Finding Graph Decompositions via SAT

Wenting Zhao*, Mark H. Liffiton*, Peter G. Jeavons† and Dan Roberts*
*Illinois Wesleyan University, Bloomington, IL USA
†Department of Computer Science, University of Oxford, UK
*{wzhao,mliffito,drobert1}@iwu.edu, †peter.jeavons@cs.ox.ac.uk

***Abstract*—We begin a systematic study of how Graph Decomposition problems may be represented using propositional formulas, and hence solved using SAT-solver technology. By making use of symmetry breaking techniques we are able to obtain solutions to several previously unknown cases and to significantly reduce the time needed to compute decompositions. However some fairly small instances remain unsolved, and thus provide an interesting challenge to SAT-solver technology.**

I. Introduction

The study of combinatorial designs is a very active field in mathematics, with links to geometry, group theory, number theory and linear algebra [6]. It has applications in many areas including experimental design, coding theory, cryptography and bioinformatics. It is also a rich source of challenging open problems: the existence of combinatorial designs is often unknown, even for very small problem sizes [4], [6].

A promising technique for tackling combinatorial search problems of many kinds, which has risen to prominence over the past two decades, is the use of SAT-solvers. Such tools are now capable of solving some instances with millions of variables, and in recent years they have been successfully applied to solve several open problems in combinatorial designs: Heule et al. have completely solved the Boolean Pythagorean Triples problem [13]; Zhang produced 25 new r-self-orthogonal Latin squares [24]; Koshimura et al. obtained solutions for two open job-shop scheduling problems ABZ9 and YN1 [16].

However, standard combinatorial design problems can prove surprisingly difficult to solve effectively using SAT-solvers, even when the number of variables is quite small. The general techniques developed for efficient SAT-solving appear not to be enough to solve many common combinatorial design problems; additional techniques often seem to be required to reduce the search space and avoid redundant work.

In this paper, we explore the technique of using SAT-solvers to tackle an important family of combinatorial design problems known as *Graph Decomposition* problems. We develop a framework to encode Graph Decomposition problems into SAT, under which one can flexibly deal with many variants of the problem such as the packing problem, the resolvability problem and the multidecomposition problem (for a survey of the Graph Decomposition problem and its variants, see [6]). We also explore the symmetries of Graph Decomposition problems and develop some effective techniques for breaking such symmetries in the SAT formulation.

We first apply our techniques to the central problem of decomposing a complete graph K_n into smaller complete graphs. By applying symmetry-breaking techniques, the time needed to obtain a K_6-decomposition of K_{36} was reduced from being infeasible to 20 minutes. The smallest currently unknown case of this type is the problem of finding a K_6-decomposition of K_{51}. This problem still remains out of reach, although we have encoded it as a SAT-formula with only 4335 variables and shown that around 36% of these can be fixed by symmetry breaking. It therefore represents an intriguing challenge to the developers of SAT-solvers.

To show the flexibility of our techniques, we then apply our framework to encode the $(C_6, \overline{C_6})$-multipacking problem for K_n and the problem of finding almost-resolvable (2-star)-decompositions of K_n. For these problems we are able to produce answers to several previously-unknown cases.

In the remainder of the paper, we set out formal definitions for the Graph Decomposition problem and the SAT problem in Section II. We describe our framework for encoding Graph Decomposition problems and breaking symmetries in such problems in Section III, present three case studies to show the effectiveness of these techniques in Section IV, and provide some details of our implementations in Section V. We briefly describe related work and compare the Graph Decomposition problem with another recently-solved combinatorial problem in Section VI. Finally we conclude in Section VII.

II. Preliminaries

We first introduce the *Graph Decomposition Problem*. A *graph* is an ordered pair $G = (V, E)$ where V is a set of vertices, or nodes, and E is a set of edges, which are 2-element subsets of V. An *H-decomposition of G* is a partition of E into subgraphs isomorphic to H. The copies of H are called the *blocks* of the decomposition. Given a fixed graph H, the *H-decomposition problem* is to determine whether an input graph G admits an H-decomposition.

For any graph G, there are three necessary conditions for the existence of an H-decomposition of G: first, $|V(G)| \geq |V(H)|$; second, $|E(G)|$ is divisible by $|E(H)|$; finally, the degree of every vertex in G can be obtained as a sum of degrees of vertices in H. In some cases, these three conditions appear to be sufficient for such a decomposition to exist, but more often, additional conditions are required, and in many cases these have not yet been fully identified. Hence, the existence of H-decompositions for many graphs is currently unknown.

We also recall the definition of the *Boolean Satisfiability Problem (SAT)*. A propositional formula $\mathcal{F}$ is said to be in *conjunctive normal form (CNF)* if it is a conjunction of *clauses*, $\mathcal{F} = \bigwedge_{i=1..n} C_i$; each clause C_i is a disjunction of *literals*, $C_i = l_{i1} \lor l_{i2} \lor \cdots \lor l_{ik_i}$; and each literal is either a Boolean variable x or its negation $\neg x$. For example, consider the following formula: $\varphi = (a) \land (\neg b) \land (\neg a \lor \neg c) \land (b \lor \neg c)$ The formula φ consists of four clauses over three Boolean variables a, b, c, where the first two clauses are unit clauses and the other two clauses are binary clauses, that is, disjunctions of two literals.

