Optimal Symmetry Breaking for Tournaments

Evan Lohn*, Marijn Heule*
*Carnegie Mellon University. *elohn@andrew.cmu.edu, mheule@andrew.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. 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 UNSAT, in the worst case a solver must search through all possible 'symmetric' parts of the problem space, which slows the generation of UNSAT 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 lessening 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. The goal of a SBP is to preserve the satisfiability of the formula while removing regions of the search space known to be symmetric to other regions from consideration. When searching for the existence for a graph with particular structure (there are several open problems asking exactly that **CITE**), a large class of problem symmetries naturally results from the existence of isomorphic labeled graphs. These symmetries exist independent of any desired graph property related to graph structure, and largely result from the fact that SAT solvers are searching 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. An isolator is "trivial" if it admits all possible labeled graphs (i.e. the constant $true$ function), or "perfect" if it admits exactly one graph per equivalence class. Additionally, an isolator is "optimal" if it uses the fewest possible clauses to admit a given number of graphs. An optimal perfect isolator is therefore the shortest list of clauses that admits exactly one member of each equivalence class in the set of graphs being searched over. Having optimal or near-optimal isolators is useful in practice, as SAT solvers generally take longer to solve formulas with more clauses.

Prior work has shown that it is possible to generate small isolators for undirected graphs [1]; this work 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. The most well known is the Tournament Ramsey number problem, an analog to Ramsey numbers that (when solved) answers the question of "at what size tournament n must a transitive sub-tournament of size k appear". Other applications include Sumner's conjecture and various election models in social choice theory (**CITE**).

The first contribution of this work is the generation of compact (relatively few clause) isolators for small tournaments, extending prior work in this direction on undirected tournaments [1]. Second, we present a methodology for the generation of (imperfect) compact tournament isolators that asymptotically matches the search space reduction of a perfect isolator. Finally, we make several comparisons between the isolators for undirected graphs and tournaments that can be used to improve or better understand each class of graph.

II. Preliminaries

Do I need this? If so, there's probably a more precise way of defining these SAT concepts

We define the following common concepts from SAT literature:

1) A "variable" is an object that can be assigned to either True or False
2) A positive or negative "literal" is a shorthand for the predicate that a given variable is assigned True or False, respectively.
3) A "clause" is a disjunction of 0 or more literals.
4) A "unit clause" (sometimes referred to as simply a "unit") is a clause containing exactly one literal.
5) A "CNF formula" is a conjunction of clauses (CNF=Conjunctive Normal Form). Unless otherwise specified, "formula" refers to "CNF formula."
6) An assignment σ is a function from variables to truth values (True/False). σ satisfies a formula F if the boolean function denoted by F returns True given the inputs specified by σ.

We also define some graph-theoretic concepts:

1) 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. V is a set of natural numbers for the purposes of this paper, and the phrase "G is an n-vertex tournament" means $|V| = n$.

2) Given an n-vertex tournament $G = (V, E)$ and a permutation p on V, $p(G)$ is defined as $p(G) = (V, \{(p(v_1), p(v_2))|(v_1, v_2) \in E\})$ and is colloquially referred to as "applying p to G."
3) Two n-vertex tournaments G_1, G_2 are isomorphic (written $G_1 \simeq G_2$ exactly when there exists a permutation p on the vertices of G_1 such that $p(G_1) = G_2$. When any such p exists, it is referred to as an isomorphism between G_1 and G_2.
4) The "equivalence class" (also, "isomorphism class") E_G of a tournament G is defined as $E_G = \{G'|G \simeq G'\}$.
5) An automorphism p on a tournament G is any p such that $p(G) = G$. The set of automorphisms of G form a group under function composition; this group is referred to as $Aut(G)$.

Finally, we define the following isolator-related concepts:

1) Some or all of the variables in a formula F can be defined to have the semantics "edge e exists in graph G". If this is the case, we say that F admits 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.
2) An isolator for n-vertex tournaments is a formula F that admits at least one tournament from each equivalence class on n-vertex tournaments
3) A perfect isolator is an isolator that admits *exactly* one tournament from each equivalence class.
4) A perfect isolator F is "optimal" if there does not exist a perfect isolator with fewer non-unit clauses than F.
5) 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.

