Solving the Identifying Code Set Problem with Grouped Independent Support[*] (Extended Version)[†]

Anna L.D. Latour[1] , **Arunabha Sen**[2‡] and **Kuldeep S. Meel**[1]
[1]National University of Singapore, Singapore
[2]Arizona State University, Tempe, AZ, USA

Abstract

An important problem in network science is finding an optimal placement of sensors in nodes in order to uniquely detect failures in the network. This problem can be modelled as an *identifying code set (ICS)* problem, introduced by Karpovsky *et al.* in 1998. The ICS problem aims to find a cover of a set S, such that the elements in the cover define a *unique signature* for each of the elements of S, and to minimise the cover's cardinality. In this work, we study a *generalised identifying code set (GICS)* problem, where a unique signature must be found for each *subset* of S that has a cardinality of at most k (instead of just each element of S). The concept of an *independent support* of a Boolean formula was introduced by Chakraborty *et al.* in 2014 to speed up propositional model counting, by identifying a subset of variables whose truth assignments uniquely define those of the other variables.

In this work, we introduce an extended version of independent support, *grouped independent support (GIS)*, and show how to reduce the GICS problem to the GIS problem. We then propose a new solving method for finding a GICS, based on finding a GIS. We show that the prior state-of-the-art approaches yield integer-linear programming (ILP) models whose sizes grow exponentially with the problem size and k, while our GIS encoding only grows polynomially with the problem size and k. While the ILP approach can solve the GICS problem on networks of at most 494 nodes, the GIS-based method can handle networks of up to 21 363 nodes; a $\sim$ 40$\times$ improvement. The GIS-based method shows up to a 520$\times$ improvement on the ILP-based method in terms of median solving time. For the majority of the instances that can be encoded and solved by both methods, the cardinality of the solution returned by the GIS-based method is less than 10% larger than the cardinality of the solution found by the ILP method.

1 Introduction

Imagine that you are in charge of ensuring the fire-safety of a hotel. Your smoke detectors can sense a fire in the room in which they are placed immediately, and sense a fire in an adjacent room with a time delay. You realise that this means that you can detect every fire, even if you do not place a detector in every room. When you tell the hotel manager, they ask you to minimise the number of smoke detectors that you place. Additionally, they tell you to make sure that, even if as many as five fires break out in different rooms at the same time, you can uniquely identify these multiple rooms based on the set of smoke detectors that detect smoke. How many detectors do you need, and where do you place them?

The above situation is an example of a *sensor placement problem*. This well-studied problem has applications ranging from satellite deployment [Sen *et al.*, 2019], to power grid monitoring [Padhee *et al.*, 2020], to identifying criminals [Basu and Sen, 2021b] or spreaders of misinformation [Basu and Sen, 2021a], and is typically formulated on graphs. In the example above, nodes represent the hotel rooms, with edges between adjacent rooms.

Graphs are fundamental tools for modelling the interaction between objects. For many real-world computational problems, a node in a graph represents a resource object and an edge between two nodes models the ability for the corresponding objects to communicate. Resource objects are often abstractions of critical objects such as satellites, informants in crime networks, or servers. The critical nature of these objects necessitates reliable failure detection. For this, we often rely on sensors, placed strategically on certain nodes.

In this paper, we study a generalised version of the *identifying code set (ICS)* [Karpovsky *et al.*, 1998] problem. In our version, a sensor placed in a node detects a failure that occurs in that node immediately, and detects failures in neighbouring nodes with a small time delay. A *generalised identifying code set (GICS)* is a set of nodes in which we must place a sensor such that any set of at most k simultaneous failures can be *uniquely* identified by the placed sensors. Conceptually, a GICS is a dominating set (*i.e.*, a set of nodes such

[*]Open-source tool, reproducibility info, and extended version of this paper are available at https://github.com/meelgroup/gismo.

[†]This is an extended version of a work with the same title, published at IJCAI 2023 as paper #4051: https://www.ijcai.org/proceedings/2023/0219.pdf.

[‡]Work done while Arunabha Sen visited NUS.

that each node is either in that set or is a neighbour of a node in that set) in an undirected graph, such that each subset of nodes with cardinality at most k can be uniquely identified by the sensors placed on the nodes this dominating set.

Existing methods for finding and minimising GICSes with one failure at a time, employ an *integer-linear programming (ILP)* encoding [Padhee *et al.*, 2020; Basu and Sen, 2021a; Basu and Sen, 2021b]. A straightforward generalisation of this ILP formulation to support multiple simultaneous failures scales poorly with network size and the number of simultaneous failures. This explosion of the model size limits the applicability of ILP-based methods to small networks and support for only one node failure at a time.

The primary contribution of this work is a novel computational technique for solving GICS problems, with a much more compact encoding. Specifically, we propose the concept of *grouped independent support (GIS)* (an extension of *independent support* [Chakraborty *et al.*, 2014b; Ivrii *et al.*, 2016; Soos and Meel, 2022; Yang *et al.*, 2022]), and show how we can reduce the problem of finding a GICS to the problem of finding a GIS. We then propose a new algorithm, called GISMO, to compute a GIS.

The main benefit of this approach is that the more compact encoding enables us to solve GICS problems on much larger networks than the networks that can be solved by the state of the art. Indeed, our empirical analysis demonstrates that GISMO is able to handle networks of up to 21 363 nodes, while the ILP-based approach could not handle networks beyond 494 nodes, thus representing a $\sim 40\times$ improvement in terms of the size of the networks. Furthermore, depending on the number of simultaneous failures, the instances that can be encoded by both methods are solved up to $520\times$ faster by the GIS-based approach than by the ILP-based method. For the majority of those instances, the cardinality of the result returned by GISMO was at most 10% larger than the cardinality returned by the ILP-based method.

A conceptual contribution is to expand the usefulness of the notion of independent support. The computation of independent supports has, to the best of our knowledge, so far only been used as a preprocessing step for model counting and uniform sampling [Chakraborty *et al.*, 2014b; Ivrii *et al.*, 2016; Lagniez *et al.*, 2016; Lagniez *et al.*, 2020; Yang *et al.*, 2022; Soos and Meel, 2022]. We are the first to use the independent support for modelling and solving an NP-hard problem directly.

The remainder of this paper is organised as follows. We briefly discuss notation and provide relevant definitions in Section 2, where we also provide a motivating example of a GICS problem. Then, we describe the current state of the art for solving GICS problems in Section 3. Section 4 describes GIS, the reduction from the GICS problem to the GIS problem, and GISMO. We present an experimental evaluation of our implementation of GISMO on a variety of networks in Section 5, and conclude in Section 6.

2 Preliminaries

We briefly introduce our notation, recall relevant concepts, and define the *generalised identifying code set (GICS)*.

2.1 Definitions and Notation

Graphs. We consider an undirected, loop-free graph $\Gamma = (V, E)$ on nodes V and edges E. We denote nodes with lower case letters $u, v, w \in V$. The distance between two nodes u and v is the number of edges on the shortest path between them, and is denoted by $d(u, v)$. If $d(u, v) = 1$, we call the nodes u and v *direct neighbours* of each other. The neighbourhood function $N_d(v)$ returns the set of nodes that are at a distance d from node v. We define the *closed d-neighbourhood* of a node v as $N_d^+(v) = N_d(v) \cup \{v\}$. For a set of nodes U, we define the neighbourhood function $N_d(U) = \bigcup_{u \in U} N_d(u)$, and define the closed neighbourhood of U, $N_d^+(U)$, analogously.

Boolean Satisfiability. We denote a set of Boolean variables with the capital letter X and denote individual Boolean variables with lowercase letters $x, y, z \in X$. We denote truth values with 1 (*true*) and 0 (*false*). A literal l is a variable (*e.g.*, x) or its negation (*e.g.*, $\neg x$). A disjunction of literals is called a *clause*. We say that a formula F is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. A *full assignment* $\sigma : X \mapsto \{0, 1\}$ assigns a truth value to each variable in X. We use $\sigma(x)$ to denote the truth value that σ assigns to variable x. Given a subset $Y \subseteq X$, $\sigma_{\downarrow Y} : Y \mapsto \{0, 1\}$ denotes the assignment *projected* onto Y, thus specifying the truth values that the variables in Y get under σ. Given a Boolean formula $F(X)$, we call an assignment σ a *solution* or *model* of $F(X)$ if $F(\{x \mapsto \sigma(x) \mid x \in X\}) \models 1$. We denote the set of all models of F with $Sol(F)$. Similarly, we denote the set of all models of $F(X)$ projected on the subset $Y \subseteq X$ as $Sol_{\downarrow Y}(F)$. We call the variables X that appear in F the *support* of F. If a Boolean formula has at least one solution, we say that it is *satisfiable*. Otherwise, we call it *unsatisfiable*.

Minimality. Let T be a set of items, and let $S \subseteq T$ be a subset. Given a set $\mathcal{C}$ of constraints on sets, we call S *set-minimal w.r.t.* $\mathcal{C}$ if S satisfies all constraints in $\mathcal{C}$ and there exists no proper subset of S that also satisfies all those constraints. We call S a *cardinality-minimal* set if S is minimal, and there exists no $S' \subseteq T$ that is also minimal, but whose cardinality is strictly smaller than that of S.