Throughout the paper, we will treat all Boolean variables as $0/1$ variables, where the value 1 represents True and the value 0 represents False. A propositional formula $\mathcal{F}$ is said to be *satisfiable* if there exists an assignment to all variables that makes $\mathcal{F}$ evaluate to 1; otherwise, we say $\mathcal{F}$ is *unsatisfiable*. The above formula φ is satisfiable because it evaluates to 1 under the assignment $a = 1, b = 0, c = 0$.

The *Boolean satisfiability problem (SAT)* is defined as follows: given a propositional formula, determine whether it is satisfiable, and if so find a satisfying assignment. Tackling this problem has become much more efficient over the past two decades because of the development of software tools known as automated SAT-solvers. Hence it is now a viable approach for many search problems to translate each instance into a corresponding SAT instance and apply a standard SAT-solver.

To translate a Graph Decomposition problem into SAT, it is often useful to explicitly encode information about the cardinality of certain sets. While this can be done using standard formulas in CNF, it is often more efficient to use special purpose *cardinality constraints*, or counting constraints, which impose a bound on the number of literals in some set that can be assigned the value 1. Given a set of n literals $\{a_1, a_2, \ldots, a_n\}$ and an integer bound k, s.t. $0 \leq k \leq n$, a cardinality constraint is defined as:

$$\sum_{i=1}^{n} a_i \gtreqless k$$

where $\gtreqless$ is any relation from the set $\{\leq, =, \geq\}$, forming *AtMost*, *Equals*, and *AtLeast* constraints, respectively. We will refer to an instance including cardinality constraints in addition to clauses as a *SAT+cardinality instance*.

In this paper, we will develop general techniques to encode a Graph Decomposition problem instance for an input graph G into a SAT+cardinality instance. That is, solving this SAT+cardinality instance is equivalent to answering the question of whether or not there exists a decomposition of G of the required type. Moreover, if the SAT+cardinality instance is satisfiable, then the satisfying assignment, or the model, can be decoded into a decomposition of G of the required type. On the other hand, if the SAT+cardinality instance is shown to be unsatisfiable, that will constitute a proof that such a decomposition does not exist.

Note that both SAT and the Graph Decomposition problem are NP-complete [7], so it is known that in principle there will exist reductions from one to the other. Here, we propose a simple and efficient way to translate from various kinds of instances of the Graph Decomposition Problem to instances of SAT (plus additional cardinality constraints), and show that such translations can be used in practice with current SAT-solving tools to solve some previously unknown cases.

III. Framework

A. Encoding

When encoding a problem into a SAT instance, there are generally two major questions we need to answer – what are the variables, and what are the constraints (i.e., clauses), both of which play a key part in correctness and efficiency.

a) Variables: We propose two types of variables to encode Graph Decomposition problems: vertex variables and edge variables. A vertex variable indicates whether a particular *vertex* is included in a particular block, and an edge variable indicates whether a particular *edge* is included in a particular block.

Given the graphs H and G, the number of copies of H required for an H-decomposition of G (i.e., the number of blocks) is given by $c = |E(G)|/|E(H)|$. For each copy of H, we associate one copy of G with it. For a copy G_i where $0 \leq i < c$, we make a variable for each vertex of G_i, denoted by $v_{i,j}$ where $0 \leq j < |V(G)|$. The variable $v_{i,j}$ will be set to 1 iff the vertex j is used in the ith copy of H; otherwise, it is set to 0. Similarly, we can make a variable for each edge of G_i, denoted by $v_{i,\{x,y\}}$ for all $\{x, y\} \in E(G)$. The variable $v_{i,\{x,y\}}$ will be set to 1 iff the edge $\{x, y\}$ is used in the ith copy of H; otherwise, it is set to 0.

b) Constraints: We propose several kinds of constraints that are useful for Graph Decomposition problems and variants. The exact restrictions required will be determined by the exact form of Graph Decomposition problem we are encoding.

1) **#vertices:** A necessary condition for a graph decomposition is that, in each copy of G, exactly $|V(H)|$ vertices are used. That is, for every $0 \leq i < c$, $\Sigma_{j=0}^{|V(G)|-1} v_{i,j} = |V(H)|$.
2) **#edges:** Similarly, the number of edges that is used in each copy of G should be exactly $|E(H)|$. That is, for every $0 \leq i < c$, $\Sigma_{\{x,y\} \in E(G)} v_{i,\{x,y\}} = |E(H)|$.
3) **Edges being mutually exclusive and complete:** By the definition of a graph decomposition, each edge in G is used exactly once. Thus, for all $\{x, y\} \in E(G)$, $\Sigma_{i=0}^{c-1} v_{i,\{x,y\}} = 1$.
4) **Degree condition:** We can also look at the degree of each vertex in H. For example, if vertex x is used in the ith copy of G, then the number of its adjacent vertices present in that block should be not less than the minimal vertex degree in H and not greater than the maximal vertex degree in H. This type of constraint is particularly helpful when all vertices in H have the same degree.
5) **Linking vertex and edge variables:** if using both types of variables simultaneously, we associate them by generating the constraints $v_{i,\{x,y\}} \to v_{i,x} \land v_{i,y}$. That is, if the edge $\{x, y\}$ is present in the ith block, then the vertices x and y have to be present there as well. Notice

that the converse is also true in the important special case when H is a complete graph; however, in general when vertices x and y are both present in a block, the edge $\{x, y\}$ is not necessarily part of that block.

B. Symmetry Breaking

To prove that G does not admit an H-decomposition, it is necessary (in some way) to eliminate every possible combination of subgraphs. The search space is thus (potentially) exponentially large. Therefore, reducing the search space and avoiding redundant work is crucial to efficient performance.