III. Methodology for Generating Isolators

A. Generation of Map Files

Our first two approaches 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 labelg script bundled with the NAUTY tool for graph isomorphisms [2]. labelg 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.

Let I_{ab}^n be the index of the arc from a to b in an n-vertex tournament. The output format of the original graph arcs was extensible for higher n, satisfying the property that $I_{ab}^n = I_{ab}^{n+1}$. In particular, with $K = n(n-1)/2$ as the largest edge index for graphs of size n, the edges $(1, n+1), (2, n+1), ...(n, n+1)$ for $n+1$-vertex tournaments were labeled $K+1, K+2, ..., K+n$, respectively. We will drop the superscript n when referring to I_{ab}^n in the future, as its value does not depend on n.

The arc indices described thus far only refer to the arcs in the direction of increasing vertex index, i.e. arc index 2 corresponds to $(1, 3)$ and not $(3, 1)$. We further define $I_{ab} = -I_{ba}$. However, when outputting graphs for the map file we simply exclude all negative arc indices to reduce file size, as the negative edges can be inferred with knowledge of n because tournaments are complete graphs. An example map file line for $n = 5$ tournaments is:

```
11     2 5 7 0
```

The first number in the line (11) is the arbitrary label given to the isomorphism class of the graph described by the rest of the line. The remaining non-zero numbers describe all lexicographically increasing arcs present in the graph. In particular, Arc index 2 represents the arc $(1, 3)$, 5 represents $(2, 4)$, and 7 represents $(1, 5)$. So, all other vertex pairs in this tournament have arcs in lexicographically decreasing order, i.e. vertex 5 is connected to vertices 4 and 3 by the arcs $(5, 4)$, $(5, 3)$. Each line is terminated with a 0, following a convention from CNF.

B. Optimal, Perfect Isolator SAT encoding

We re-implemented and modified the perfect isolator encoding for undirected graphs [1] to be used for tournaments. Formally, we encoded the question "Is there a set of k non-unit clauses and a set of unit clauses whose union is a perfect isolator for n-vertex tournaments". This first section will explain the encoding of the simpler problem: "Is there a set of k clauses whose union is a perfect isolator for n-vertex tournaments", which will be modified to the final problem definition in following sections.

For each isolator clause c and arc literal a, we defined the variable $In(c, a)$ to represent "a is in c". Then, for each graph G from the map file for n vertices, we define a variable $Kills(G, c)$ to mean "clause c does not admit graph G". This specification is implemented as follows with a Tseitin encoding [3] to handle the equality and conjunctions:

$$Kills(G, c) = \bigwedge_{a \in A_G} \neg In(c, a) \tag{1}$$

where 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 from the map file, meaning "Graph G is the canonical representative of its isomorphism class." We implement this as follows (again using Tseitin):

$$Canon(G) = \bigwedge_{c} \neg Kills(G, c) \tag{2}$$

Finally, for each isomorphism class E extracted from the mapfile as a set of graphs, let E_1 be the set $\{Canon(G)|G \in$

$E\}$. We add the following clauses to the formula for each isomorphism class E:

$$AMO(E_1) \wedge \bigvee_{c \in E_1} c \tag{3}$$

Where AMO is the "at most one" operation on a set of clauses, implemented via a Sinz encoding [4]. The disjunction in equation (3) represents an "at least one" operation, so the formula encodes the statement "exactly one graph is canonical in isomorphism class E." Therefore, a satisfying assignment to this problem corresponds to a perfect isolator on k clauses. If the problem is UNSAT for k and SAT for $k+1$, this proves that the perfect isolator with $k+1$ clauses is optimal.

