note that all $ST_n$ graphs have this property by definition. After finding this new tournament, we updated the formula to include the blocking clauses for the new tournaments and its isomorphisms. Kissat did not produce further solutions when using 640 shuffled (clause permuted) version of the updated formula on 640 cores in a day, so it is possible that the formula is simply unsatisfiable.
Future work using our results may lead to further improvements on bounds for the tournament Ramsey number problem.

VIII. CONCLUSIONS

Our techniques allow the generation of isolators with asymptotically optimal numbers of unit clauses, as well as perfect, optimal isolators for $n \leq 6$ and compact isolators for $n = 7.8$ found by random probing. We further demonstrate how small isolators can be effectively used in the search for much larger graphs relevant to tournament existence problems. Future work using our results may lead to further improvements on bounds for the tournament Ramsey number problem.

ACKNOWLEDGEMENTS

This work was partially supported by the Hoskinson Center for Formal Mathematics and the National Science Foundation under grant CCF-2015445. We thank Jeremy Avigad for his comments on earlier drafts and John Mackey for his advice on the graph theoretical parts of the paper.

REFERENCES