A *solution symmetry* in a constraint satisfaction problem (including a SAT problem) is formally defined [5] as a permutation of the set of variable-value assignments that preserves the set of solutions. When such solution symmetries exist, they may be applied to any solution to potentially obtain other, equivalent solutions. Hence each solution symmetry partitions the set of solutions into equivalence classes, which are the orbits of the set of solutions under the action of that symmetry [5].

A common approach to solving problems with symmetries more efficiently is to add additional *symmetry-breaking* constraints to the problem which allow only one representative from each equivalence class of solutions under that symmetry. Adding such symmetry breaking constraints will not change the satisfiability of an instance; it may however dramatically reduce the number of models for a satisfiable instance, and it may dramatically reduce the search space for an unsatisfiable instance.

In the context of Graph Decomposition problems, such symmetries can arise in several different ways. For example, in our problem formulation each block is numbered; permuting the blocks will simply map any solution to an equivalent solution. We refer to this form of symmetry as *block symmetry*. To break such a symmetry, we can impose extra conditions on the solutions. For example, if we associate each block i with a $0/1$ vector of its vertex variables $\langle v_{i,0}, v_{i,1}, \dots, v_{i,|V(G)|-1}\rangle$, then we can add constraints that only permit solutions where the vectors for the blocks are ordered lexicographically.

Similarly, in our formulation each vertex in G is numbered; there may be permutations of the vertices that preserve the edge relation. Such a permutation is known as an automorphism of G. Any automorphism of G will preserve solutions to the graph decomposition problem, and so will give rise to a solution symmetry. We refer to this form of symmetry as *vertex symmetry*, and again such symmetries can be broken with extra constraints. For example, if we associate each vertex j with a $0/1$ vector of its variables with an entry for each block $\langle v_{0,j}, v_{1,j}, \dots, v_{c-1,j}\rangle$, then we can add constraints that only permit solutions where the vectors for vertices in the same orbit of an automorphism are ordered lexicographically.

As we will see below, breaking both block symmetries and vertex symmetries at the same time is similar to the problem of breaking both row and column symmetries in matrix models, which has been studied in other contexts [9].

IV. Case Studies

A. K_r-decomposition of K_n

We denote by K_i the *complete graph* on i vertices, which is a graph where every pair of distinct vertices is connected by an edge. Deciding whether there exists a K_r-decomposition of K_n, for given values of r and n, is a long-standing problem in mathematics. We remark that finding a K_r-decomposition of K_n is equivalent to finding a $2-(n, r, 1)$ balanced incomplete block design in the field of combinatorial designs [6].

When $r = 2$, this problem is trivial, as each individual edge can be taken as a block. The case of $r = 3$ was solved in 1847 by Kirkman [15] who showed that a K_3 decomposition of K_n exists iff $n \equiv 1$ or 3 $(mod\ 6)$. Such decompositions, when they exist, are known as *Steiner Triple Systems*.

The existence problem is now completely solved for all values of r up to 5, and it is partially solved for $6 \leq r \leq 9$. After that point, very little is known (for a survey, see [4]). For the cases known to exist, SAT-solvers can in principle be used to enumerate all solutions; for the unsolved cases, SAT-solvers can in principle be used to decide their existence.

We will focus on the case of $r = 6$, that is, the problem of K_6-decomposition of K_n, since this is the smallest case where the problem of existence is not entirely solved. It is solved asymptotically; that is, it is known that there exists a K_6-decomposition of K_n when $n \equiv 1$ or 6 $(mod\ 15)$ and n is sufficiently large [4].

Much work has been done to resolve the remaining cases, including [1], [2], [3]. However, there are several finite cases that remain unknown: the smallest is $n = 51$. The case of $n = 46$ was shown not to exist in 2001 by an exhaustive computational search using roughly 3 CPU years [14]. Since then, no progress has been made. Therefore, determining whether or not there exists a K_6-decomposition of K_{51} is of great interest to the graph theory community, and we take this problem as our first case study. We will also use the known cases of the K_6 decomposition problem (i.e., K_{16}, K_{21}, K_{31}, K_{36} and K_{46}) as benchmarks to test the efficiency of the techniques we have developed.

a) Encoding: For this problem, because H is a complete graph, we only need to use vertex variables. There are 1275 edges in K_{51} and 15 edges in K_6. Therefore, a K_6 decomposition of K_{51}, if it exists, would have 85 blocks, which yields $85 \times 51 = 4335$ vertex variables. The following constraints encode a valid decomposition:

1) Every vertex of K_{51} is used in exactly 10 blocks; that is, for every $0 \leq j \leq 50$, $\Sigma_{i=0}^{84} v_{i,j} = 10$.
2) In every block, there are exactly 6 vertices used; that is, for every $0 \leq i \leq 84$, $\Sigma_{j=0}^{50} v_{i,j} = 6$.
3) Every edge in K_{51} is used exactly once; the same edge cannot be used in two different blocks at the same time. Hence, for all $\{x, y\} \in E(K_{51})$, $\forall i_1 \neq i_2$, where $0 \leq i_1, i_2 \leq 84$, $\neg v_{x,i_1} \vee \neg v_{y,i_1} \vee \neg v_{x,i_2} \vee \neg v_{y,i_2}$.

The resulting SAT instance is satisfiable iff there exists a K_6-decomposition of K_{51}, and any satisfying assignment of the instance is associated with a unique K_6 decomposition of K_{51}.

b) Symmetry Breaking: In a complete graph, any permutation of the vertices is a graph automorphism. Thus, given an instance of K_{51} with each vertex indexed, we can rename the set of vertices using any permutation of the indices. This property of the graph induces a vast amount of solution symmetry in the Graph Decomposition problem. For example, to choose the first K_6 in the decomposition, $\binom{51}{6} = 18{,}009{,}460$ equivalent choices can be made, since any choice of K_6 can be obtained from any other by renaming the vertices of K_{51}.