C. Breaking Symmetries in the Encoding

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_1 and c_2, we fixed some ordering of every literal that may appear in them $a_1, a_2, \ldots, a_n$, and then created variables $e_0, e_1, \ldots, e_n$ where e_i represents that clauses c_1 and c_2 are equivalent when considering only the first i literals. e_0 is always true, and to maintain the others we added the clauses

$$e_i = e_{i-1} \wedge (In(c_1, a_i) = In(c_2, a_i))$$

via the Tseitin transformation for every $1 \leq i \leq n$. Then, we enforced a lexicographic ordering by requiring that for every i such that c_1 and c_2 were equal up to i, that if clause c_1 contained a_i then c_2 must also contain a_i. Explicitly, we added the following requirement via the Tseitin transformation for every $1 \leq i \leq n$ and pair of adjacent clauses c_1, c_2

$$e_{i-1} \wedge In(c_1, a_i) \implies In(c_2, a_i)$$

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

Another symmetry exists 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 $I_{p(a),p(b)}$ where p is a permutation of vertex labels. The resulting isolator accepts the same graphs that the original did, but under vertex permutation p. 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 of $1 \to 2 \to 3 \to \cdots \to n$. Note that every edge in this graph goes from a lower numbered vertex to a higher numbered vertex, and thus would be a positive literal in our encoding. 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

$$\bigvee_{a \in A_p} In(c, a)$$

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 $1 \to 2 \to \cdots \to n$ described above.

D. Encoding Unit Propagation

Under the encoding described above, our solver finds isolators where every clause was very large, but could be reduced to a much smaller, cleaner isolator that consisted of many units via unit propagation. 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 'unit clause' variables $Unit(a)$ representing "literal a is a unit clause". We then required that the isolator be already unit-propagated with respect to the 'unit clause' literals by adding the requirement

$$\neg In(c, a) \vee \neg Unit(a)$$

We also had to account for these units killing graphs in the $Canon$ clauses, which were updated to

$$Canon(G) = \bigwedge_{c} \neg Kills(G, c) \wedge \bigvee_{a \in A_G} \neg Unit(\neg a)$$

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, this definition of optimality is consistent with the prior work on the undirected case [1]. Note that we only needed to consider positive unit literals as per the vertex-labeling symmetry breaking, which drastically reduced the search space.

E. Provable Units

While constructing smaller isolators using the techniques above, we opted to manually inspect our results and see what patterns they shared. In doing so, we rediscovered a certain fact from graph theory literature; every tournament contains a hamiltonian path **CITE**. As such, it is always possible to construct an n-vertex isolator by extending from a set of unit clauses describing a hamiltonian path on an n-vertex tournament.

It is worth noting that we do not have a proof that it is always possible to extend an encoded Hamiltonian path to an

optimal isolator. However, building off a set of units can make searching for compact isolators much more efficient. In particular, we modified our map-file generation to optionally ignore graphs that are not admitted by a given set of units. Including these units reduces the map file size and generation time by a factor of 2^{n-1}, which makes map file-based approaches still viable for up to $n = 8$.

Given the utility of unit clauses in isolators, a natural next question would be: can we bound the number of units in n-vertex isolators, either strictly or asymptotically? As it turns out, there is a long-known result from graph theory that implies that there are at most $O(n \log n)$ units possible. By the orbit-stabilizer theorem **need to cite? or just explain more why it applies probably**, the size of the equivalence class of a graph G on n vertices is $\frac{n!}{|Aut(G)|}$, where $Aut(G)$ is the set of distinct automorphisms of G. In 1963 Erdös and Renyi **CITE** proved that as n approaches infinity, the proportion of graphs of size n with with multiple automorphisms approaches 0. Therefore, a proportion of graphs approaching 1 have equivalence classes of size $n!$, so the asymptotic number of equivalence classes is $\frac{2^{\binom{n}{2}}}{n!} \in \Theta(\frac{2^{\binom{n}{2}}}{2^{n \log n}}) = \Theta(2^{\binom{n}{2} - n \log n})$. An isolator with k unit clauses for n-vertex graphs admits at most $2^{\binom{n}{2}-k}$ 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 $n \log n$.

Next, we provide a procedure that achieves this bound.

F. TT-fixing

If a TT_k is known to exist within every member of the class of n-vertex tournaments, each 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 a TT_k on the class of n-vertex tournaments is a valid isolator, i.e. will not remove any equivalence classes. 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 (when known) provide the *largest* transitive subtournament guaranteed to exist in a tournament of a given size. 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.