Support of a Set. We use calligraphic uppercase symbols to denote sets of sets of variables. We define the *support* of a set of sets of variables $\mathcal{S}$ as follows: $sup(\mathcal{S}) := \bigcup_{S_i \in \mathcal{S}} S_i$.

Signatures. Given an undirected, loop-free graph $\Gamma := (V, E)$ with nodes V and edges E, and given a subset of nodes $D \subseteq V$. We define the *signature* of $U \subseteq V$ as the following tuple: $s_U := \langle S_U^0, S_U^1 \rangle$, where $S_U^0 := U \cap D$ and $S_U^1 := N_1^+(U) \cap D$.

Generalised Identifying Code Set (GICS). Given a graph $\Gamma := (V, E)$, a positive integer $k \leq |V|$ and $D \subseteq V$. We call D a *generalised identifying code set (GICS)* of Γ and k if, for all $U, W \subseteq V$ with $|U| \leq k$, $|W| \leq k$ and $U \neq W$, it holds that $s_U \neq s_W$. Hence, if D is a GICS of Γ and k, then the signatures of all subsets of V with cardinality at most k are unique. We call k the *maximum identifiable set size*.

The GICS Problem. Given a $\Gamma := (V, E)$ and k, the GICS problem asks to find a $D \subseteq V$ such that D is a GICS of Γ and k, and $|D|$ is minimised.

Independent Support. Given a Boolean formula $F(X)$ and a set $I \subseteq X$, we call I an *independent support* [Chakraborty *et al.*, 2014b] of F iff, for two solutions σ_1 and σ_2, the following holds: $(\sigma_{1\downarrow I} = \sigma_{2\downarrow I}) \Rightarrow (\sigma_{1\downarrow X} = \sigma_{2\downarrow X})$.

The concept of independent support was introduced in 2014 [Chakraborty *et al.*, 2014a], born from the observation that the truth values assigned to variables in solutions to a formula, can often be defined by the truth values of other variables. Hence, this property is referred to in the literature as *definability* [Lagniez *et al.*, 2016; Soos and Meel, 2022]. Tools for computing minimal independent supports include Arjun [Soos and Meel, 2022] and B+E [Lagniez *et al.*, 2016; Lagniez *et al.*, 2020].

Until now, independent supports have only been computed as a preprocessing step for counting and sampling [Chakraborty *et al.*, 2014a; Lagniez *et al.*, 2016; Lagniez *et al.*, 2020; Chakraborty *et al.*, 2014b; Ivrii *et al.*, 2016; Soos and Meel, 2022; Yang *et al.*, 2022]. In Section 4, we present a generalisation of the independent support of a Boolean formula, and show how we can use that to find solutions to the GICS problem. To the best of our knowledge, we are the first to lift computing independent supports out of the preprocessing domain, turning it into a tool for modelling and solving NP-hard problems directly.

Padoa's Theorem. Let $F(Z, A)$ be a Boolean formula on Boolean variables $Z \cup A$, with $Z \cap A = \varnothing$. We can use Padoa's theorem [Padoa, 1901] to check if a variable $z \in Z$ is defined by the other variables in Z. Let $\hat{Z}$ be a fresh set of variables, such that $\hat{Z} := \{\hat{z}_i \mid z_i \in Z\}$, and let $F\left(Z \mapsto \hat{Z}, A\right)$ be the formula in which every $z_i \in Z$ is replaced by its corresponding $\hat{z}_i \in \hat{Z}$. We assume *w.l.o.g.* that $Z := \{z_1, \ldots, z_m\}$, with $m = |Z|$. For $1 \leq i \leq m$, Padoa's theorem now defines the following formula:

$$\psi\left(Z, A, \hat{Z}, i\right) := F(Z, A) \wedge F\left(Z \mapsto \hat{Z}, A\right) \wedge \bigwedge_{j=1; j\neq i}^{m} (z_j \leftrightarrow \hat{z}_j) \wedge z_i \wedge \neg \hat{z}_i. \tag{1}$$

Intuitively, this formula asks if there exist at least two solutions to $F(Z, A)$, σ_1 and σ_2, such that $\sigma_{1\downarrow Z}$ and $\sigma_{2\downarrow Z}$ differ only in their value for z_i. If yes, then Eq. (1) is satisfiable. If no, then Eq. (1) is unsatisfiable.

2.2 Motivating Example

We model the sensor placement example from Section 1 as follows. First, we model the hotel as a graph $\Gamma := (V, E)$, where the nodes V represent rooms and two nodes u, v are connected by an edge (u, v) if the corresponding rooms are adjacent. Smoke detectors have a green light if they do not detect smoke, and have a red light if they do. All smoke detectors have a green light at $t_0 - 1$. We assume that at time t_0 a fire can break out in at most k different rooms ($k = 5$ in the example in Section 1), and that after t_0, no more fires break out. If there is a smoke detector placed in room v, and a fire breaks out in room v at time t_0, the smoke detector in room v detects the smoke at t_0, whereupon its detection light turns

$$\begin{tikzcd}
 & d \arrow[rr, dash] \arrow[dl, dash] & & e \arrow[dl, dash] \arrow[dr, dash] & \\
a^* \arrow[rr, dash] & & b \arrow[rr, dash] & & c^*
\end{tikzcd}$$

U	S^0_U	S^1_U
$\{a\}$	$\{a\}$	$\{a\}$
$\{b\}$	$\varnothing$	$\{a, c\}$
$\{c\}$	$\{c\}$	$\{c\}$
$\{d\}$	$\varnothing$	$\{a\}$
$\{e\}$	$\varnothing$	$\{c\}$
$\varnothing$	$\varnothing$	$\varnothing$

Figure 1: A graph and a GICS $D := \{a, c\}$ for $k = 1$, indicated with $*$. The table shows the signatures for all $U \subseteq V$ with $|U| \leq k$.

from green to red immediately, and remains red. A smoke detector placed in room $u \in N_1(v)$ detects the smoke from the fire in room v at $t_1 = t_0 + 1$. If its light was not yet red at time t_0, the light of the sensor in room u turns from green to red at time t_1. Hence, at time t_1 a sensor placed in room $v \in V$ is red iff there is a fire in at least one room in $N_1^+(v)$.

For a set of rooms $U \subseteq V$, we now have $s_U = \langle S^0_U, S^1_U \rangle$, where S^0_U represents the set of detectors whose lights turn red at t_0 if fires break out in all rooms in U at t_0, while S^1_U represents the set of detectors whose lights are red at $t_1 = t_0 + 1$. The GICS problem asks in which set of nodes $D \subseteq V$ to place a smoke detector, such that D is a GICS of Γ and k, and $|D|$ is minimised.

Example 1. *Figure 1 shows an example of five rooms, where we have chosen* $k = 1$. *We show a GICS for this example that places a sensor in rooms* a *and* c*, i.e.,* $D := \{a, c\}$. *The table shows the signature for each subset* $U \subseteq V$ *with* $|U| \leq 1$. *Note that each signature is unique, each non-empty subset has a non-empty signature, and that neither* a *nor* c *can be removed from* D *without destroying these two properties. This particular GICS has cardinality* 2*, which is the smallest possible cardinality for this network with* $k = 1$.

3 Related Work

Several methods have been proposed for solving a variant of the identifying code set problem that only considers a maximum identifiable set size of $k = 1$, and only requires the S^1_Us to be unique. A common approach [Sen *et al.*, 2019; Padhee *et al.*, 2020; Basu and Sen, 2021a; Basu and Sen, 2021b] models the problem as an *integer-linear program (ILP)*, to be solved with a *mixed-integer programming (MIP)* solver. We adapted the method from [Padhee *et al.*, 2020; Basu and Sen, 2021b] such that it can model the unique identification of $k > 1$ simultaneous events. The number of linear constraints in this encoding grows as $O\left(\binom{|V|}{k}^2\right)$, which is prohibitively large for all but the smallest of networks, especially if $k > 1$. We use the remainder of this section to describe our adapted encoding. To make the encoding more intuitive, we model the ICS problem as the *Unique Graph Colouring (UGC) problem*, which we introduce first. Then, we describe how to encode that problem, and finally we provide an analysis of the size of the resulting encoding.

3.1 The Unique Graph Colouring (UGC) Problem

Given an undirected graph $\Gamma = (V, E)$, we formulate a graph colouring problem as follows. At time t_0, each node can be injected with at most one colour. The injected colours are all distinct. At time $t_1 = t_0 + 1$, a colour c injected into node v seeps into all direct neighbours of v, but does not seep any further. Thus, nodes can get coloured through injection and/or seepage, and one node can be coloured with multiple distinct colours.

Since all injected colours are assumed to be unique, and each node can be injected with colour at most once, we name each colour after the node in which it is injected. Hence, if node v is injected with colour, it is injected with the colour c_v. We call the set of nodes that are injected with colour the *injection set* $D \subseteq V$. We assume that at $t_0 - 1$, all nodes are colourless.