As noted above, finding a K_6 decomposition of K_{51} is equivalent to finding a $2-(51, 6, 1)$ block design. Finding such a design can be modeled as finding a particular $0/1$-matrix [9]. Breaking symmetries in such a model by requiring the rows and the columns to be in lexicographical order is equivalent to breaking block symmetries and vertex symmetries in our framework. However, enforcing lexicographic ordering using constraints expressed by clauses is not trivial; enforcing such a constraint on two consecutive columns/rows of length n requires $(n^2 + n)/2$ clauses [10].

We thus break symmetries in a much more space-efficient way; we impose *partial* lexicographic order by adding unit clauses to the formula. Notice that in a modern SAT solver, unit clauses are often not stored as clauses; the solver immediately makes an assignment for the associated variable at the root level of the search tree, thus reducing the size of the formula. For example, we can break the symmetry between different choices for the first K_6 described above by simply assigning vertices $0-5$ to the first K_6 block; if this assignment does not lead to a solution, then, by symmetry, assigning any other six vertices to the first block will not lead to a solution either, so correctness is maintained.

Extending this idea, we obtain the symmetry breaking shown in Fig. 1. In this figure, the complete set of variables is arranged in a 51×85 matrix. Several distinct regions within this matrix are indicated by different colors. Each colored region indicates a particular form of symmetry breaking, described in more detail below. Each square filled with black indicates a variable assigned the value 1, and each square shaded with a color indicates a variable assigned the value 0. Together, the various forms of symmetry breaking we describe below reduce the number of unassigned variables from 4335 to 2759 and ensure that a large proportion of the rows and columns are lexicographically ordered without needing any additional clauses (except unit clauses).

We start by considering the purple region in columns 0-9 and across row 0. Because vertex 0 must be used ten times in a K_6-decomposition of K_{51}, and because the columns can be re-ordered arbitrarily, we assign vertex 0 to be used in blocks 0 through 9 by placing a 1 in the first 10 cells of row 0.

Next, we note that vertex 0 will have to occur in some block with every other vertex, so we can assign vertices 1 through 50 sequentially to be in one of the first 10 blocks along with vertex 0, with exactly 5 of them in each block. By re-ordering the vertices and the blocks, we can assign the value 1 to precisely those variables shown in black within the purple region. Given the cardinality constraints defined for vertices (rows) and blocks (columns), the other variables in the purple region must all be assigned the value 0.

Next we look at the green region in rows 1-5. In each row within this region, each vertex must be used 9 more times. Hence, by re-ordering the blocks we can place these 9 1s in each row in the positions shown in Fig. 1. Again, because of the cardinality constraints, the rest of the variables in those rows must then be assigned the value 0.

Next we look at the light blue region in rows 6-10, columns 10-54. In any decomposition of K_{51}, the edge $\{1, 6\}$ must appear in some block. We have already made assignments that assign vertex 1 to blocks 0 and 10 through 18. Putting vertex 6 in block 0 is not possible, as this block already contains 6 other vertices. By re-ordering equivalent blocks we can place vertex 6 in block 10. By a similar argument, we can assign vertex 7 to block 11, and so on for vertices 8 through 10. This allows us to assign the diagonal sequence of 1s in rows 6-10 and columns 10-14. Similar arguments allow us to assign all the other 1s shown in the light blue region, and then all the remaining variables in this region can be assigned 0.

Now the yellow region in the remaining columns of rows 6-10 is similar to the green region. Vertices 6 through 10 must each be used 4 more times, hence by re-ordering the blocks we can place the 4 remaining 1s in each of these rows in the positions shown by black squares in Fig. 1. Again, because of the cardinality constraints, the rest of the variables in those rows must then be assigned the value 0.

Given the assignments thus far, the last 40 rows may be divided into 8 equivalence classes (of 5 consecutive rows each) that are indistinguishable from each other. Within each equivalence class, every row (vertex) is also indistinguishable from the others. Hence by reordering these rows we may assume that the remaining 4 1s in block 10 are in the positions indicated in Fig. 1. The dark blue cells in the rest of column 10 arise from the cardinality constraint on that column, while the dark blue cells in rows 11, 16, 21, and 26 are due to the mutual exclusion clauses preventing duplicate use of any edge.

For the orange regions in columns 11-15, we first consider the 4×4 submatrix whose top-left corner is vertex 12 and block 11. Because nothing distinguishes vertices 12 through 15 thus far, we can permute these four rows arbitrarily. Since each row and each column of this submatrix can contain at most one 1 (because each edge of G occurs in at most one block), we can re-order these rows to ensure that any variables taking the value 1 are not in the shaded positions. Hence these shaded positions can all be assigned the value 0. Similar remarks apply to all other square submatrices within the orange region.

A similar argument applies to the dark orange-red regions in rows 11-15, here noting that the *columns* in each 4×4 submatrix are thus far indistinguishable. We can re-order the 4 columns in each submatrix to ensure that any variables taking the value 1 are not in the shaded positions. Hence these shaded positions can all be assigned the value 0.

We have described this symmetry-breaking for K_{51}, but the reasoning can be applied to produce similar patterns of assignments for K_6-decompositions of other sizes as well.