Instead of using Ramsey numbers to provide a guarantee on the existence of a transitive tournament, one simpler procedure for generating isolator units for n-vertex tournaments is as follows: for all $0 \leq i < n/2 - 1$, vertices $2i$ and $2i + 1$ are symmetric with respect to the remaining vertices. So, all such symmetries can be broken by adding the units corresponding to the arcs $(2i, 2i + 1)$. However, such a procedure yields approximately $n/2$ units, which is quite far from the $n \log n$ bound. The complexity of TT-fixing is justified by the number of units it provides, which we now show to be $\Theta(n \log n)$.

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

Let $units(n)$ 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 $units(n)$. Unfortunately, exact tournament Ramsey numbers are non-trivial to calculate (only up to $R(6) = 28$ is known). However, from Erdös and Moser **CITE** we have that $R(k) \leq 2^{k-1}$, i.e. that a TT_k must exist when considering any tournament on 2^{k-1} or more vertices. Erdös and Moser's bound can thus be used with TT-fixing to lower-bound $units(n)$.

We claim that $units(n) \geq \sum_{i=1}^{n} \frac{1}{2} \lfloor \log_2(i) \rfloor$. We proceed via induction, with step n depending on step $n - k$, with $k = \lfloor \log_2(n) \rfloor + 1$. 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

$$units(n) = \frac{k(k-1)}{2} + units(n-k) \tag{4}$$

By the inductive hypothesis,

$$units(n-k) \geq \sum_{i=1}^{n-k} \lfloor \log_2(i) \rfloor / 2 \tag{5}$$

Next, we have that

$$k\frac{k-1}{2} = k\lfloor \log_2(n) \rfloor / 2 = \sum_{i=n-k+1}^{n} \lfloor \log_2(n) \rfloor / 2 \geq \sum_{i=n-k+1}^{n} \lfloor \log_2(i) \rfloor / 2 \tag{6}$$

Combining the two lower bounds ((5) and (6)) for the terms of eq. (4) completes the proof:

$$\begin{aligned} units(n) &= \frac{k(k-1)}{2} + units(n-k) \\ &\geq \sum_{i=n-k+1}^{n} \lfloor \log_2(i) \rfloor / 2 + \sum_{i=1}^{n-k} \lfloor \log_2(i) \rfloor / 2 \qquad (7) \\ &= \sum_{i=1}^{n} \lfloor \log_2(i) \rfloor / 2 \qquad (8) \end{aligned}$$

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

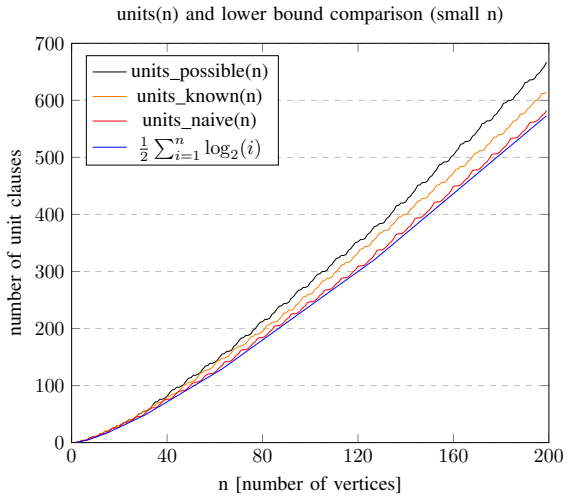

Fig. 1. A visual depiction of how the number of unit clauses produced by TT-fixing grows under different assumptions about ramsey numbers. The bottom two lines depict the strict lower bound used in the $n \log n$ units proof (blue), as well as the actual number of units TT-fixing would provide if we only used the $R(k) \leq 2^{k-1}$ bound (red). Above that (orange) is the number of units TT-fixing provides given the best currently known Ramsey number bounds (this uses purely TT-fixing no Incremental Isolator techniques). Finally, (black) shows how many unit clauses TT-fixing would provide if we proved that $R(7) = 34$ (the best known bound on $R(7)$ is $34 \leq R(7) \leq 47$ [5])

