Università di Firenze, Università di Perugia, INdAM consorziate nel CIAFM

DOTTORATO DI RICERCA
IN MATEMATICA, INFORMATICA, STATISTICA

CURRICULUM IN MATEMATICA
CICLO XXXVI

Sede amministrativa Università degli Studi di Firenze
Coordinatore Prof. Matteo Focardi

Reconstruction problems on graphs and discrete sets: theoretical results and algorithms

Settore Scientifico Disciplinare MAT/03

Dottorando:
Niccolò Di Marco

Tutore
Prof. Andrea Frosini

Coordinatore
Prof. Matteo Focardi

Anni 2020/2023

Contents

Chapter 1

Introduction

This PhD thesis covers research topics belonging to *Graph Theory* and *Discrete Tomography*, two fields of discrete mathematics. Nowadays, they both have become fundamental research areas and constitute a bridge between theoretical computer science and mathematics.

Graph Theory and Discrete Tomography have many points in common since, usually, graphs can be considered as a topological generalization of what is generally studied in Discrete Tomography (e.g. binary matrices, discrete planes, etc.). However, this latter area is more involved in *reconstruction problem*, which asks to retrieve some spatial information starting from some type of aggregate measurements.

The following sections will introduce the main goals and objectives of these two areas, focusing on the problems that are covered here.

1.1 Graph and Hypergraph Theory

Graphs are mathematical structures that model relationships between objects and *Graph Theory* is the discipline that studies their geometrical, topological and statistical properties.

In particular, a graph is an ordered pair that consists of a set of objects (called *vertices* or *nodes*) and a set of relations between them (called *edges*).

The story of graph theory begins with a puzzle known as the Seven Bridges of Königsberg, which posed the question of whether it was possible to take a walk through the town of Königsberg crossing each one of its bridges on the Pregel river exactly once, returning to the starting point.

The puzzle was solved by Euler in 1735, who approached the problem analytically by modelling land masses as vertices and bridges as edges, using for the first time the concept of *graph*. In general, mathematicians initially deal with graphs as recreational objects, studying them as puzzle games [1].

Quickly after this event, many mathematicians and computer scientists started to study these objects, focusing also on the computational complexity of problems related to them. Table 1.1 shows examples of problems on graphs and their relative complexity. However, many others are still unsolved, as for the *graphs isomorphism* problem.

Problem	**Complexity**
Hamiltonian cycle	$NP - complete$ [2]
Eulerian cycle	Polynomial [3]
Graph isomorphism	Unknown
Clique detection	$NP - complete$ [4]
Vertex cover	$NP - complete$ [4]

Table 1.1: Examples of problems on graphs and their complexity.

Mathematicians noted also that this type of structure has an enormous modelling potential since they can represent important phenomena in different areas of mathematics and computer science [5, 6, 7, 8, 9, 10]. They also find application in several further areas such as chemistry, psychology, sociology and economy [11, 12, 13, 14].

Recently, the concept of *hypergraph* has been introduced as a generalization of graphs. In particular, *hypergraphs* are composed, similarly to graphs, by a set of vertices and a set of edges. The main differences consist in allowing edges to contain a general number of vertices, thus modelling more than dyadic interactions. Obviously, this new notion depicts much more complex scenarios. For this reason, the simplified concept of $k-$uniform hypergraph is usually considered, i.e. a hypergraph in which each hyperedge contains exactly k vertices. In particular, a graph can be considered as a $2-$uniform hypergraph.

In this PhD thesis, we consider reconstruction problems on hypergraphs and graphs. This class of problem roots in the recognition of graphs from their degree sequences. This question has been a challenging problem whose solution dates back to the well-known result of Erdös and Gallai [15] in 1960: an integer sequence $d = (d_1, \ldots, d_n)$ is graphic (i.e. it exists a graph G with that degree sequence) if and only if $\sum_{i=1}^{n} d_i$ is even and

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min\{k, d_i\}, \quad 1 \leq k \leq n.$$

This same problem related to hypergraphs remained open until 2018 when Deza et al. proved its NP-completeness [6] even in the simplest case of 3-uniform hypergraphs.

Before this result, many necessary and a few sufficient conditions were present in the literature and they mainly rely on a result by Dewdney [16]. As an example, Behrens et al. [17] proposed a sufficient and polynomially testable condition for a degree sequence to be k-graphic; their result still does not provide any information about the associated k-hypergraphs. Soon after, in [18, 19, 20], a series of polynomial time algorithms were proposed to reconstruct one of the k-hypergraphs associated with each degree sequence of some classes including that studied in [17].