Now, we can define a colour signature for a node $v \in V$ the following tuple: $s_{\{v\}} = \langle S_v^0, S_v^1 \rangle$, with $S_{\{v\}}^0 = \{c_v \mid v \in D\}$ and $S_{\{v\}}^1 = \{c_u \mid u \in N_1^+(v) \cap D\}$. Similarly, we can define the colour signature of a set of nodes $U \subseteq V$ as $s_U = \langle S_U^0, S_U^1 \rangle$, with $S_U^0 = \bigcup_{u \in U} S_{\{u\}}^0 = \{c_u \mid u \in U \cap D\}$ and $S_U^1 = \bigcup_{u \in U} S_{\{u\}}^1 = \{c_u \mid u \in N_1^+(U) \cap D\}$.

Given a positive integer $k \leq |V|$, the *unique graph colouring (UGC)* problem seeks to find an *injection set* $D \subseteq V$ of nodes that must each be injected with a distinct colour, such that the cardinality of D is minimised and the following constraints are satisfied:

1. At time t_1 no node is colourless (*i.e.*, $\forall v \in V$ we have $S_{\{v\}}^1 \neq \varnothing$).
2. For each pair of subsets (U, W), with $U, W \subseteq V$, $|U|, |W| \leq k$, and $U \neq W$, it holds that $s_U \neq s_W$.

3.2 Encoding the UGC problem in ILP

To model the unique graph colouring problem as an ILP, we first define Boolean *injection* variables $X := \{x_v \mid v \in V\}$, such that $x_v = 1$ iff node v is injected with a colour at time t_0. We also define *seepage* variables $Y = \{y_v \mid v \in V\}$ with integer domains, where for each $y_v \in Y$, we define the domain as $Dom(y_v) := [0, |N_1^+(v)|]$.

We now define a number of linear constraints to encode the UGC problem. First, we define the constraints that express the dynamics of seepage:

$$C_{seepage} := \left\{ y_v = \sum_{u \in N_1^+(v)} x_u \mid v \in V \right\}. \tag{2}$$

We must require that each node is labelled with at least one colour at time t_1, which we encode with the following set of constraints:

$$C_{alo} := \{y_v \geq 1 \mid v \in V\}. \tag{3}$$

Finally, we have to guarantee that for each pair of distinct, non-empty subsets (U, W), with $U, W \subseteq V$ and $|U|, |W| \leq k$, it holds that $s_U \neq s_W$. Observe that this requirement means that $S_U^0 \neq S_W^0$ or $S_U^1 \neq S_W^1$ must hold. This implies a constraint on some of the nodes in $N_1^+(U) \cup N_1^+(W)$, which we explain here and illustrate in Fig. 2.

For $S_U^0 \neq S_W^0$ to hold, we must guarantee that at least one node in the symmetric difference of U and W is injected with colour (the vertically hatched area in Fig. 2). To see why,

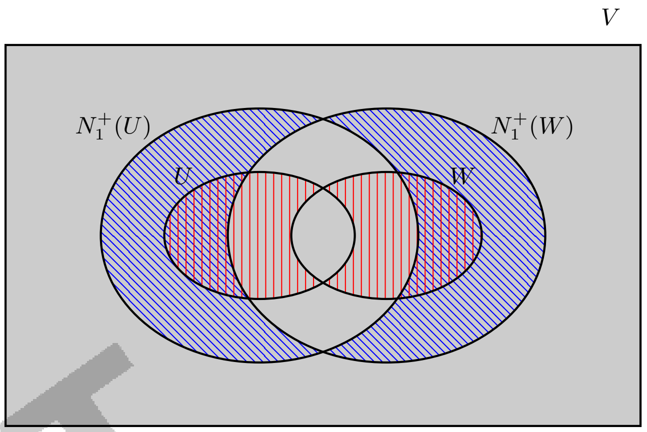

Figure 2: The rectangle represents the set of nodes V. The two small ellipses represent the sets of nodes $U, W \subseteq V$. The larger ellipses represent their respective closed 1-neighbourhoods, $N_1^+(U)$ and $N_1^+(W)$. A sensor placed in one of the nodes in the vertically hatched set will ensure that $S_U^0 \neq S_W^0$ holds. A sensor placed in one of the nodes in the diagonally hatched set will ensure that $S_U^1 \neq S_W^1$ holds. Hence, placing a sensor in any node in the hatched set will ensure that $s_U \neq s_W$ holds.

assume that, *w.l.o.g.*, we inject node $v \in U$ with colour c_v. Hence, $c_v \in S_{\{v\}}^0$ and thus $c_v \in S_U^0$. By the definition of $U \triangle W$, $v \notin W$. Hence, $c_v \notin S_W^0$, and thus $S_U^0 \neq S_W^0$.

For $S_U^1 \neq S_W^1$ to hold, we must guarantee that at least one node in $N_1^+(U) \triangle N_1^+(W)$ is injected with colour (the diagonally hatched area in Fig. 2). To see why, suppose we inject the colour c_v into node $v \in N_1^+(U) \triangle N_1^+(W)$. We can, *w.l.o.g.*, distinguish the following two cases:

1. $v \in U$ is injected with c_v. By the definition of $N_1^+(U) \triangle N_1^+(W)$, $v \notin N_1^+(W)$. Hence, $d(v, w) > 1$ for all $w \in W$. Consequently, $c_v \notin S_W^1$. Since $v \in U$, we have $c_v \in S_U^0$ and thus $c_v \in S_U^1$. Hence, $S_U^1 \neq S_W^1$.
2. $v \in N_1(U) \setminus U$ is injected with c_v. By the definition of $N_1^+(U) \triangle N_1^+(W)$, $v \notin N_1^+(W)$. Hence, $d(v, w) > 1$ for all $w \in W$. Consequently, $c_v \notin S_W^1$. Since $v \in N_1^+(U)$, we have $c_v \in S_U^1$. Hence, $S_U^1 \neq S_W^1$.

We thus introduce the following set of constraints:

$$C_{unique} := \left\{ \sum_{v \in (U \triangle W) \cup \left(N_1^+(U) \triangle N_1^+(W)\right)} x_v \geq 1 \mid U, W \subseteq V; U \neq W; 1 \leq |U|, |W| \leq k \right\} \tag{4}$$

Finally, we define the objective function as:

$$\text{minimise} \sum_{v \in V} x_v. \tag{5}$$

Hence, a solution $D \subseteq V$ to the UGC problem can be obtained from a solution $\sigma : X \mapsto \{0, 1\}$ to the ILP encoding of the problem as follows: $D = \{v \in V \mid \sigma(x_v) = 1\}$.

3.3 ILP Model Size

With the above definitions of the linear constraints in the ILP encoding of the UGC problem, we now make an estimate of the number of constraints needed for this encoding. Equation (2) contributes exactly $n = |V|$ constraints to the encoding, as does Eq. (3). The potentially largest contribution to the size of the ILP encoding, however, comes from Eq. (4), since it is comparing all subsets of sizes of 1 up to and including k to each other. Hence, the maximum number of constraints that it adds to the model can be expressed as follows:

$$|C_{unique}| = O\left(\sum_{i=1}^{k}\binom{n}{i}\cdot\left(\left(\binom{n}{i}-1\right)+\sum_{j=i+1}^{k}\binom{n}{j}\right)\right). \tag{6}$$

Since we have:

$$\begin{aligned}\sum_{i=1}^{k}\binom{n}{i}\cdot\left(\left(\binom{n}{i}-1\right)+\sum_{j=i+1}^{k}\binom{n}{j}\right)\\ \leq \sum_{i=1}^{k}\binom{n}{i}\cdot\left(\sum_{j=i}^{k}\binom{n}{j}\right),\end{aligned} \tag{7}$$

we can simplify Eq. (6) and write:

$$|C_{unique}| = O\left(\sum_{i=1}^{k}\binom{n}{i}\cdot\left(\sum_{j=i}^{k}\binom{n}{j}\right)\right). \tag{8}$$

In real-world applications, it is reasonable to assume that $k \leq {}^{|V|}/_{2}$. In that case, the largest term in Eq. (8) is $\binom{n}{k}^2$. Hence:

$$|C_{unique}| = O\left(\binom{|V|}{k}^2\right). \tag{9}$$

Note that, because Eq. (4) is a set, the number of constraints in practise may already be a lot smaller than that. However, to further limit the size of the ILP encoding, we attempted to minimise the number of constraints without affecting the correctness of the encoding.

We made an effort to minimise the model size of the ILP encoding by identifying redundant constraints and removing them from the encoding. We refer the interested reader to Appendix A for more details on that minimisation.

4 Approach

In this section, we discuss our novel approach to solving the GICS problem, which uses an encoding whose size *does not* explode, but rather grows polynomially with the problem size and k. We first introduce the *grouped independent support (GIS)*, an extension of *independent support* [Chakraborty *et al.*, 2014a; Ivrii *et al.*, 2016], then show how we can reduce finding a GICS to finding a GIS, and finally propose an algorithm for finding a GIS of minimised cardinality: GISMO.