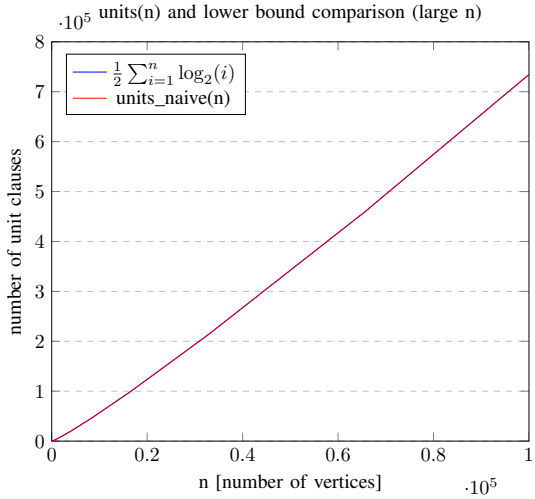

Fig. 2. A depiction of the tightness of the bound derived in the $n \log n$ units proof of TT-fixing as n grows large. For all n in the interval $[2, 1e5)$ the maximum deviation of the two functions was 48, while the average was ≈ 30.1

H. 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 **CITE**. We further note that it is always possible to combine an isolator on m vertices with an isolator on n vertices to create a new isolator on $m + n$ vertices. The method for doing so involves applying each isolator to a disjoint subset of vertices. More formally, given two injective functions $f_m : [1, m] \to [1, m+n]$, $f_n : [1, n] \to [1, m+n]$ s.t the ranges of f_m and f_n are disjoint, each literal representing

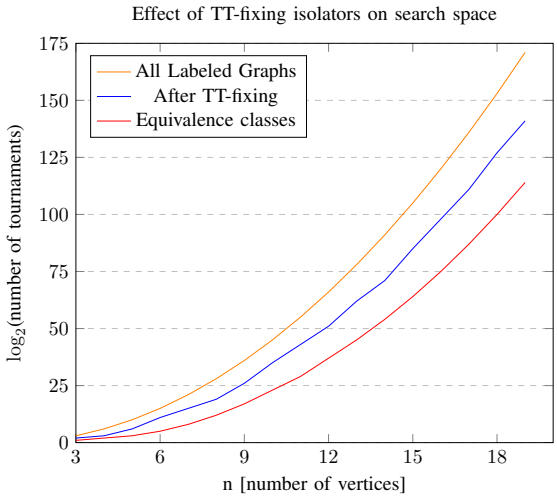

Fig. 3. A depiction of the search space reduction provided by TT-fixing up to $n = 18$ in $\log_2$ space. The top (orange) line is the total number of graphs a SAT solver must search in a graph existence problem in the absence of an isolator. The bottom (red) line is the number of unlabeled tournaments on n vertices; in other words, this is the minimum number of graphs that any brute-force solver must search to solve a graph existence problem. This data was taken from OEIS sequence A000568 **CITE**, which unfortunately limits the size of n for which we can make this comparison to $n = 19$. The middle (blue) line shows how many graphs a SAT solver would need to check in the presence of an isolator created by TT-fixing. As n grows large, the gap between the bottom two lines should grow small relative to the gap between the top two lines as per the $n \log n$ units upper bound proof.

some arc (a, b) in the m-vertex isolator gets transformed to the literal representing arc $(f_m(a), f_m(b))$, and likewise with f_n for the n-vertex isolator. The resulting isolator is valid by the following proof.

Consider an arbitrary $m + n$-vertex tournament g_{mn}; it contains disjoint subtournaments g_m, g_n on m and n vertices selected by considering the ranges of f_m, f_n as defined above. g_m and g_n are admitted by m and n vertex isolators respectively under some permutations p_m, p_n on the ranges of f_m, f_n. Composing p_m and p_n and applying the resulting permutation to g_{mn} produces a graph isomorphic to g_{mn} admitted by the conjunction of the two isolators, proving that every equivalence class has at least one member admitted by said union $\square$.