These type of problems deals with the determination of geometrical properties of unknown objects, usually modelled by binary matrices, from their projections, i.e., quantitative measurements of the number of primary constituents along prescribed directions (see [21, 22] for the main results and the open problems). So, the reconstruction of a k-hypergraph from its degree sequence can be translated in the reconstruction of a binary matrix, i.e., its incidence matrix, from horizontal and vertical projections, i.e., the constant vector of entries k and the degree sequence, respectively.

However, some relevant related questions remain open, in particular, the study of the uniqueness (up to isomorphism) of k-hypergraphs sharing the same degree sequence.

After Chapter 2 in which we provide basic definitions and results, in Chapter 3 we deal with this problem by considering two hypergraphs H_1 and H_2 with the same degree sequence. Their symmetric difference $H_1 \ominus H_2$ produces a null hypergraph when assigning a $+1$ and -1 label to each hyperedge of H_1 and H_2, respectively. Vice versa, given a null hypergraph H, call H_1 the hypergraph with the same vertex as H but only the positive hyperedges of H and H_2 the same but with only the negative hyperedges of H. It's easy to see that H_1 and H_2 have the same degree sequence. In [23], the notion of null hypergraph has been used to study the changes performed on hyperedges that allow one to move through all the 3-hypergraphs with the same degree sequence. We link the null label of a k-hypergraph with its 2-intersection graph, showing that the existence of a Hamiltonian cycle in the 2-intersection graph is sufficient to define a null label of the related k-hypergraph.

Following this line of research, we focused on classes of intersection graphs, that were revealed to be a useful tool in the null label problem. In particular, intersection graphs are constructed by general hypergraphs considering connections among the hyperedges. These objects have been studied also in previous work, where the main focus was to determine the complexity of recognizing if a graph can belong to a certain class of intersection graphs (see [24] for an example).

Following these footsteps, in Chapter 4 we detail the known results about these objects. In particular, we prove that is $NP-$complete to decide if a certain graph is the $2-$intersection graph of a $3-$uniform hypergraph and the same holds considering claw-free graphs. However, moving to triangulated claw-free graphs, the problem can be solved polynomially.

To further extend the connection with the Discrete Geometry field, in Chapter 5 we consider the *Minimum Surgical Probing* (MSP) problem, which asks to retrieve some numerical values assigned to each vertex of a hypergraph using aggregate measures over the set of neighbours of each node. This question arises from the *Microscopic Image Reconstruction* problem, well-known in Discrete Geometry and it has been recently introduced and solved for graphs [25] having labels with real values.

We finally extend the framework to hypergraphs and we solve the problem for classes of hypergraphs defined according to their $2-$intersection graph. Since in general the problem is $NP-$hard if binary labels are considered, we decided to step back to graphs and consider graph convexities. In particular, in the final section of the chapter, we consider MSP_{conv}, a variant in which we ask to reconstruct binary labels arranged to form a convex subset of the graph under consideration.

1.2 Discrete Tomography

Discrete Tomography (DT) is a field of research that emerged in the last 30 years when researchers in Material Science and Electron Microscopy dealt with reconstruction problems starting from projections collected by crossing a material with a beam that counts the number of atoms crossed [26]. They intended to use the same strategies as in Computerized Tomography (CT) in order to reconstruct the 3D structure of different materials. However, CT algorithms have been created to deal with materials at a scale in which they can be assumed continuous and poorly adapt for a level where the set of atoms is closer to a discrete set of points. This issue, together with other difficulties, led to the development of tomography techniques that can be used to reconstruct discrete objects. For example, see [27, 28] for crystalline structures analyzed through DT perspective.

In this context, the technologies gave the impulse to study a new range of questions dealing with the reconstruction of discrete sets of points, initiating *Discrete Tomography*.

So, DT concerns the retrieval of geometrical information about the internal (and so sometimes inaccessible) structure of combinatorial objects from quantitative measurements of their primary constituents along linear (or multidimensional, in general) subspaces. These measurements are usually addressed as *projections*. The involved combinatorial objects may vary from generic discrete sets to constrained ones as graphs or hypergraphs ([21, 23]). Among the studied problems in DT, we find the retrieval of necessary and sufficient conditions for a pair of vectors to be the horizontal and vertical projections of an $m \times n$ binary matrix. Since, in general, the number of matrices sharing the same projections grows exponentially with their dimension, in most applications some further information is needed to obtain a solution as close as possible to the original object. For example, it has been proven that reconstructing discrete set from three or more projections is $NP-$complete [29]. Therefore, research tackles the algorithmic challenges of limiting the class of possible solutions in different ways, e.g. increasing the number of projections or adding geometrical information, like adding convexity constraints.