4.1 Grouped Independent Support (GIS)

We define *grouped independent support (GIS)* as follows:

Definition 1. *Given a formula $F(Z, A)$, with $Z \cap A = \varnothing$ and a partitioning $\mathcal{G}$ of Z into non-empty sets, such that $sup(\mathcal{G}) = Z$. The subset $\mathcal{I} \subseteq \mathcal{G}$ is a* grouped independent support *of $\langle F(Z, A), \mathcal{G}\rangle$ if the following holds:*

$$\begin{aligned}&\forall \sigma_1, \sigma_2 \in Sol(F)\\ &\left((\sigma_{1\downarrow sup(\mathcal{I})} = \sigma_{2\downarrow sup(\mathcal{I})}) \leftrightarrow (\sigma_{1\downarrow Z} = \sigma_{2\downarrow Z})\right).\end{aligned} \tag{10}$$

Intuitively, this means that if all solutions projected on to $Z = sup(\mathcal{G})$ are unique, then all solutions projected onto $sup(\mathcal{I}) \subseteq sup(\mathcal{G})$ are unique, and vice versa. The '$\rightarrow$' in Eq. (10) means that, for all solutions to $F(Z, A)$, the truth values of the variables in $sup(\mathcal{G}) \setminus sup(\mathcal{I})$ are defined by the truth values of the variables in $sup(\mathcal{I})$. Note that GIS is a generalisation of independent support, since finding an independent support corresponds to finding a GIS where all the groups have cardinality 1.

Observe that the problem of checking whether a given set $\mathcal{I}$ is a grouped independent support is in co-NP. In contrast, checking whether an assignment satisfies ILP constraints is in polynomial time. Therefore, a priori, it is natural to wonder if it is worth reducing GICS to a (potentially) computationally harder problem. We pursue such a reduction in the hope that the reduction may come at the gain of smaller problem encodings. In the remainder of this section, we show that such a gain is indeed possible.

4.2 A Reduction from GICS to GIS

We now present a reduction from finding a GICS to finding a GIS, using the example problem from Section 2.2. Let $X := \{x_v \mid v \in V\}$ and $Y := \{y_v \mid v \in V\}$ be sets of Boolean variables such that $x_v = 1$ and $y_v = 1$ iff a sensor placed in room v has a red light at time t_0 and t_1, respectively. We capture this in the following Boolean formula:

$$F_{detection} := \bigwedge_{v\in V}\left(y_v \leftrightarrow \bigvee_{u\in N_1^+(v)} x_u\right). \tag{11}$$

Additionally, we must require that at most k fires break out at the same time, which we do with the following formula (recall that if a fire breaks out at time t_0 in room v, the light of a sensor placed in room v turns red at t_0):

$$F_{card,k} := \sum_{v\in V} x_v \leq k. \tag{12}$$

Converting these constraints to CNF and conjoining them, we obtain the following formula in CNF:

$$F_k(X \cup Y, A) = F_{detection} \wedge F_{card,k}, \tag{13}$$

where A is a (possibly empty) set of auxiliary variables needed for the CNF encoding of the cardinality constraint. Additionally, we define one group for each node in the network: $\mathcal{G} := \{G_v := \{x_v, y_v\} \mid v \in V\}$.

Now, we can find a GICS by encoding the problem into CNF according to Eq. (13), finding a GIS, and then extracting the sensor set as: $D := \{v \mid G_v \in \mathcal{I}\}$.

Lemma 1. *Given a loop-free, undirected network $\Gamma := (V, E)$ on nodes V and edges E, a maximum identifiable set size $0 < k \leq |V|$, and given a GIS of Eq. (13) $\mathcal{I} \subseteq \mathcal{G}$ with groups $\mathcal{G} := \{G_v := \{x_v, y_v\}\}$. The set $D := \{v \in V \mid G_v \in \mathcal{I}\}$ is a GICS of Γ.*

Proof. We prove this by showing that there is exactly one projected solution $\sigma_{U\downarrow sup(\mathcal{I})} \in Sol_{\downarrow sup(\mathcal{I})}(F_k)$ for each $U \subseteq V$ with $|U| \leq k$, and that each $\sigma_{U\downarrow sup(\mathcal{I})}$ maps to exactly one signature $s_U \in Sigs(\Gamma, k, D)$ for each $U \subseteq V$ with $|U| \leq k$. Specifically, we prove that the relations

$$R_1 : \mathcal{P}_{\leq k}(V) \mapsto Sol(F_k), \tag{14}$$

$$R_2 : Sol(F_k) \mapsto Sol_{\downarrow sup(\mathcal{I})}(F_k), \quad \text{and} \tag{15}$$

$$R_3 : Sol_{\downarrow sup(\mathcal{I})}(F_k) \mapsto Sigs(\Gamma, k, D) \tag{16}$$

are all bijective, where $\mathcal{P}_{\leq k}(V)$ denotes all $U \subseteq V$ with $|U| \leq k$.

Let us first prove R_1's bijectivity. Let $\sigma_U \in Sol(F_k)$ be a solution such that

$$\sigma_U(X_u) = \begin{cases} 1 & \text{if } u \in U \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

Note that, by Eq. (12), $|U| \leq k$ must hold for each $\sigma_U \in Sol(F_k)$. Note also that, by Eq. (11), $\sigma_U(X_u)$ defines $\sigma_U(Y_u)$. R_1's surjectvity and injectivity are evident by Eq. (17) and the fact that there are no other constraints on the variables in D than the ones in Eq. (11), hence, R_1 is bijective.

R_2's bijecivity is proved by observing that $\mathcal{I}$ is a GIS of F_k by assumption, and hence Eq. (10) must hold.

Finally, R_3's surjectivity and injectivity are evident from the observation that, by the interpretation of X and by Eqs. (11) and (17), we have:

$$\begin{aligned} \sigma_{U\downarrow sup(\mathcal{I})} := &\{x_u \mapsto 1 \mid x_u \in X \cap sup\,(\mathcal{G}_U \cap \mathcal{I})\} \\ &\cup \{x_u \mapsto 0 \mid x_u \in X \cap sup\,(\mathcal{I} \setminus \mathcal{G}_U)\} \\ &\cup \left\{y_u \mapsto 1 \mid y_u \in Y \cap sup\left(\mathcal{G}_{N_1^+(U)} \cap \mathcal{I}\right)\right\} \\ &\cup \left\{y_u \mapsto 0 \mid y_u \in Y \cap sup\left(\mathcal{I} \setminus \mathcal{G}_{N_1^+(U)}\right)\right\}. \end{aligned} \tag{18}$$

Hence, by the assumption that $D := \{v \in V \mid G_v \in \mathcal{I}\}$, we can construct $S_U^0 := \{u \mid x_u \in \sigma_{U\downarrow sup(\mathcal{I})} \wedge \sigma_{U\downarrow sup(\mathcal{I})}(x_u) = 1\} = \{u \mid x_u \in X \cap sup\,(\mathcal{G}_U \cap \mathcal{I})\} = U \cap D$, which is precisely the definition of S_U^0 from Section 2.1. Similarly, we can construct $S_U^1 := \{u \mid y_u \in \sigma_{U\downarrow sup(\mathcal{I})} \wedge \sigma_{U\downarrow sup(\mathcal{I})}(y_u) = 1\} = \left\{u \mid y_u \in Y \cap sup\left(\mathcal{G}_{N_1^+(U)} \cap \mathcal{I}\right)\right\} = N_1^+(U) \cap D$, which is precisely the definition of S_U^1 from Section 2.1. □

Intuitively, we showed that the highlighted columns in Table 1 encode the signatures in the table in Fig. 1, and vice versa. Each solution to F_k in Eq. (13) corresponds to selecting a set $U \subseteq V$ with $|U| \leq k$ as the set of nodes with failures. By Definition 1, $\sigma_{U\downarrow sup(\mathcal{I})} \neq \sigma_{W\downarrow sup(\mathcal{I})}$ for sets $U, W \in V$ with $|U|, |W| \leq k$. Therefore, a GIS $\mathcal{I}$ guarantees unique signatures for all such U, W. Hence, the uniqueness

Algorithm 1 The GISMO algorithm.

Input: Formula $F(Z, A)$ with partitioning of Z into $\mathcal{G}$, a time limit τ.
Output: GIS $\mathcal{I} \subseteq \mathcal{G}$.

```
 1: $E \leftarrow \{e_i \mid z_i \in Z\}$
 2: Initialise $\phi\left(Z, A, \hat{Z}\right)$                      ▷ Eq. (19)
 3: $Q \leftarrow sup\,(\mathcal{G})$
 4: $\mathcal{I} \leftarrow \varnothing$
 5: for $G \in \mathcal{G}$ do
 6:     $Q \leftarrow Q \setminus G$
 7:     $C \leftarrow Q \cup sup\,(\mathcal{I})$
 8:     $\xi \leftarrow \bigwedge_{z_i \in C} e_i$
 9:     for $z \in G$ do
10:         $\psi \leftarrow \phi \wedge \xi \wedge z \wedge \neg\hat{z}$  ▷ note similarity to Eq. (1)
11:         $sat \leftarrow$ CHECKSAT$(\psi, \tau)$
12:         if $sat$ then
13:             $\mathcal{I} \leftarrow \mathcal{I} \cup \{G\}$
14:             break
15: return $\mathcal{I}$
```

requirement is implicitly required by the semantics of GIS, and we do not need to encode it explicitly.