This technique is practically useful for creating compact isolators for large n. Although TT-fixing guarantees asymptotic optimality, when the procedure reaches the point of generating units for $k = 8$ or fewer remaining vertices, more units can be added by using the units from a known compact isolator for k vertices. For example, TT-fixing will generate 9 units when processing 8-vertex (sub-)tournaments, while our isolator for $n = 8$ contains 11 units.

figure describing the composite process TBD

I. Probing

In addition to the SAT encoding approach to isolator generation, we also generated isolators using a method from prior work called "random probes" [1]. 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-superficial 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.

J. Canonical set techniques

One major critique of our approaches for producing *perfect* isolators is that they do not scale to $n > 9$. Already at n=9 there are 2^{36} labeled tournaments on 9 vertices; this number can be reduced by enforcing all 13 known possible unit clauses during the generation of the map file, but the cnf produced by our SAT-based method is not solvable in reasonable time, and probes for $n = 8$ take ~ 2 days compared to several seconds for $n = 7$. As such, we also experimented with a canonical-set-based approach that scales with the number of equivalence classes.

The canonical-set-based approach is based on the well-known idea that one way to create an isolator is to express a particular set of canonical representatives in disjunctive normal form (DNF), then convert the DNF to CNF. Because this approach may lead to much larger isolators than the optimal ones, we first used the canonical sets induced by our optimal isolators as proofs of concept; the resulting CNFs were equal in length to our optimal isolators up to n=6 (our isolators for n=7 and above are not optimal, and thus were not considered). Next, we tried the naive approach of using an "arbitrary" canonical set (we used the canonical set produced by the **CITE** NAUTY program). The size of the isolators generated was very suboptimal, as expected (see table **TBD** below).

A more promising approach involves modifying a given arbitrary canonical set to have certain properties. Given a canonical set, we detect the hamiltonian path $1 \to 2, 2 \to 3, \ldots n-1 \to n$ via an algorithm modeled on the proof of h-path existence, then permute each member of the canonical set to have that path. Intuitively, procedures such as this one cause the canonical set members to have more edges in common, which will lead to more unit clauses in the final CNF. These experiments gave better results than the naive approach, but (as expected) did not succeed in matching the true optimal isolator length. A possible direction for future work is the development of heuristics for permuting canonical set representatives to produce canonical sets admitting short isolators.

IV. Results

A. Small Optimal Isolators

Our SAT encoding allowed us to compute optimal isolators up to $n = 6$. Figures 4 and 7 graphically display optimal, perfect isolators for $n = 4, 5$ by displaying a graph from each isomorphism class. Figure 6 presents the same image

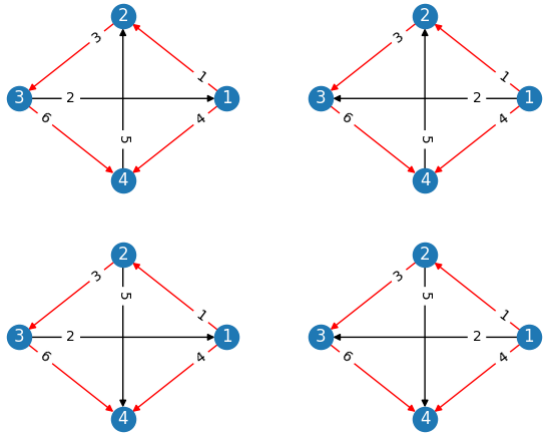

Fig. 4. Isomorphism class representatives admitted by a particular isolator for 4-vertex tournaments. Red edges are edges fixed by unit clauses of the isolator, and the isolator has only unit clauses.