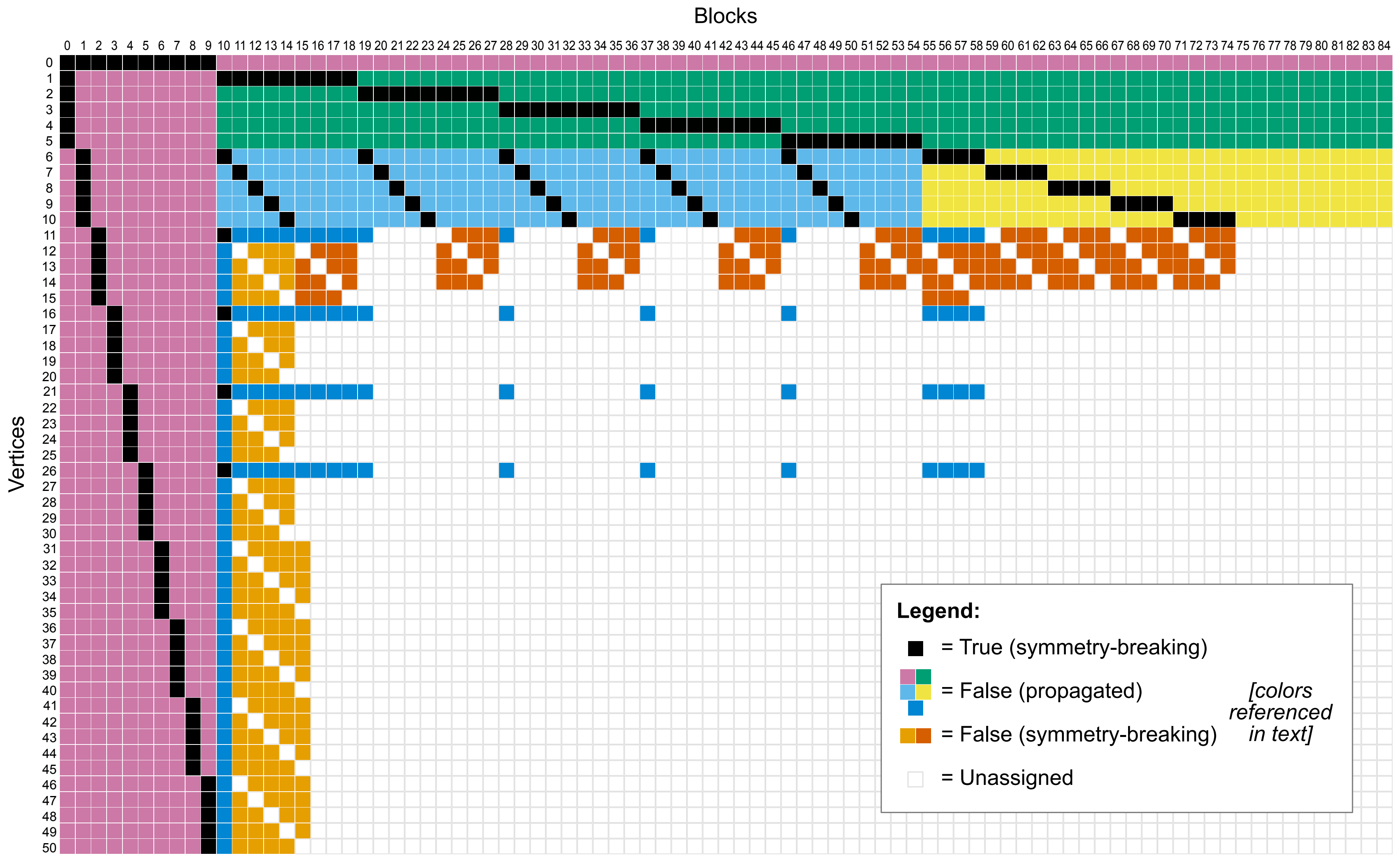


Fig. 1. Complete symmetry breaking for K_{51}.

c) Results: By using our encoding, the instances of K_6-decomposition of K_{16} and of K_{21} were proven UNSAT in 0.01 seconds with no symmetry breaking predicates needed. Without symmetry breaking, it took roughly 100 seconds to produce a K_6-decomposition of K_{31}; with the unit clauses added, a solution was computed within less than a second. Similarly, without symmetry breaking, the SAT-solver was not able to produce an answer for K_{36} with a timeout of several days; with symmetry breaking predicates, an UNSAT answer was determined in 20 minutes. Unfortunately, the instances of K_6-decomposition of K_{46} and K_{51} both ran out of memory (>16GB) after roughly 14 days. However, we can still see that symmetry breaking plays a key role in determining the existence of a graph decomposition as n increases.

B. $(C_6, \overline{C_6})$-multipacking of K_n

One variant of the Graph Decomposition Problem that is also of interest to graph theorists is, rather than decomposing a graph into isomorphic copies of *one* graph, to ask whether we can decompose the graph into isomorphic copies of *several* graphs. An (H_1, H_2)*-multidecomposition of* G is a partition of the edge set of G into isomorphic copies of H_1 and H_2 such that at least one copy of each graph is present.

One necessary condition for the existence of an (H_1, H_2)-multidecomposition of G is that there must exist a positive integer solution to $h_1x + h_2y = |E(G)|$, where h_1 is the number of edges of H_1 and h_2 is the number of edges in H_2. When an integer solution does not exist, we can ask how close we can get to a multidecomposition – the problem then becomes to find a *multipacking* of the graph. An (H_1, H_2)-*multipacking of* G is a partition of a *subset* of the edge set of G into isomorphic copies of H_1 and H_2 such that at least one copy of each graph is present. The set of edges of G not used in copies of H_1 or H_2 is called the *leave* of the (H_1, H_2)-multipacking of G, denoted L. The goal is to find a *maximum* multipacking solution, that is, one where the leave contains the fewest possible edges.