We can prove the following lemma by simple analysis of Eq. (13) and techniques for encoding cardinality constraints into CNF [Sinz, 2005; Philipp and Steinke, 2015], and refer the reader to the extended version of this paper for that proof:

Lemma 2. *$F_k(X \cup Y, A)$ has $O\,(k \cdot |V| + |E|)$ clauses.*

Proof. $F_{detection}$ is a conjunction of material equivalences. Using De Morgan's law, we can write the material equivalence for node v as a conjunction of $deg(v) + 1$ clauses, where $deg(v)$ is the degree of v. Hence, we need $|V| + \sum_{v \in V} deg(v) = 2 \cdot |E| + |V| = O(|E|)$ material equivalences to encode $F_{detection}$. Cardinality constraints like the one in $F_{card,k}$ can be encoded in different ways. The sequential unary counter [Sinz, 2005] uses $O(k \cdot |V|)$ clauses to encode a cardinality constraint. Hence, the total number of clauses needed to encode F_k is $O\,(k \cdot |V| + |E|)$. □

The above lemma highlights the potential exponential gains in encoding from GIS-based approach. While the ILP-based approach would lead to encodings with $O\left(\binom{|V|}{k}^2\right)$ constraints, our GIS-based approach requires only $O\,(k \cdot |V| + |E|)$ clauses.

4.3 Finding a GIS with GISMO

Algorithm 1 shows our algorithm for finding a GIS. On a high level, the algorithm iterates over all groups of variables and uses Padoa's theorem to determine if at least one of the variables in each variable group is *not* defined by other variables outside the group. If this is the case, the group must be part of a GIS. We now describe GISMO in more detail.

Recall Padoa's theorem from Section 2.1. By choosing $Z := sup\,(\mathcal{G}) = X \cup Y$, we can define ψ for Eq. (13). If, for an $1 \leq i \leq m$, $\psi\left(Z, A, \hat{Z}, z_i\right)$ is unsatisfiable, then we know the following: if a partial assignment $\sigma_{\downarrow Z\setminus\{z_i\}}$ can be extended to $\sigma_{\downarrow Z}$, then there is only one possible value that z_i can take in $\sigma_{\downarrow Z}$ such that σ is a model of Eq. (13). Hence, if ψ is unsatisfiable, then for each σ, the truth value of z_i is defined by the truth values of the variables $Z \setminus \{z_i\}$.

In Algorithm 1, we use Padoa's theorem as follows. We introduce a fresh set of indicator variables $E = \{e_i \mid z_i \in Z\}$ (Line 1), and define the following formula (Line 2):

$$\phi\left(Z, A, \hat{Z}\right) := F(Z,A) \wedge F\left(Z \mapsto \hat{Z}, A\right) \wedge \bigwedge_{j=1}^{m} e_j \rightarrow (z_j \leftrightarrow \hat{z}_j)\,. \tag{19}$$

In Line 3, we introduce Q, the set of candidate variables that could be in the support of the GIS $\mathcal{I}$ that is returned by GISMO. We initialise $\mathcal{I}$ with $\varnothing$.

The 'for'-loop that starts at Line 5 in Algorithm 1 iterates over the groups in partition $\mathcal{G}$. In each iteration, we define the set C, which contains the variables for which we want to check if they define the variables in the group G that is considered in that iteration. By design, the set $C \cup G$ is an independent support of $F(Z, A)$. In the 'for'-loop that starts at Line 9, we test for each variable $z \in G$ if that variable is defined by the variables in C, and thus if C is an independent support. If z is *not* defined by the variables in C (and hence ψ is satisfiable), we know that, given the current C, z is needed to define all solutions, and thus that C is *not* an independent support of $F(Z, A)$. Hence, we add z's entire group to the GIS $\mathcal{I}$ (in Line 13). If all variables in G are defined by the variables in C, then C is an independent support and none of the variables in G are needed to define all solutions, so G is not added to $\mathcal{I}$, and not considered again.

At the start of each iteration of the outer 'for'-loop, $Q \cup sup(\mathcal{I})$ is a set of variables that define the variables in $Z \setminus (Q \cup sup(\mathcal{I}))$. During the execution of the algorithm, more and more groups of variables are removed from Q, and some groups are added to I, if that is deemed necessary for $Q \cup sup(\mathcal{I})$ to still define the variables in $Z \setminus (Q \cup sup(\mathcal{I}))$. At the end of the algorithm, Q is empty, and hence all groups in $\mathcal{I}$ contain variables that are necessary for defining the variables in $Z \setminus sup(\mathcal{I})$. Hence, the $\mathcal{I}$ returned by the algorithm is a GIS for $\langle F(Z, A), \mathcal{G}\rangle$. Recall that D is a GICS, and that in our reduction, each group corresponds to a node. Therefore, intuitively, GISMO starts with $D = V$, and then removes nodes from D until no nodes can be removed without removing the GICS-ness of D.

The time limit τ in Line 12 is given in a maximum number of conflicts that the SAT solver may encounter before giving up. If the SAT solver reaches τ before it determines the (un)satisfiability of ψ, then ψ is treated as satisfiable. Hence, in practice it may happen that G is defined by the variables in C, but is nevertheless added to $\mathcal{I}$.

We refer the reader to the extended version of this paper for the proof of the following lemma:

Lemma 3. *Given an input formula $F(Z, A)$ with group partitioning $\mathcal{G}$ such that $sup(\mathcal{G}) = Z$, Algorithm 1 returns a GIS $\mathcal{I}$ of $\langle F(Z, A), \mathcal{G}\rangle$.*

	X, or S_U^0					Y, or S_U^1				
U	x_a	x_b	x_c	x_d	x_e	y_a	y_b	y_c	y_d	y_e
$\{a\}$	1	0	0	0	0	1	1	0	1	0
$\{b\}$	0	1	0	0	0	1	1	1	0	1
$\{c\}$	0	0	1	0	0	0	1	1	0	1
$\{d\}$	0	0	0	1	0	1	0	0	1	1
$\{e\}$	0	0	0	0	1	0	1	1	1	1
$\varnothing$	0	0	0	0	0	0	0	0	0	0

Table 1: The rows of the truth table of Eq. (13) that correspond to models of Eq. (13), for the small graph in Fig. 1. The grey columns highlight a cardinality-minimal GIS for this formula that corresponds to the GICS in Example 1.

If the call to CHECKSAT(ψ, τ) never times out, GISMO returns a set-minimal GIS of the input formula and partition. The cardinality of that GIS is potentially larger than the cardinality-minimal solution that is guaranteed by the ILP encoding proposed by [Padhee *et al.*, 2020].

Note the similarity of GISMO to the algorithm for high-level minimal unsatisfiable core extraction presented in [Nadel, 2010]. Indeed, finding a set-minimal independent support can be reduced to finding a group-oriented (or high-level) minimal unsatisfiable subset [Ivrii *et al.*, 2016].

We illustrate GISMO with an example, based on the problem in Example 1. To aid our discussion, we provide a truth table containing all solutions to Eq. (13) for the problem in Example 1, in Table 1.

Example 2. *Let $X := \{x_a, \ldots, x_e\}$, $Y := \{y_a, \ldots, y_e\}$, and $\mathcal{G} := \{G_a := \{x_a, y_a\}, \ldots, G_e := \{x_e, y_e\}\}$. Let $k = 1$, and let $F_1(Z, A)$ be defined as in Eq. (13), with $Z = sup(\mathcal{G})$.*

After initialisation, $Q = sup(\mathcal{G})$ and $\mathcal{I} = \varnothing$. Let us assume that the algorithm now selects group G_e as the first group to test. This causes both Q and C to be updated to $\{x_a, y_a, \ldots, x_d, y_d\}$, and $\xi := e_{x_a} \wedge e_{y_a} \wedge \ldots \wedge e_{x_d} \wedge e_{y_d}$. Let us assume that the algorithm first tests $y_e \in G_e$ for definability, constructing $\psi := \phi \wedge \xi \wedge y_e \wedge \neg\hat{y}_e$ and checking for satisfiability. We inspect Table 1 to check if ψ is satisfiable. As we can see in the table, there are no two rows that agree on the truth values of variables $\{x_a, y_a, \ldots, x_d, y_d\}$, but differ in their truth values of variable y_e. Hence, ψ is unsatisfiable, and the algorithm moves to the second iteration of the inner 'for'-loop to perform the same test for variable x_e, finding again that ψ is unsatisfiable.

The algorithm concludes that all variables in G_e are defined by the variables in C, and moves on to test the next group. Let us assume that the algorithm tests group G_d next. It finds that G_d also does not belong in the GIS, and moves on to group G_c. Now we have $Q = C := \{x_a, y_a, x_b, y_b\}$, $\xi := e_{x_a} \wedge e_{y_a} \wedge e_{x_b} \wedge e_{y_b}$, and $\psi := \phi \wedge \xi \wedge y_c \wedge \neg\hat{y}_c$.

Let us assume that the algorithm first checks $y_c \in G_c$. Inspecting Table 1, we find that there are no two rows that agree on their truth values for x_a, y_a, x_b and y_b, but disagree on their truth value for y_c. Hence, ψ is unsatisfiable, and the algorithm moves on to test x_c.