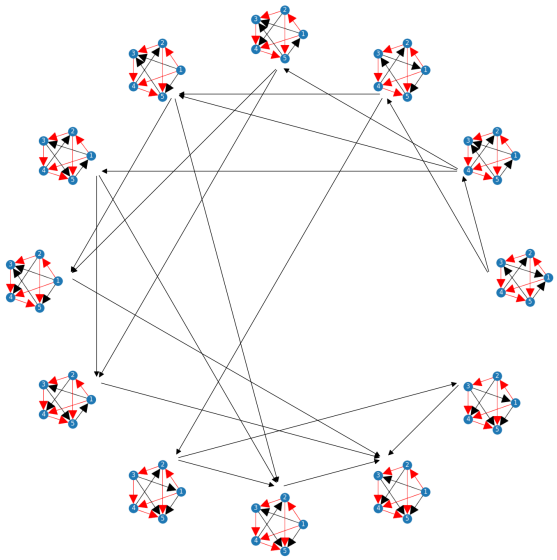

Fig. 5. Flowchart showing one-off relationships between n=5 equivalence classes

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. For $n = 4, 5$ we also computed the number of different perfect optimal isolators up to isomorphism, where we define two isolators to be isomorphic if they are equivalent under some vertex permutation. There is only a single perfect optimal isolator for $n = 4$, while there are 5 different (perfect optimal) isolators for $n = 5$. Interestingly, each of the 5 can be expressed such that they have the same 6 unit clauses, and differ only in the 2 non-unit clauses present in optimal $n = 5$ isolators.

B. Larger Isolators

In scaling up to larger graphs for isolators, we quickly run into barriers in our ability to solve problems in our encoding after $n = 6$. For $n = 7$, solving the SAT instance directly

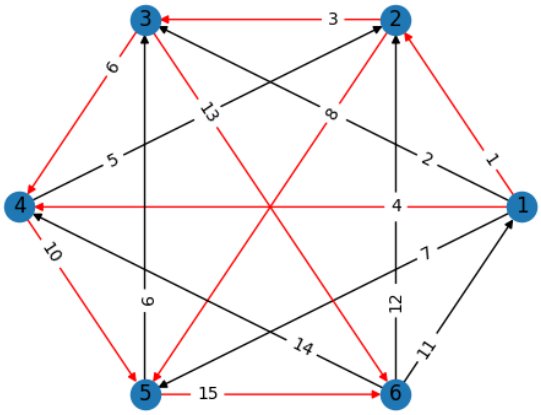

Fig. 6. One of the 56 Isomorphism class representatives admitted by a particular isolator for 6-vertex tournaments. Red edges are edges fixed by unit clauses of the isolator.

became clearly infeasible (taking several days without any signs of progress). Random probing was the only method that allowed us to find an isolator; 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 $n = 6$. Table I describes the best isolator found during the semester for $n = 1$ through 7. Several thousand probes were required to find our best known isolator for $n = 7$.

Vertices	Best units	Best non-units	Isomorphism classes
1	0	0	1
2	1	0	1
3	2	0	2
4	4	0	4
5	6	2	12
6	8	6	56
7	9	47	456

TABLE I

THE SMALLEST ISOLATORS FOUND BY NUMBER OF NON-UNIT CLAUSES FOR ALL VERTEX COUNTS UP TO 7, AS WELL AS THE UNITS IN THAT ISOLATOR. THESE ARE KNOWN OPTIMAL THROUGH 6 VERTICES.

C. Comparison of undirected graph and tournament isolators

1) Global Clause Properties: Because edgeless and complete graphs are isomorphism classes for any n in the undirected case, every clause of an undirected graph isolator must contain at least one positive and one negative literal (assuming no auxiliary variables). These two graphs do not exist in the case of tournaments; the closest parallel is transitive tournaments satisfying $AB \wedge BC \rightarrow AC$ for every three vertices A, B, C in the tournament. Unlike the undirected case (with exactly 1 empty graph and 1 complete graph), there are $n!$ isomorphic transitive tournaments on n vertices. It is possible to select the particular transitive tournament TT 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 AB s.t. $A < B$ in vertex numbering. However, this approach is not yet proven to admit an optimal isolator.