Among connected (finite) sets a dominant role deserved to *polyominoes*, that are

$4-$connected (i.e. vertical and horizontally connected) sets of points of the integer lattice, considered up to translation.

In [30] the authors present an algorithm (called, from now on, $HVRec$) to reconstruct hv-convex set in polynomial time starting from their horizontal and vertical projections. This algorithm consists of two separate parts: it first reconstructs an internal $hv-$convex kernel of points which is common to all the convex polyominoes having the input projections. Then, it expands the kernel to reach the desired projections maintaining the $hv-$convexity by means of a $2-SAT$ formula φ, one of whose valuations corresponding to one of the searched solutions can be computed in polynomial time.

On the other hand, the computational complexity of the reconstruction of full convex polyominoes is still an open problem, but it may benefit from the strategy described above. Interestingly, the problem of the uniqueness for (fully) convex sets has been solved in [31], where the authors find that in general 7 directions are needed to obtain a unique reconstruction, since in some particular cases 6 directions create ambiguities.

In particular, in Chapter 6 we consider this problem by defining a generalization of $HVRec$, called $CRec$. The idea is the following: starting from a couple of horizontal and vertical projections our algorithm executes the kernel reconstruction performed by $HVRec$ with further inclusion of the points in its *convex hull*, so obtaining a convex kernel.

The last part of the algorithm expands the kernel reaching the desired projections by using, in general, a formula whose valuations represent all the possible solutions to the reconstruction problem. Unfortunately, shifting to full convexity, this formula, say φ_{Conv}, becomes a generic SAT one, sliding its computational complexity to the non-polynomiality, assuming that $P \neq NP$. We deepen the characteristics of φ_{Conv} and offer a perspective to decrease its computational complexity.

In [32, 33], the authors stressed that, differently from the hv-convex polyominoes reconstruction, the formula φ_{Conv} imposes convexity on a specific region of the border of the kernel without, in general, providing the global convexity of the whole border. This observation turns out to be the main reason that prevents the extension of $HVRec$ to the case of convex polyominoes and the motivation of our study.

In the final section of Chapter 6 we consider the retrieval of $hv-$convex polyominoes using scans obtained from square and cross windows. This problem is called ***Microscopic Image Reconstruction*** and has been defined in [34]. In particular, the authors consider projections taken over a rectangular scanning window W.

As previously discussed, an extension of this problem, called *Minimum Surgical Probing* (MSP), has been initially introduced in [25]. The authors consider a graph instead of a set of points lying in the $\mathbb{Z}^2$ grid, where each vertex of the $G = (V, E)$ is associated with a real value ℓ_v. The task is to find the minimum number of surgical probes (i.e., the minimum number of known points' values) to uniquely recover ℓ_v for each node $v \in V$ given the neighborhood probes (projections) $P_v = \sum_{u \in N[v]} \ell_u$, with $N[v]$ being the neighbors of v. This question arises from the fact that, in general, there are multiple configurations satisfying the same projections. In particular, the uniqueness of the reconstruction depends on the rank of the associated adjacency matrix of the considered graph G.

As one can easily imagine, MSP can be naturally adapted to binary matrices and polyominoes to deal with cases in which no unique solution exists to the problem. In general, different matrices satisfy the same projections, and therefore it becomes relevant to find the minimum number of surgical probes needed to uniquely determine it.

Keeping the framework, our work concerns the MSP problem on polyominoes. More precisely, we investigate the possibility of a fast and faithful reconstruction of hv-convex polyominoes using projections obtained from square or diamond windows, i.e., from the knowledge of the 8 or 4-neighborhood aggregate values, respectively.

We provide some algorithms to perform the reconstruction task: if the dimensions of the unknown hv-polyomino are greater than 5, then the faithful reconstruction is achieved without surgical probes; otherwise, at most two surgical probes are required. This same study related to 4-neighbours scans is then performed, obtaining analogous results.

Chapter 2

Basic Notions

In this chapter, we introduce the main definitions and set the notation used throughout the Thesis, providing some results that introduce the research lines.

2.1 Graphs and hypergraphs

A *graph* G is defined as a pair $G = (V, E)$ such that $V = \{v_1, \dots, v_n\}$ is the set of *vertices* or *nodes* and $E \subseteq V \times V$ is a collection of pairs of vertices called *edges*. For each $v \in V$, we define the *open neighbourhood* $N(v) = \{w \in V \mid (v, w) \in E\}$. Similarly, we define $N[v] = N(v) \cup \{v\}$ its *closed neighbourhood*. The *degree* of $v \in V$ is $d(v) = |N(v)|$.