The rows $\{c\}$ and $\{e\}$ in Table 1 agree on their truth values for x_a, y_a, x_b and y_b, but disagree on their value for x_c. Consequently, ψ is satisfiable, and we update $\mathcal{I} := \{G_c\}$.

Let us assume that in the next iteration, the algorithm *checks group G_b. It finds that, for both x_b and y_b, ψ is unsatisfiable, so G_b is discarded and not added to the GIS. In the final iteration, we have $Q := \varnothing$, $C := \{x_c, y_c\}$ and $\xi := e_{x_c} \wedge e_{y_c}$. It is easy to see from Table 1 that $\psi := \phi \wedge \xi \wedge x_a \wedge \neg \hat{x}_a$ is satisfiable (inspect rows $\{b\}$ and $\{e\}$), and thus the algorithm updates and returns $\mathcal{I} := \{G_a, G_c\}$.*

Proof. Throughout the execution of Algorithm 1, the set $C = Q \cup sup(\mathcal{I})$ remains a GIS of $\langle F(Z), \mathcal{G}\rangle$. At initialisation, this set is equal to $C = Q = sup(\mathcal{G})$. In each iteration, the algorithm selects a group G (Line 5), removes the variables in G from C (Line 6), and checks if C is still a GIS of $F(Z)$ and $\mathcal{G}$ (Lines 9, 10 and 12 to 14). If not, the variables of G are added back to C (Line 13). If C still is a GIS without the variables in G, then the variables in G are *not* added back to C. In either case, G is never considered again, and in either case, C remains a GIS of $\langle F(Z), \mathcal{G}\rangle$, until the algorithm terminates. □

5 Experiments

In this section we describe our experiments aimed at evaluating the performance of GISMO, comparing it to the state-of-the-art ILP-based method.

5.1 Experimental Setup

Solving methods. We evaluate a method that encodes the problem into the CNF in Eq. (13) and then solves it by finding a GIS with GISMO. In this section, we refer to this method as 'GISMO'. We provide more details on the concrete implementation of the CNF encoding used for this method in Appendix C. We compare the performance of GISMO to an ILP-based approach, as discussed in Section 3. We use PBPBS, based on initials of authors [Padhee *et al.*, 2020], to refer to the ILP-based approach. We refer the reader to the extended version of this paper for details on implementation.

Software. Our implementation of GISMO uses SAT solver CryptoMiniSat [Soos *et al.*, 2009] version 5.11.7 (the latest version, last updated in December 2022) to determine the satisfiability of ψ in Line 12 in Algorithm 1. We implemented the scripts for encoding networks into CNF (Eq. (13)) or ILP (Section 3) with Python 3.5, using PBLib [Philipp and Steinke, 2015] for the CNF encoding of the cardinality constraint. We solved the ILPs with CPLEX 12.8.0.0.[1]

Hardware. We ran our experiments on a high-performance cluster, where each node is equipped with two Intel E5-2690 v3 CPUs, each with 12 cores and 96 GB RAM, running at 2.60 GHz, under Red Hat Enterprise Linux Server 6.10.

Experimental Parameters. We allow GISMO and PBPBS each one core, 3 600 CPU s and 4 GB RAM to encode and solve each (network, k) combination. For GISMO we set a time limit of $\tau = 5\,000$ conflicts for the call to CryptoMiniSat in Line 12. For CPLEX we use the default settings. The running times we report are all user time measured in CPU s.

[1]Available at www.ibm.com/analytics/cplex-optimizer.

Problem Instances. Our benchmark set comprises 50 undirected networks obtained from the *Network Repository* [Rossi and Ahmed, 2015] and from the *IdentifyingCodes* GitHub repository [Basu and Sen, 2021b].[2], including grid-like networks, such as road networks and power networks, and social networks, such as collaboration networks and crime networks. Their sizes vary from 10 to 1 087 562 nodes and 14 to 1 541 514 edges, and their median degrees vary from 1 to 78. See Appendix B for details on our benchmark set.

5.2 Research Questions

The experiments in this section are aimed at answering the following main research questions:

Q1 How many instances are solved by PBPBS and GISMO?

Q2 How do the solving times of PBPBS and GISMO scale with k and $|V|$?

Q3 How does the number of clause in the CNF encoding scale with k and $|V|$?

Q4 How do the cardinalities of the solutions returned by PBPBS and GISMO compare?

In summary, we find that GISMO solves nearly $10\times$ more problem instances than PBPBS within the time limit of 3600 seconds per problem instance. The instances that can be encoded by both methods are solved up to $520\times$ faster by GISMO than by PBPBS, depending on k. On these instances, we find that the solution returned by GISMO is at most 60% larger than that returned by PBPBS, but that most instances, the cardinality of the solution returned by GISMO is less than 10% larger. We find that the size of the CNF in Eq. (13) scales polynomially with $|V|$ and k. The largest problem that could be encoded and solved by PBPBS has 494 nodes. The largest problem that could be encoded and solved by GISMO has 21 363 nodes; a $\sim$ 40-fold improvement.

5.3 Experimental Results

In the remainder of this section, we describe our experimental results and answer our research questions.

Q1: Number of Solved Instances. We report the number of solved instances by GISMO and PBPBS for the 9 tested values of k in Table 2. An instance is solved if the solving method terminates before the timeout time. In the case of PBPBS, this means that the returned solution is cardinality-minimal. In the case of GISMO, this means that the solution is (close to)[3] set-minimal. Overall, GISMO solved 289 of the 450 problem instances, while PBPBS solved only 36 out of 450. Hence, the GISMO solves over 8 times as many problem instances as PBPBS. We now delve into the internals of GISMO and PBPBS: GISMO was able to encode the GICS problem into CNF (Eq. (13)) for each value of k for 49 of the 50 networks. The largest network it could encode into CNF has 227 320 nodes and 814 134 edges. We find that GISMO returned a GIS for most of these CNFs. On the other hand, PBPBS was able to encode at most 11 of the 50 benchmarks

[2]Available at https://networkrepository.com and https://github.com/kaustav-basu/IdentifyingCodes.

[3]Because of the time limit τ in Line 12 of Algorithm 1.

k	1	2	3	4	6	8	10	12	16
PBPBS	5 663 (11)	6 063 (8)	6 508 (5)	6 912 (2)	6 915 (2)	6 925 (2)	6 934 (2)	6 935 (2)	6 935 (2)
GISMO	969 (46)	3 146 (29)	3 346 (28)	3 425 (27)	3 097 (30)	3 062 (30)	2 959 (31)	2 940 (31)	2 393 (37)

Table 2: PAR-2 scores and number of solved instances (in parentheses) for PBPBS and GISMO, and for each value of k that we evaluated. The PAR-2 scores are given in CPU s, and we used a timeout of 3600 s. For each k, the total number of instances was 50.

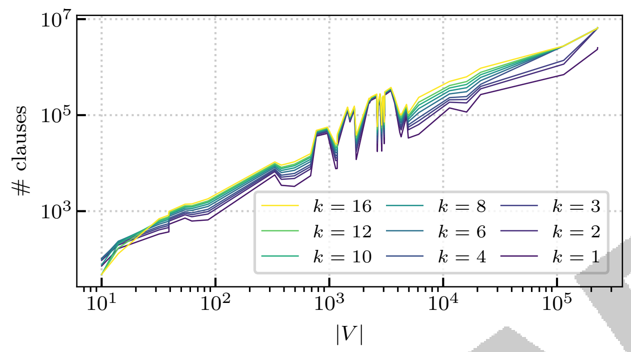

Figure 3: Number of clauses in the CNF encoding as a function of the number of nodes in the input network.

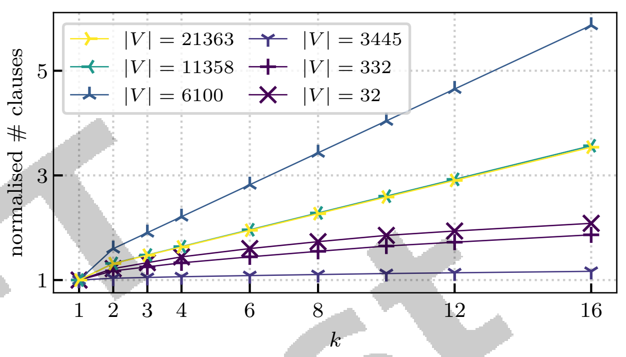

Figure 4: Normalised size of the CNF as a function of k, for selected networks. For each example, we divided the number of clauses in $F_k(Z, A)$ by the number of clauses in $F_1(Z, A)$, such that the model sizes are normalised *w.r.t.* the smallest model for each graph.

into an ILP, which was for $k = 1$ (it performed worse as k increased). The largest network that it could encode has 494 nodes and 1080 edges. For larger values of k, PBPBS's ability to encode the networks drops rapidly, being only able to encode and solve 2 out of the 50 networks for $k \geq 4$. These two networks are the smallest in our benchmark set, with only 10 an 14 nodes.