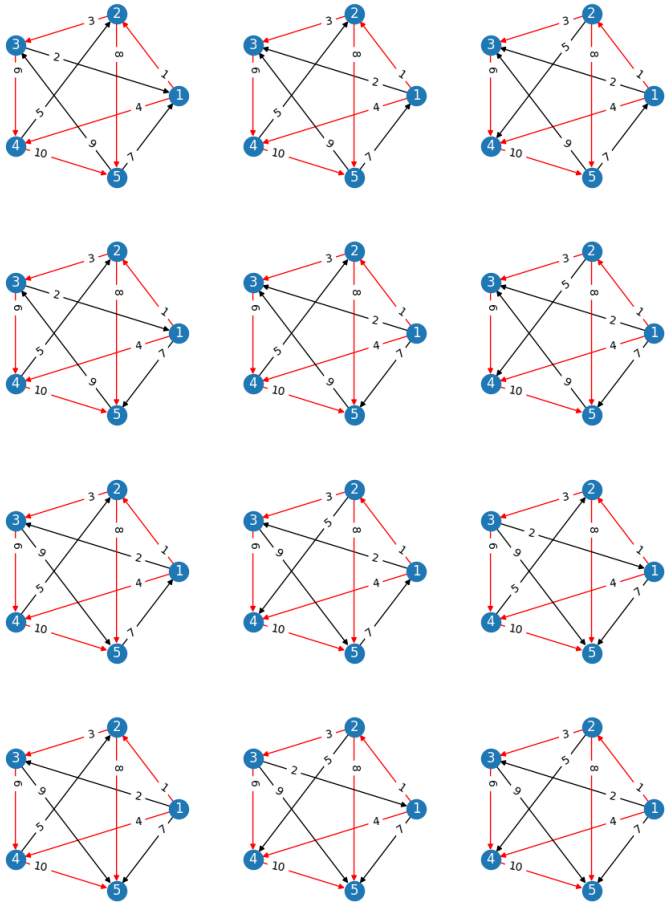

Fig. 7. Isomorphism class representatives admitted by a particular isolator for 5-vertex tournaments. Red edges are edges fixed by unit clauses of the isolator. The two non-unit clauses in the isolator are $2 \vee -5 \vee 9$ and $2 \vee 7 \vee -9$.

2) Unit/Binary Clauses: Another consequence of undirected isolators requiring at least one positive and one negative literal per clause is that undirected isolators have no 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 replaces the generation of units for TT_k with the following clauses:

$(a_0 \vee a_1)$

$(\neg a_0 \vee \neg a_1)$

$\{\neg a_0 \vee I_{bc} | (b, c) \in [1..k] \times [1..k] \text{ s.t. } b < c\}$

$\{\neg a_1 \vee \neg I_{bc} | (b, c) \in [1..k] \times [1..k] \text{ s.t. } b < c\}$

where a_0, a_1 are auxiliary variables representing the concepts "the k-clique is red/blue" respectively. We note that these clauses are "almost" units in the sense that after a solver makes a decision about which of a_0, a_1 to set as True, an entire $k(k-1)/2$ edges are set by unit propagation.

D. Ease of Isomorphism

Another interesting difference between undirected graphs and tournaments is the low number isomorphism classes for tournaments when n is small (see table I). Intuitively, this happens because it is "easy" for tournaments to be isomorphic. The two options for the edge between vertices A and B in the undirected case are AB existing or not existing. Crucially, an undirected graph G will never be isomorphic to G' constructed by adding or removing an edge of G (an operation that can be seen as "flipping" an edge to its other possibility). However, "flipping" an edge of a tournament T (changing the edge's direction) will produce $T' \simeq T$ iff the two vertices A and B of the flipped edge had the same edges to the rest of the graph (the isomorphism is via the permutation that swaps A and B).

REFERENCES

[1] M. J. Heule, "Optimal symmetry breaking for graph problems," *Mathematics in Computer Science*, vol. 13, no. 4, pp. 533–548, 2019.

[2] B. D. McKay and A. Piperno, "Practical graph isomorphism, ii," *Journal of symbolic computation*, vol. 60, pp. 94–112, 2014.

[3] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of reasoning*. Springer, 1983, pp. 466–483.

[4] C. Sinz, "Towards an optimal cnf encoding of boolean cardinality constraints," in *International conference on principles and practice of constraint programming*. Springer, 2005, pp. 827–831.

[5] D. Neiman, J. Mackey, and M. Heule, "Tighter bounds on directed ramsey number r (7)," *arXiv preprint arXiv:2011.00683*, 2020.