For our second case study we will focus on the problem of finding a maximal $(C_6, \overline{C_6})$-multipacking of K_n, where C_6 is the cycle graph with six vertices, and $\overline{C_6}$ is the complement of C_6, that is, the graph on the same vertices such that two distinct vertices of $\overline{C_6}$ are adjacent iff they are not adjacent in C_6. Gao and Roberts solved this problem for general cases by a recursive construction, building on several base cases, including K_{11} and K_{17} [11]. No generic constructions are known for these base cases, but they can be solved by translating to SAT, as we will now show.

a) Encoding: Our encoding for this multipacking problem follows the general framework described above. For the multipacking problem, we use both vertex variables and edge variables. Because the constraints on each block are built individually, we can encode some blocks to contain C_6 while other blocks contain $\overline{C_6}$. Let c be the total number of blocks in the multipacking, c_1 be the number containing C_6, and c_2 be the number containing $\overline{C_6}$. There are three types of constraints applied to all blocks:

1) Every edge of K_n not in the leave L is used exactly once: $\forall \{x,y\} \in E(K_n) \setminus L$, $\Sigma_{i=0}^{c-1} v_{i,\{x,y\}} = 1$.
2) In every block, exactly 6 vertices are used: for every $0 \le i < c$, $\Sigma_{j=0}^{n-1} v_{i,j} = 6$.
3) For every edge used in a block, its corresponding vertices are also present: $v_{i,\{x,y\}} \to v_{i,x} \wedge v_{i,y}$.

Blocks containing C_6 also have the following constraints:

1) In every such block, there are exactly 6 edges: for every $0 \le i < c_1$, $\Sigma_{\{x,y\} \in K_n \setminus L} v_{i,\{x,y\}} = 6$.
2) Every vertex used in such a block has degree two. For example, if vertex j is used in the ith block, then exactly two edges adjacent to vertex j are present in the ith block.
3) Every such block is triangle free: for every triple of vertices x, y, z in block i, if all the 3 vertices are present in the block, then there has to be a pair of vertices that do not share an edge; that is, $\neg v_{i,x} \vee \neg v_{i,y} \vee \neg v_{i,z} \vee \neg v_{i,\{x,y\}} \vee \neg v_{i,\{x,z\}} \vee \neg v_{i,\{y,z\}}$. This is necessary because there are two types of graphs satisfying all the previous constraints; one is C_6, and the other is two disconnected triangles on six vertices.

And blocks containing $\overline{C_6}$ have the following constraints:

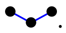
1) In every such block, there are exactly 9 edges: for every $c_1 \le i < c$, $\Sigma_{\{x,y\} \in K_n \setminus L} v_{i,\{x,y\}} = 9$.
2) Every vertex used in such a block has degree three.
3) Given a triple of vertices, if all of them are present in a block and the edges between two pairs of vertices are missing, then the edge between the third pair of vertices has to be present. That is, for every triple of vertices x, y, z in block i, $\neg v_{i,x} \vee \neg v_{i,y} \vee \neg v_{i,z} \vee v_{i,\{x,y\}} \vee v_{i,\{x,z\}} \vee v_{i,\{y,z\}}$.

b) Results: Using the above encoding, our SAT-solver identified a $(C_6, \overline{C_6})$-multipacking for K_{11} with $3C_6$ and $4\overline{C_6}$ within a second and a $(C_6, \overline{C_6})$-multipacking for K_{17} with $18C_6$ and $3\overline{C_6}$ within a minute. Notice that we did not break symmetries here, because adding symmetry-breaking constraints will reduce the number of solutions available, and our goal is only to find one solution to prove existence.

C. Almost resolvable (2-star)-decompositions of K_n

The Graph Decomposition Problem can be broadened to ask for the existence of a *resolvable* graph decomposition. A "parallel class" in an H-decomposition of G is a subset of blocks which span the vertices of G, i.e. each vertex of G is contained in the subset exactly once. A decomposition is resolvable if the blocks can be partitioned into parallel classes.

When a resolvable decomposition is impossible due to a mismatch in graph sizes (i.e., $|V(G)| \equiv 0 \pmod{|V(H)|}$), then we can ask for an *almost-resolvable* decomposition instead. An almost-resolvable H-decomposition of a graph G is a decomposition with classes that span all but j vertices of G, where $j < |V(H)|$. In other words, each class except one spans as many vertices as possible with copies of H, but there may be one smaller *partial* class made up of "leftover" edges and vertices.

Resolvable star decompositions have been completely characterized [22], and almost resolvable (2-star)-decompositions in which exactly one vertex is left out of each parallel class were characterized in Yu [21], where a 2-star, or 2-path, is the graph with three vertices and two edges . However, it remains an open problem to characterize almost resolvable (2-star)-decompositions in which 2 vertices are left out of each parallel class.