For $S \subseteq V$, let $G[S]$ denote the subgraph of G *induced* by S, which has vertex set S and edge set $\{(u, v) \in E \mid u, v \in S\}$. For $v \in V$, we write $G - v = G[V \setminus \{v\}]$. Similarly, for $S \subsetneq V$ and $v \in V \setminus S$ we write $G[S] + v = G[S \cup \{v\}]$. For $e \in E$, we write $G - e = (V, E \setminus \{e\})$.

A subset $S \subseteq V$ is a *clique* if $G[S]$ is a *complete graph*, i.e., every pairwise distinct vertices $u, v \in S$ are adjacent. We denote with K_p the clique on p vertices and we indicate K_3 as a *triangle*. $K_{1,p}$ is the star on $p + 1$ vertices, that is, the graph with $V = \{u, v_1, v_2, \dots, v_p\}$ and $E = \{(u, v_1), (u, v_2), \cdots, (u, v_p)\}$. In particular, a special case is $K_{1,3}$, known as *claw*. For example, consider the graph G depicted in Figure 2.1. $G[\{v_1, v_2, v_3\}]$ is a triangle while $G[\{v_4, v_5, v_6, v_7\}]$ is a claw.

For $S \subset V$ the clique $G[S]$ is *maximal* if for any $v \in V \setminus S$ then $G[S] + v$ is not a clique.

A *path* is a sequence of distinct vertices such that two consecutive vertices are connected by an edge, i.e. $P_k = u_1 u_2 \dots u_k$, with $(u_i, u_{i+1}) \in E$. A *cycle* C is a path in which the first and the last nodes are equal, i.e. $u_1 = u_k$. A cycle is *Eulerian* if it crosses each edge exactly once. On the other hand, a cycle is *Hamiltonian* if it crosses each vertex exactly once. For $k \geq 1$, $P_k = u_1 u_2 \cdots u_k$ is a *chordless* path if no two

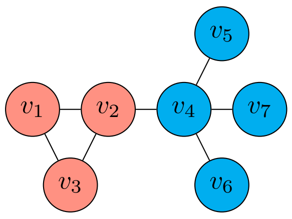


Figure 2.1: The subgraph induced by $\{v_1, v_2, v_3\}$ forms a triangle. On the other hand, the subgraph induced by $\{v_4, v_5, v_6, v_7\}$ forms a claw.

vertices are connected by an edge that is not in P_k, i.e. if $V_{P_k} = \{u_1, \ldots, u_k\}$ then $G(V_{P_k}) = P_k$. Finally, a path $P_k = u_1 u_2 \ldots u_n$ is *triangular* if each chord has the form $(u_i, u_{i+2}) \in E, 1 \leq i \leq n-2$. Similar definitions hold for chordless and triangulated cycles. For example, consider the graph depicted in Figure 2.3. The path $ubcv$ is a shortest path, while $udegv$ is a chordless path (it is not a shortest path, but it does not contain any chord). Finally, the path $uabcv$ is a triangle path. In fact, only a short chord is present.

A chordless cycle of length greater than 4 is called a *hole*. A graph without a hole is *chordal* or, equivalently, *triangulated*. A *cut-edge* in a connected graph G is an edge $e \in E$ such that $G - e$ is not connected.

For a fixed graph H we write $H \subseteq G$ whenever G contains an induced subgraph isomorphic to H. Instead, G is H*-free* if G has no induced subgraph isomorphic to H.

The notion of graph can be generalized to that of hypergraph by removing the constraint on the cardinality of the edges: a hypergraph $H = (V, E)$ is formed by a set $V = \{v_1, \ldots, v_n\}$ of vertices and a set $E \subseteq \mathcal{P}(V)$ of *hyperedges*. In the sequel we indicate them, with an abuse of notation, as edges. The notion of neighbours and degree simply generalize the ones of graphs.

In the sequel, we will consider only graphs and hypergraphs that are *simple*, i.e., they do not allow multiple edges and an edge cannot contain the same vertex multiple times. A hypergraph whose hyperedges have fixed cardinality is called k*-uniform*, or simply a k-hypergraph (in particular, the case $k = 2$ corresponds to graphs).

The degree sequence $(d_1, d_2, \ldots, d_n)$ of a hypergraph is the list of its vertex degrees arranged in non-increasing order.

Given two hypergraphs H_1 and H_2 we define their *symmetric difference* $H_1 \ominus H_2$ as the hypergraph having the same vertex set and edge set equal to $E_1 \ominus E_2 := (E_1 \setminus E_2) \cup (E_2 \setminus E_2)$.

2.1.1 Classes of intersection graphs

Given a $k-$hypergraph H, it is possible to use the following notion to construct new graphs related to H.