Q2: Solving Time. Table 2 compares the PAR-2 scores[4] of GISMO to those of PBPBS. GISMO is up to $\sim$ 6 times faster than PBPBS, in terms of PAR-2 scores, for smaller values of k and $2\times$ faster for larger values of k. Since PBPBS often times out during encoding phase (due to the blow-up of the size of the encoded formula), we also a provide comparison, in Table 3, for the instances for which the encoding phase of PBPBS did not time out and for which the underlying ILP solver did not timeout either, which was the case for all instances for which the encoding did not time out. It is worth remarking that all such instances were solved by GISMO as well. Here, we find that GISMO is up to $593.18/1.14 \approx 520\times$ faster than PBPBS in terms of median solving time. Overall, our results that GISMO achieves significant performance improvements over PBPBS, in terms of running time.

Q3: Model Size. Figure 3 shows that the number of clauses in the CNF encoding scales polynomially with the number of variables in the instance. The oscillations in the plot can be explained by the $|E|$-contribution to the CNF size (Lemma 2). Our benchmark set contains both social networks (with high median degrees) and grid-like networks (with low median degrees), and hence with different densities. Figure 4 shows typical examples of how the number of clauses in the CNF encoding grows with increasing k.

Q4: Solution Quality. We compared the quality of solutions returned by PBPBS and GISMO over the 36 instances that PBPBS could solve. In particular, we computed the ratio $r := |\mathcal{I}|/|D_{ILP}|$, wherein $\mathcal{I}$ is the set computed by GISMO, while D_{ILP} is the set computed by PBPBS. In our experiments, we found that $1 \leq r \leq 1.6$, but for the majority of instances we found $r < 1.1$. Furthermore, 4 out of 36 instances had a ratio $r = 1$. Hence, even in our naive implementation, the cardinalities of our solutions are almost as good as the minimum cardinality guaranteed by PBPBS.

6 Conclusion

In this paper, we focused on the problem of generalised identifying code set (GICS) problem which, given an input network, aims to find a set of nodes in which to place sensors in order to uniquely detect node failures, and where the number of placed sensors must be minimised. We first identified the primary bottleneck of the prior state-of-the-art approach based on an ILP encoding: the blowup in the encoding size. To address this shortcoming, we introduced *grouped independent support (GIS)* and reduced the GICS problem to the problem of finding a GIS of a Boolean formula. Relying on the fact that algorithms for finding a minimised independent support are fast in practice, we designed and implemented an algorithm, GISMO, that finds a minimised, though not necessarily cardinality-minimal, *grouped* independent support. Our empirical evaluation demonstrates that GISMO achieves significant performance improvements over the prior state-of-the-art technique in terms of running time, while producing

[4]The PAR-2 score is a penalised average runtime. It assigns a runtime of two times the time limit for each benchmark the tool timed out on, or ran out of memory on.

k (# instances)	1 (11)	2 (8)	3 (5)	4 (2)	6 (2)	8 (2)	10 (2)	12 (2)	16 (2)
PBPBS	143.59	96.53	367.69	5.45	87.00	349.56	557.71	577.94	593.18
GISMO	4.01	5.11	1.31	1.14	1.17	1.16	1.10	1.15	1.14

Table 3: Median solving times for the instances that could be encoded into ILP.

solutions that tend to be close to cardinality-minimal.

Acknowledgements

This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004], Ministry of Education Singapore Tier 2 grant MOE-T2EP20121-0011, and Ministry of Education Singapore Tier 1 Grant [R-252-000-B59-114]. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore www.nscc.sg. One of the authors (Sen) acknowledges the support of the National University of Singapore during his sabbatical at NUS. We thank Mate Soos for his help in debugging, Kaustav Basu for providing benchmark problems, and anonymous reviewers for their constructive feedback.

References

[Abío *et al.*, 2011] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. BDDs for pseudo-boolean constraints — revisited. In *SAT*, volume 6695 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2011.

[Abío *et al.*, 2013] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A parametric approach for smaller and better encodings of cardinality constraints. In *CP*, volume 8124 of *Lecture Notes in Computer Science*, pages 80–96. Springer, 2013.

[Bailleux *et al.*, 2006] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. A translation of pseudo boolean constraints to SAT. *J. Satisf. Boolean Model. Comput.*, 2(1-4):191–200, 2006.

[Basu and Sen, 2021a] Kaustav Basu and Arunabha Sen. Epidemiological model independent misinformation source identification. In *ICWSM Workshops*, 2021.

[Basu and Sen, 2021b] Kaustav Basu and Arunabha Sen. Identifying individuals associated with organized criminal networks: A social network analysis. *Soc. Networks*, 64:42–54, 2021.

[Chakraborty *et al.*, 2014a] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *AAAI*, pages 1722–1730. AAAI Press, 2014.

[Chakraborty *et al.*, 2014b] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Balancing scalability and uniformity in SAT witness generator. In *DAC*, pages 60:1–60:6. ACM, 2014.

[Eén and Sörensson, 2006] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *J. Satisf. Boolean Model. Comput.*, 2(1-4):1–26, 2006.

[Ivrii *et al.*, 2016] Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints An Int. J.*, 21(1):41–58, 2016.

[Karpovsky *et al.*, 1998] Mark G. Karpovsky, Krishnendu Chakrabarty, and Lev B. Levitin. On a new class of codes for identifying vertices in graphs. *IEEE Trans. Inf. Theory*, 44(2):599–611, 1998.

[Lagniez *et al.*, 2016] Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Improving model counting by leveraging definability. In *IJCAI*, pages 751–757. IJCAI/AAAI Press, 2016.

[Lagniez *et al.*, 2020] Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Definability for model counting. *Artif. Intell.*, 281:103229, 2020.

[Nadel, 2010] Alexander Nadel. Boosting minimal unsatisfiable core extraction. In *FMCAD*, pages 221–229. IEEE, 2010.

[Padhee *et al.*, 2020] Malhar Padhee, Reetam Sen Biswas, Anamitra Pal, Kaustav Basu, and Arunabha Sen. Identifying unique power system signatures for determining vulnerability of critical power system assets. *SIGMETRICS Perform. Evaluation Rev.*, 47(4):8–11, 2020.

[Padoa, 1901] A. Padoa. Essai d'une théorie algébrique des nombres entiers, précédé d'une introduction logique à une theorie déductive quelconque. *Bibliothèque du Congrès international de philosophie*, 3:309–365, 1901.

[Philipp and Steinke, 2015] Tobias Philipp and Peter Steinke. Pblib - A library for encoding pseudo-boolean constraints into CNF. In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 9–16. Springer, 2015.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[Sen *et al.*, 2019] Arunabha Sen, Victoria Horan Goliber, Kaustav Basu, Chenyang Zhou, and Sumitava Ghosh. On upper and lower bounds of identifying code set for soccer ball graph with application to satellite deployment. In *ICDCN*, pages 307–316. ACM, 2019.

[Sinz, 2005] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005.

[Soos and Meel, 2022] Mate Soos and Kuldeep S. Meel. Arjun: An efficient independent support computation technique and its applications to counting and sampling. In *ICCAD*, page 71. ACM, 2022.

[Soos *et al.*, 2009] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[Yang *et al.*, 2022] Jiong Yang, Supratik Chakraborty, and Kuldeep S. Meel. Projected model counting: Beyond independent support. In *ATVA*, volume 13505 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2022.

A Minimising the Number of ILP Constraints

We now describe "removing supersets", an optimisation that we implemented in our ILP encoding, aimed at limiting the number of constraints. The idea is to not add and/or remove redundant constraints that are satisfied if another constraint is satisfied.

Consider the set of constraints in Eq. (4). Each constraint is formulated over a set of nodes $Z := (U \triangle W) \cup \left(N_1^+(U) \triangle N_1^+(W)\right)$. Let $Z' \subset Z$ (with $Z' \neq \varnothing$) be a proper subset of Z. Note that if the constraint $\sum_{z \in Z'} x_z \geq 1$ is satisfied, then so is $\sum_{z \in Z} x_z \geq 1$. Hence, we can remove constraints that correspond to a superset of nodes that are already present as another constraint.

We implemented this as follows. While generating the C_{unique} set of constraints, each time we generate a new constraint C_{new}, we check if the nodes represented in it are a superset of the nodes represented in an existing constraint. If this is indeed the case, we simply do not add C_{new} to C_{unique}. If the nodes in C_{new} are not a superset of the nodes in an existing constraint, we check if there is an existing constraint on a set of nodes that is a superset of C_{new}. If yes, we remove that existing constraint from C_{unique} and replace it by C_{new}.

We found that using the "remove supersets" optimisation has a slight advantage over not using it, because it allowed us to encode one more network into ILP. When using the "remove supersets" optimisation, we find that the main reason that networks cannot be encoded into ILP is that the encoding script runs out of time (3600 seconds). When *not* using the "remove supersets" optimisation, we find that the main reason for failure to encode is that the script runs out of memory (4 GB).

Figure 5 shows that the size of the encoding can be reduced by several orders of magnitude by using the 'remove supersets' optimisation. However, this comes at the price of a longer encoding time, as is evidenced by the fact that both methods could encode problems that the other method could not encode (either because it ran out of time, or because it ran out of memory).

We also see that increasing k tends to increase the size difference, which is as expected. Increasing k also increases the probability that the network cannot be encoded into ILP at all, which is also as expected.