To complete a recursive construction proving the existence of almost-resolvable (2-star)-decompositions for all K_n, we needed to find almost-resolvable (2-star)-decompositions for four sizes of K_n (n = 17, 20, 29, and 32) with some additional requirements on the form of the decompositions [20]. In each instance needed by the proof, the decomposition not only has to be an almost-resolvable (2-star)-decomposition, but the vertices have to be partitioned into "regular" and "infinity" vertices and each class of the decomposition must contain a specific number of *pure* edges (between two "regular" vertices, or between two "infinity" vertices) and *mixed* edges (between one "regular" and one "infinity" vertex).

a) Encoding: As H in these decompositions is not a complete graph, it is not sufficient to use only vertex variables, as assigning vertices to the blocks is insufficient to specify the edges of the block. Our encoding uses a variation on edge variables: instead of creating a variable indicating whether a given edge was included in a given *block*, each variable instead represented whether or not an edge was included in a given *class*. This means that edges are not assigned to particular blocks but rather to a particular class of blocks. With additional constraints, we ensured that the edges included in a class do in fact form 2-stars within blocks. The constraints were as follows:

1) **Mutual exclusion + completeness**: Every edge must be included in exactly one class [item number 3 in the list of constraints in Section III-A].
2) **Vertex degrees**: Every vertex can have at most 2 incident edges included in a given class [item 4 in III-A].
3) **Edge counts**: For any class other than the unique partial class, the number of included edges must be **floor**$(|V(G)|/|V(H)|) \times |E(H)|$, and the partial class must include as many edges as are "leftover" [similar to item 2 in III-A].
4) **2-Star decomposition**: For any edge e included in a class, there must be exactly one other edge in that class sharing a vertex with e. (If every edge is incident on exactly one other included edge, then the class must be formed of 2-stars, as required.)
5) **Pure and mixed classes**: For the recursive construction, certain classes in a decomposition can contain no mixed edges, and certain classes can contain no pure edges between two "infinity" vertices. This is accomplished by forcing any such edge variables within the given classes to be 0.

These constraints are satisfiable iff there exists an almost-resolvable (2-star)-decomposition of a given K_n that satisfies the requirements of the recursive construction.

b) Symmetry Breaking: While not all vertices are equivalent in these instances due to the partition between the regular vertices and the infinity vertices, they are still equivalent within each part. Therefore, we broke the vertex symmetries within each part by forcing one class to include 2-stars "in vertex order" within each set of vertices. Specifically, for the k regular vertices $V_1, V_2, \ldots, V_k$, edges were assigned to 2-stars as (V_1-V_2-V_3), (V_4-V_5-V_6), etc., and the same ordering of 2-stars was enforced within the infinity vertices.

c) Results: We encoded the four instances for which solutions were needed (K_{17}, K_{20}, K_{29}, and K_{32}) with the above encoding. The formulas ranged in size from 1,904 to 12,400 variables and from 843,100 to 22,111,674 constraints. We were able to solve two of the four by finding decompositions for each: the K_{17} formula was solved in roughly one minute, while K_{20} took several hours. Removing the symmetry breaking constraints in this case was significantly detrimental to performance (the K_{17} instance took roughly 25 times longer without them). K_{29} and K_{32}, both substantially larger formulas, remain unsolved.

V. Tools

We solved the cardinality instances arising from these encodings with the MiniCard constraint solver [17], an extension of MiniSat [8] that handles cardinality constraints natively (i.e., as opposed to encoding them into CNF). We had to make one extension to MiniCard to handle implied cardinality constraints needed for encoding the "degree condition" (item 4 in section III-A), which was used in the multipacking case study (section IV-B).

While implications in CNF can be handled easily (the implication $y \rightarrow C$, where C is a disjunction, is equivalent to $\neg y \vee C$), an implied cardinality constraint like $y \rightarrow \text{AtMost}(lits, bound)$ has no such trivial construction. By extending the definition of a cardinality constraint to allow *duplicate literals* with a certain semantics, however, an implication can be encoded within a single constraint. Specifically, we modified MiniCard to allow duplicate literals within a cardinality constraint such that each contributes separately to reaching or exceeding the constraint's bound. For example, $\text{AtMost}([x_1, x_1, x_1, x_2, x_3], 4)$ (note that the literals are now a multiset, not a set) is then equivalent to $3x_1 + x_2 + x_3 \leq 4$.

With this modification, an implication of an AtMost constraint with k literals and bound b:

$$y \rightarrow \text{AtMost}(\{l_1, l_2, \ldots, l_k\}, b)$$

can be encoded as a single AtMost with a bound of k and with $k - b$ copies of y added to the implied AtMost's literals:

$$\text{AtMost}([y] \times (k-b) \uplus [l_1, l_2, \ldots, l_k], k)$$

This new constraint is satisfied if y is False, as the number of remaining literals is equal to the constraint bound. If y is True, however, the effective bound is reduced by the number of y literals, and hence the bound on the remaining k literals is $k - (k - b) = b$, making the induced constraint equivalent to the original AtMost constraint in that case.

VI. Related Work

To the best of our knowledge, solving Graph Decomposition problems using SAT has not been directly studied; however, there has been previous work on the closely related problem of finding *balanced incomplete block designs (BIBD)* [18]. BIBD problems have been widely used as benchmarks to evaluate the efficiency of newly-developed symmetry-breaking techniques [9], [12], [19]. However, none of this work was able to produce new results for unsolved cases.

Many papers have sought to tackle other hard combinatorial problems using SAT (for a survey, see [23]), and one recent success is that the *Boolean Pythagorean Triples (BPT) problem* was completely solved by SAT-solvers [13], closing a problem that had been open for over two decades. Here, we briefly discuss some distinctions between BPT and Graph Decomposition (GD) problems.

A triple $(a, b, c) \in \mathbb{N}^3$ is called Pythagorean if $a^2 + b^2 = c^2$. The BPT problem asks, given a set of integers $\{1...n\}$, whether it is possible to partition the set into two subsets such that no Pythagorean triples are present in the same subset.