Definition 1. Given a k-uniform hypergraph $H = (V, E)$, its l-intersection graph $G = L_k^l(H), 1 \leq l < k$ is a graph $G = (E, F)$ such that the vertex set is E and $ee' \in F$ if and only if $|e \cap e'| = l$, i.e. two hyperedges of H share exactly l elements.

In general, we define $\mathcal{L}_k^l$ as the class of graphs G such that there exists a k-uniform hypergraph H such that $G = L_k^l(H)$. In such a case we say that G has the $l-$intersection property or it is *reconstructable*. In particular, $\mathcal{L}_2^1$ is called the class of the *line graphs*, usually denoted as $L(G)$, widely studied in *Graph theory*. Figure 2.2 shows an example of a graph G and its $L_2^1(G)$.

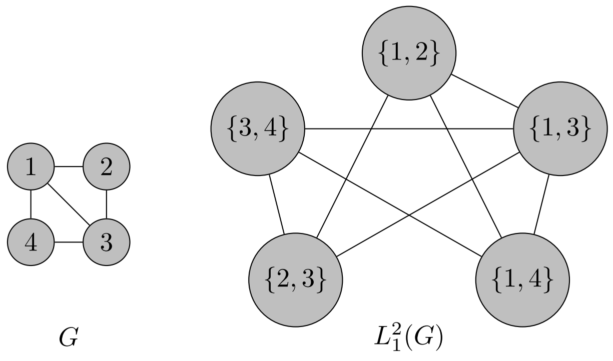


Figure 2.2: An example of a graph (left) and its line graph (right).

To conclude, for two vertices u, v, we define their multiplicity as the number of edges containing both u and v. Moreover, $m(H)$ denotes the maximum multiplicity among all pairs of vertices.

In chapter 4 we investigate the complexity of the reconstruction of graphs in L_k^l, focusing on $\mathcal{L}_3^2$.

2.1.2 Convexity on graphs

Convexity on graphs is a well-studied notion. We introduce standard definitions following [35]. Let $\mathcal{C} \subseteq \mathcal{P}(V)$ be a family of subsets of V. The pair $(V, \mathcal{C})$ is a *convexity space* if $\emptyset, V \in \mathcal{C}$, and $\mathcal{C}$ is closed under intersection.

The elements of $\mathcal{C}$ are the *convex sets*. For $U \subseteq V$, the *convex hull* $\langle U \rangle_{\mathcal{C}}$ is the smallest convex set containing U. The pair $(G, \mathcal{C})$ is a *graph convexity space* if $(V, \mathcal{C})$ is a convexity space and $G[U]$ is connected for any $U \in \mathcal{C}$. The most natural examples of graph convexities are *path convexities* which are defined by *interval functions* I :

$V \times V \mapsto \mathcal{P}(V)$. Interval functions extend to subsets $U \subseteq V$ as follows: $I(U) = \bigcup_{u,v \in U} I(u,v)$. For path convexities, the convex hull $\langle U \rangle_{\mathcal{C}}$ is given by $I(U)$. Set U is convex if $I(U) = U$, i.e., U is *closed* under the operator I. By definition $u, v \in I(u,v)$ and $I(v,v) = \{v\}$.

We consider convexities based on *shortest paths* and *chordless paths*. We define the following interval functions for vertices $u, v \in V$.

$$I_g^G(u,v) = \{w \in V \mid w \text{ is on a shortest path between } u \text{ and } v \text{ in } G\}.$$

The shortest paths between two vertices are called the *geodesics*. A set $U \subseteq V$ is *g-convex* if $U = I_g^G(U)$, and $\mathcal{C}_g := \{U \mid U = I_g^G(u)\}$ is the *geodesic convexity*. The following is the second interval function we consider.

$$I_m^G(u,v) = \{w \in V \mid w \text{ is on a chordless path between } u \text{ and } v \text{ in } G\}.$$

Chordless paths are induced paths and are called *monophonics*. A set $U \subseteq V$ is *m-convex* if $U = I_m^G(U)$, and $\mathcal{C}_m := \{U \mid U = I_m^G(U)\}$ is the *monophonic convexity*.

A *short chord* of a path P is a chord that connects two vertices with distance 2 along P , i.e., (u,v) is a short chord if (u,w) and (v,w) are path edges, for some vertex w. Note that u, v, w form a triangle. A *triangle path* is a path that admits only short chords and leads us to the final interval function.

$$I_t^G(u,v) = \{w \in V \mid w \text{ is on a triangle path between } u \text{ and } v \text{ in } G\}.$$

A set $U \subseteq V$ is *t-convex* if $U = I_t^G(U)$, and $\mathcal{C}_t := \{U \mid U = I_t^G(U)\}$ is the *triangle path convexity*.