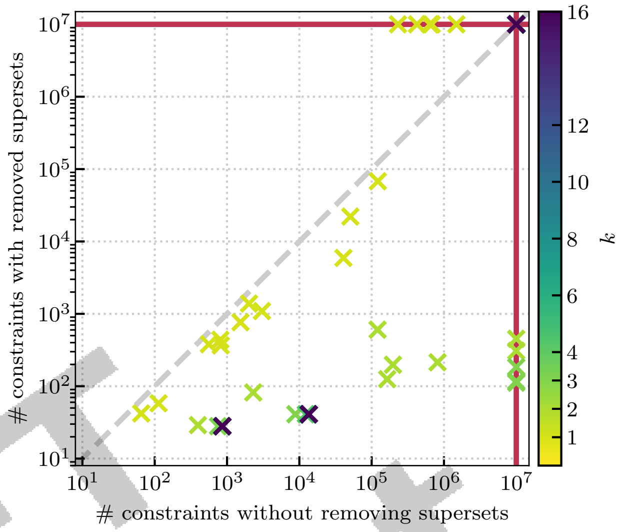

Figure 5: Comparison of the number of ILP constraints when removing super sets or not

B Benchmark set

Our benchmark set comprises 50 undirected networks obtained from the *Network Repository* [Rossi and Ahmed, 2015] and from the *IdentifyingCodes* GitHub repository [Basu and Sen, 2021b].[5], including grid-like networks, such as road networks and power networks, and social networks, such as collaboration networks and crime networks. We provide an overview in Table 4.

C CNF Encoding of the GICS Problem

We use PBLib [Philipp and Steinke, 2015] to encode the cardinality constraint in Eq. (12) into CNF. In PBLib's implementation, cardinality constraints can be encoded using Binary Decision Diagrams (BDDs) [Bailleux *et al.*, 2006; Eén and Sörensson, 2006; Abío *et al.*, 2011], adder networks [Eén and Sörensson, 2006] or cardinality networks [Abío *et al.*, 2013]. Encoding a cardinality constraint with degree k on n variables requires $O\left(n^2\right)$ clauses for the BDD encoding, $O\left(n \log^2 n\right)$ clauses for the adder network encoding, and $O\left(n \log^2 k\right)$ clauses for the cardinality network encoding. PBLib automatically chooses the best encoding for the given constraint. Hence, in our implementation, the actual number of clauses needed to encode Eq. (13) into CNF may be more than $O(k \cdot |V| + |E|)$. However, the number of clauses is still at most $O(|V|^2 + |E|)$, which is polynomial in the size of the network, and independent of k.

D Encoding Versus Solving

We ran experiments in which we separated the encoding phase and the solving phase for both GISMO and PBPBS, so we could look at them in a bit more detail. Table 5 shows the

[5]Available at https://networkrepository.com and https://github.com/kaustav-basu/IdentifyingCodes.

network	$\|V\|$	$\|E\|$	$med(d)$
ParisAdj.txt	10	14	2.5
PhilippineAdj.txt	14	51	8
soccer-ball.txt	32	90	6
road-chesapeake.mtx	39	170	7
ZerkaniAdj.txt	39	89	4
MadridAdj.txt	54	226	6.5
soc-dolphins.mtx	62	159	5
ca-sandi-auths.mtx	86	124	2
inf-USAir97.mtx	332	2 126	5
ca-netscience.mtx	379	914	4
power-494-bus.mtx	494	1 080	4
power-685-bus.mtx	685	1 967	5
socfb-Caltech36.mtx	769	16 656	36
socfb-Reed98.mtx	962	18 812	29
power-1138-bus.mtx	1 138	2 596	4
road-euroroad.edges	1 174	1 417	2
power-eris1176.mtx	1 176	9 864	5
socfb-Haverford76.mtx	1 446	59 589	70
socfb-Simmons81.mtx	1 518	32 988	37
socfb-Swarthmore42.mtx	1 659	61 050	59
power-bcspwr09.mtx	1 723	4 117	4
socfb-Amherst41.mtx	2 235	90 954	70
socfb-Bowdoin47.mtx	2 252	84 387	65
socfb-Hamilton46.mtx	2 314	96 394	72
socfb-Trinity100.mtx	2 613	111 996	76
road-minnesota.mtx	2 642	3 303	2
socfb-USFCA72.mtx	2 682	65 252	37.5
socfb-Williams40.mtx	2 790	112 986	70
socfb-nips-ego.edges	2 888	2 981	1
socfb-Oberlin44.mtx	2 920	89 912	50
socfb-Wellesley22.mtx	2 970	94 899	52
socfb-Smith60.mtx	2 970	97 133	56
web-edu.mtx	3 031	6 474	3
socfb-Vassar85.mtx	3 068	119 161	66.5
socfb-Middlebury45.mtx	3 075	124 610	71
socfb-Pepperdine86.mtx	3 445	152 007	71
socfb-Colgate88.mtx	3 482	155 043	78
socfb-Mich67.mtx	3 748	81 903	30
ca-GrQc.mtx	4 158	13 422	3
web-EPA.edges	4 271	8 909	2
web-spam.mtx	4 767	37 375	6
power-US-Grid.mtx	4 941	6 594	2
ca-Erdos992.mtx	6 100	7 515	1
web-indochina-2004.mtx	11 358	47 606	5
web-webbase-2001.mtx	16 062	25 593	1
ca-CondMat.mtx	21 363	91 286	5
road-luxembourg-osm.mtx	114 599	119 666	2
ca-dblp-2010.mtx	226 413	716 460	4
ca-citeseer.mtx	227 320	814 134	4
road-roadNet-PA.mtx	1 087 562	1 541 514	3

Table 4: The 50 networks in our benchmark set, sorted by number of nodes. We also report the number of edges and the median degree.

k	1	2	3	4	6	8	10	12	16
PBPBS (enc)	11 257 (11)	12 098 (8)	12 984 (5)	13 824 (2)	13 827 (2)	13 837 (2)	13 846 (2)	13 847 (2)	13 847 (2)
PBPBS (solve)	97 (11)	86 (8)	39 (5)	0 (2)	4 800 (2)	8 640 (2)	8 640 (2)	8 640 (2)	8 640 (2)
GISMO (enc)	188 (49)	203 (49)	195 (49)	197 (49)	204 (49)	194 (49)	196 (49)	194 (49)	195 (49)
GISMO (solve)	812 (46)	2 919 (33)	3 214 (30)	3 158 (31)	2 871 (33)	2 403 (37)	2 177 (41)	2 101 (41)	1 755 (42)

Table 5: PAR-2 scores and number of encoded or solved instances (in parentheses) for GISMO and PBPBS, and for each value of k that we evaluated. The PAR-2 scores are given in CPU s, and we used a timeout of 3600 s for the encoding phase and a timeout of 7200 s for the solving phase. For each k, the total number of instances was 50.

PAR-2 values for the encoding phase and the solving phase for both methods. Note that, if the encoding phase could encode n out of 50 instances, the PAR-2 value of the solving phase is computed over only those n instances, not the total number of 50 instances. Consequently, even though it looks like PBPBS solves instances much faster than GISMO for $k = 1$, the PAR-2 value for PBPBS is only computed for the 11 smallest instances that PBPBS could encode into ILP, while the PAR-2 value for GISMO is computed for the 49 (mostly larger) networks that could be encoded into CNF.

For the instances that could be encoded into ILP, we report the median encoding and solving times in Table 6, for PBPBS and GISMO. Here, we clearly see that the encoding part is hard for PBPBS as k increases. As for solving, we find that for most instances, the solving times of both methods are so fast as to not be distinguishable.

Comparing GISMO to PBPBS in terms of running time may be somewhat unfair, since PBPBS guarantees a cardinality-minimal solution, where GISMO does not. Both methods can be used as anytime algorithms, since they report the cardinality of the smallest solution found so far. Hence, it would be interesting to record how much time it would take PBPBS to find a solution of the same cardinality as the best solution found by GISMO, and record that in a table like Table 2. While CPLEX does log the cardinality of the best solution found so far, the added timestamps are in terms of iterations, not seconds, so it is hard to make that comparison. Furthermore, when inspecting Table 6, we see that this experiment might only make sense for the instances with very small k. However, looking at the encoding times for the PBPBS, we find that they tend to be so much longer than the total time it takes GISMO to encode and solve the problem, that even if CPLEX were configured to return whenever it finds a solution of the same quality as GISMO does, it would still be several orders of magnitude slower than GISMO.

k (# instances)	1 (11)	2 (8)	3 (5)	4 (2)	6 (2)	8 (2)	10 (2)	12 (2)	16 (2)
PBPBS (enc)	2.06	4.31	167.71	5.45	86.10	349.56	557.71	577.94	593.18
GISMO (enc)	4.00	5.10	1.31	1.14	1.17	1.16	1.10	1.15	1.14
PBPBS (solve)	122.31	64.71	0.00	0.00	0.00	0.00	0.00	0.00	0.00
GISMO (solve)	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
PBPBS (total)	143.59	96.53	367.69	5.45	87.00	349.56	557.71	577.94	593.18
GISMO (total)	4.01	5.11	1.31	1.14	1.17	1.16	1.10	1.15	1.14

Table 6: Median encoding and solving times for the instances that could be encoded into ILP (extended version of Table 3).