We note that Graph Decomposition (GD) problems can be more complex than the BPT problem in the following ways: 1) BPT only needs to partition the set into two parts, while GD problems often seek a partition into many parts, and therefore must explore more possibilities for the partitioning; 2) the Pythagorean triples are all known beforehand and the only constraint is to ensure no such triple is present in the same part, while a GD problem often comes with a variety of requirements in addition to mutually exclusive edges, such as resolvability; 3) most importantly, BPT only has very few symmetries – an integer can either be in one or the other part – but the symmetries in a GD problem can grow exponentially with the number of vertices.

VII. Conclusions & Future Work

We have developed a framework to encode the Graph Decomposition problem and investigated efficient forms of symmetry breaking for such problems. Two kinds of associated Boolean variables were presented: vertex variables and edge variables. Using vertex variables alone can result in small instances but is often not sufficient to make a sound encoding for more complex forms of Graph Decomposition problems.

While there is no general technique to encode any arbitrary Graph Decomposition problem, we presented several general forms of constraints that can be combined to flexibly deal with many standard variants of such problems. By applying this framework, we showed a substantial improvement in the solution time to find a K_6-decomposition of K_{36} and successfully derived new $(C_6, \overline{C_6})$-multipackings and almost resolvable (2-star)-decompositions for certain complete graphs.

Many avenues are open for further research. For example, we have shown that in the SAT formulation of the problem of finding a K_r-decomposition of K_n many symmetries can be broken using unit clauses, and there is room for further extension and automation of this approach. Although the question of the existence of a K_6-decomposition of K_{51} is still out of reach, it may be possible to establish non-existence by considering an abstraction of the SAT formulation described here or by adapting the *Cube-and-Conquer (C&C)* approach used in [13] to prove unsatisfiability for this instance.

References

[1] R. Abel. Some new BIBDs with $\lambda = 1$ and $6 \leq k \leq 10$. *Journal of Combinatorial Designs*, 4(1):27–50, 1996.

[2] R. Abel and W. Mills. Some new BIBDs with $k = 6$ and $\lambda = 1$. *Journal of Combinatorial Designs*, 3(5):381–391, 1995.

[3] R. J. R. Abel, I. Bluskov, M. Greig, and J. de Heer. Pair covering and other designs with block size 6. *Journal of Combinatorial Designs*, 15(6):511–533, 2007.

[4] P. Adams, D. Bryant, and M. Buchanan. A survey on the existence of G-designs. *Journal of Combinatorial Designs*, 16(5):373–410, 2008.

[5] D. A. Cohen, P. Jeavons, C. Jefferson, K. E. Petrie, and B. M. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3):115–137, 2006.

[6] C. J. Colbourn and J. H. Dinitz. *Handbook of combinatorial designs*. CRC press, 2006.

[7] D. Dor and M. Tarsi. Graph decomposition is NP-complete: A complete proof of holyer's conjecture. *SIAM Journal on Computing*, 26(4):1166–1187, 1997.

[8] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT-2003)*, volume 2919 of *LNCS*, pages 502–518, 2003.

[9] P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *International Conference on Principles and Practice of Constraint Programming*, pages 462–477. Springer, 2002.

[10] A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Propagation algorithms for lexicographic ordering constraints. *Artificial Intelligence*, 170(10):803–834, 2006.

[11] Y. Gao and D. Roberts. Multidesigns for a graph pair of order 6. *arXiv preprint arXiv:1705.09638*, 2017.

[12] A. Grayland, I. Miguel, and C. M. Roney-Dougal. Snake lex: An alternative to double lex. In *International Conference on Principles and Practice of Constraint Programming*, pages 391–399. Springer, 2009.

[13] M. J. Heule, O. Kullmann, and V. W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 228–245. Springer, 2016.

[14] S. Houghten, L. Thiel, J. Janssen, and C. Lam. There is no (46, 6, 1) block design. *Journal of Combinatorial Designs*, 9(1):60–71, 2001.

[15] T. P. Kirkman. On a problem in combinations. *Cambridge and Dublin Math. J*, 2(191-204):1847, 1847.

[16] M. Koshimura, H. Nabeshima, H. Fujita, and R. Hasegawa. Solving open job-shop scheduling problems by SAT encoding. *IEICE TRANSACTIONS on Information and Systems*, 93(8):2316–2318, 2010.

[17] M. H. Liffiton and J. C. Maglalang. A cardinality solver: More expressive constraints for free (poster presentation). In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT-2012)*, pages 485–486, 2012.

[18] S. Prestwich. Balanced incomplete block design as satisfiability. In *Proceedings of the 12th Irish Conference on Artificial Intelligence and Cognitive Science*, 2001.

[19] J.-F. Puget. Symmetry breaking using stabilizers. In *International Conference on Principles and Practice of Constraint Programming*, pages 585–599. Springer, 2003.

[20] D. Roberts. personal communication, 2012.

[21] M.-L. Yu. Almost resolvable P_k-decompositions of complete graphs. *Journal of Graph Theory*, 15(5):523–541, 1991.

[22] M.-L. Yu. On tree factorizations of K_n. *Journal of graph theory*, 17(6):713–725, 1993.

[23] H. Zhang. Combinatorial designs by SAT solvers. *Handbook of Satisfiability*, 185:533–568, 2009.

[24] H. Zhang. 25 new r-self-orthogonal Latin squares. *Discrete Mathematics*, 313(17):1746–1753, 2013.