If the graph G is clear from the context, we drop the super-script in the interval functions. Figure 2.3 illustrates the differences between the path convexities. Observe that $I_g(u,v) \subseteq I_m(u,v) \subseteq I_t(u,v)$, for any $u, v \in V$. Finally, we remark that the sets $I(u,v)$ (for each path convexity) can be computed in polynomial time.

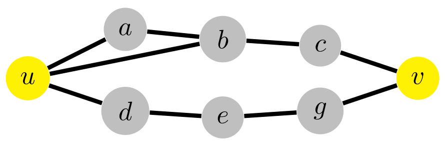


Figure 2.3: For vertices u and v, let $I_g(u,v)$ be the set of all vertices on a shortest path between them (analogously, we define $I_m(u,v)$ and $I_t(u,v)$). For the example graph, verify that $I_g(u,v) = \{u,b,c,v\}$, $I_m(u,v) = I_g(u,v) \cup \{d,e,g\}$, and that $I_t(u,v) = I_m(u,v) \cup \{a\}$.

We use the notion of convex graphs in Chapter 5, where we study a reconstruction problem on graphs, call Minimum Surgical Probing, that ask to find a convex subset of the vertex set.

2.2 Notions on polyominoes

Now we move to planar discrete sets of points, that constitute the second main part of the Thesis. A planar discrete set S is a finite set of points of the integer lattice $\mathbb{Z}^2$, considered up to translation. The dimension of S are those of its minimal bounding rectangle. The set S can be naturally represented as a set of unitary cells centred on the points of S (see Fig. 2.4, (a)). Note that we can indicate S equivalently as a binary matrices. In general, we indicate as $p = (x, y)$ the points of $\mathbb{Z}^2$ with integer coordinates (x, y).

A *row* (resp. *column*) of S is its intersection, when non-void, with an infinite strip of cells whose centres lie on horizontal (resp. vertical) lines.

To each discrete set of dimension $m \times n$ we can associate two integer vectors $H = (h_1, \ldots, h_m)$ and $V = (v_1, \ldots, v_n)$ using the following definition.

Definition 2. Given a finite lattice set $S \subseteq \mathbb{Z}^2$ included in the rectangle $[1, m] \times [1, n]$, its *vertical projections* is the vector $V \subseteq \mathbb{Z}^m$ whose $i-$th coordinate v_i is the number of points of S in the vertical line $x = i$. Similarly, its *horizontal projections* of S is the vector $H \in \mathbb{Z}^n$ whose $j-$th coordinate counts the point in the horizontal line $y = m - j + 1$.

For example, the horizontal projections of the discrete set depicted in Figure 2.4 are $H = (1, 2, 3, 4, 2, 1, 1)$ while its vertical projections are $V = (3, 3, 1, 6, 1)$.

Moreover, we define the *feet* of a finite lattice set $S \subseteq \mathbb{Z}^2$.

Definition 3. Given a finite lattice set $S \subseteq \mathbb{Z}^2$, suppose that $y_{min}(S)$ ($y_{max}(S)$) is the minimum ordinate (abscissa) of the points of S and a similar definition holds for $x_{min}(S)$ and $x_{max}(S)$. We define the South, East, North and West feet as the sets:

- $South(S) = S \cap \{y = y_{min}(S)\}$;
- $West(S) = S \cap \{x = x_{min}(S)\}$;
- $East(S) = S \cap \{x = x_{max}(S)\}$;
- $North(S) = S\{y = y_{max}(S)\}$.

A wide literature links the geometrical and topological characteristics of discrete sets with their projections [21, 22]. A special focus is on the properties of connectedness and convexity, providing several results on the related sub-classes.

So, we define a *polyomino* P a $4-$connected (i.e. connected along horizontal and vertical directions) planar discrete set. P is $h-$convex (resp. $v-$convex) if each rows (resp. columns) is connected (see Fig 2.4, (b)). If P is both h-convex and $v-$convex then we say that P is $hv-$convex. Moreover, we say that a binary matrix is $hv-$convex if its $1-$entries form a $hv-$convex polyomino.

A notion of convexity resembling that of continuous sets has been defined, taking care of pathological situations that may arise when continuous convex shapes are discretized into convex discrete sets [36, 37]. In the case of *polyominoes*, the notion of convexity turns in the natural simple form of the equivalence between the polyomino and its *convex hull* (see Fig 2.4, (c) and (d)).

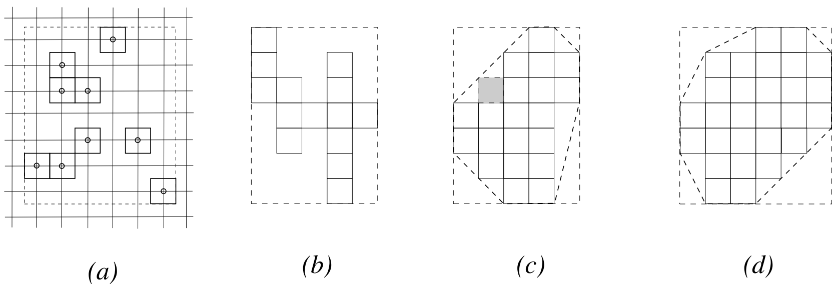


Figure 2.4: (a) a generic discrete set of points in $\mathbb{Z}^2$ and its representation as a set of cells on a squared surface; (b) a vertically convex polyomino; (c) a hv-convex polyomino. Its convex hull and the cell that prevents it to be full convex are dashed and highlighted; (d) a (full) convex polyomino and its convex hull.

2.2.1 Coding the boundary of a convex polyomino

We initially present some concepts of combinatorics on words. Let Σ denote a finite set of symbols. Its elements are called *characters* and Σ is called the *alphabet*. A *word* over Σ is an element of $\Sigma^* = \bigcup_{n \in \mathbb{N}} \{w_{i_1} \dots w_{i_n} | w_{i_j} \in \Sigma, 1 \leq j \leq n\}$. If $w = w_1 \dots w_n \in \Sigma^*$, we define its length $|w| = n$, i.e. the number of characters in it. Moreover, we denote with $|w|_a$ the number of occurrences of letter a in word w. The *mirror images* of w is the word $\tilde{w} = w_n \dots w_1$. If $w = \tilde{w}$, w is a *palindrome* word. The words w, w' are said to be *conjugate* if there exists u, v such that $w = uv$ and $w' = vu$. The *conjugacy class* of a word is defined as the set of all its conjugates and is equivalent to the set of all circular permutations of its letters.

Taking a step back to discrete geometry, it is common to represent a polyomino (without holes) through its *boundary word* [38], i.e., the word on four letter alphabet $W' = \{0, \bar{0}, 1, \bar{1}\}$ obtained by coding the path that clockwise follows the boundary of the cell representation of the polyomino starting from a specific point. We choose to associate the letters 0 (resp. 1) to the horizontal (resp. vertical) step, while $\bar{0}$ and $\bar{1}$ represent the horizontal step and the vertical step when travelling in opposite directions with respect to 0 and 1, respectively. Note that the *boundary word* depends on both the starting point and the orientation used to travel the border. However, they all belong to the same conjugacy class, previously defined. If the polyomino is hv-convex, we can identify four points W, N, E and S as the points where the polyomino's boundary first touches the west, north, east and south sides of its minimal bounding rectangle, respectively, when moving clockwise along it.

These four points determine four paths, according to the starting and ending points, i.e., WN, NE, ES and SW-paths, as depicted in Fig. 2.5. A path is WN-convex (resp. NE, ES and SW-convex) if it is the WN-path (resp. NE, ES and SW-path) of a convex polyomino. Each of the four paths is *monotone*, i.e., it uses only two of the four Freeman coding steps.

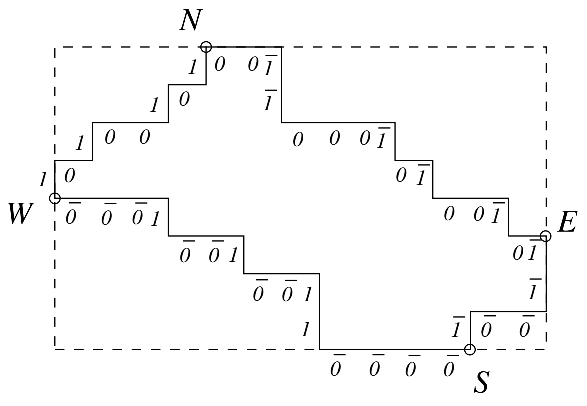


Figure 2.5: The boundary of an hv-convex polyomino and its decomposition into four monotone paths.

Note also that the paths divide the polyomino into 4 areas, which we denote with WN, NE, ES and SW.

2.2.2 Christoffel words

Recall that if $a, b \in \mathbb{N}$, then a and b are *co-prime* if 1 is the only positive integer that divides both a and b. We introduce the definition of *lower Christoffel path*.

Definition 4. Given two co-prime integer numebrs a, b, the *lower Christoffel path* of slope a/b is the (discrete) path from $(0, 0)$ to (b, a) in the integer lattice $\mathbb{Z}^2$ that satisfies the following condition:

- the path lies below the line segmente that begins at the origin and ends at (a, b);
- the region in the plane enclosed by the path and the line segment contains no other points of $\mathbb{Z}^2$ besides those of the path.

The *upper Christoffel path* is defined analogously, using paths in $\mathbb{Z}^2$ that lie above the line segment.

Figure 2.6 shows an example of an upper and lower Christoffel path. We also introduce the concept of *Lyndon word*.

Definition 5. A Lyndon word $l \in \Sigma^*$ is a word such that $l = uv$, with $u, v, \in \Sigma^*$ implies that $l < uv$, where $<$ is the standard lexicographical order.

For our purposes, Lyndon words are important due to the following result.

Theorem 1. *[39] Any word $w \in \Sigma^*$ admits a unique factorization as a sequence of decreasing Lyndon words $w = l_1^{n_1} \dots l_k^{n_k}$, $l_1 > l_2 > \dots > l_k$, where $n_i \geq 1$ and each l_i is a Lyndon word.*

To both Christoffel paths one can associate the so-called *Christoffel word* on the binary alphabet $\Sigma = \{0, 1\}$, such that the letter 0 represents a horizontal step and the letter 1 a vertical step. The Christoffel word commonly indicates the word w related to its lower Christoffel path and whose slope $\rho(w)$ equals $\frac{a}{b}$. Since the upper and the lower Christoffel paths of the same slope are mirror images of each other, the two related words $\widetilde{w}$ and w are mirror images of each other. By definition, $|w| = |\widetilde{w}| = a + b$. Figure 2.6 represents the Christoffel words associated with the two paths.

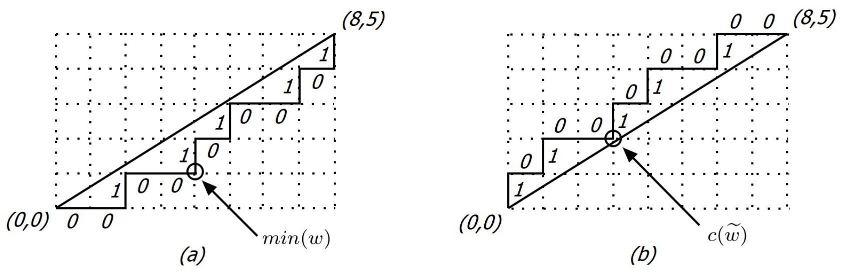


Figure 2.6: (a) the lower Christoffel path of the segment with slope $\frac{5}{8}$ and the related Christoffel word w; (b) the upper Christoffel path of the same segment and the word $\widetilde{w}$ associated to it. The points $min(w)$ and $c(\widetilde{w})$ are also highlighted

It is well known that each Christoffel word w different from 0 or 1 can be uniquely split as a concatenation of two smaller Christoffel words w_1 and w_2, providing the so-called *standard factorization* introduced in [40]. In fact, the following Theorem holds.

Theorem 2. *[40] A Christoffel word w has a unique factorization w = uv (indicated as standard factorization), where u and v are both Christoffel words.*

Such factorization involves the (unique) closest point $c(w)$ from the line segment of slope $\rho(w) = \frac{a}{b}$. So, it holds that $w = w_1 w_2$, where w_1 is the word leading from $(0, 0)$ to $c(w)$, and w_2 is the word leading from $c(w)$ to (a, b). By abuse of notation, we indicate with $c(w)$ also its index position in w. Moreover, we indicate as $min(w)$ the (index of the) furthest point from the same line (see Fig. 2.6). Note that $min(w)$ is also (the index of) the closest point in the upper Christoffel word $\widetilde{w}$, i.e., $min(w) = c(\widetilde{w})$.

It is worthwhile noticing the following property of Christoffel words.

Proposition 1. [36] A word w is $NW-$convex if and only if its unique Lyndon factorization $w = l_1^{n_1} \dots l_k^{n_k}$ is such that all l_i are primitive Christoffel words.

This result provides a simple and elegant way to determine if a polyomino is *convex*. In [37] it has been proved that such a decomposition is unique and it can be obtained by the Lyndon factorization of the WN-path. Moreover, the authors also provide an algorithm that uses these notions to compute the convex hull of a polyomino.

We use the notions of polyominoes and Christoffel words in Chapter 6, where we study their reconstruction starting only horizontal and vertical projections, and then considering some aggregate measure over certain types of windows.

Chapter 3

Null label problem

In this chapter we consider the null label problem on $3-$hypergraphs. In particular, we focus on the $2-$intersection graphs $L_3^2(H)$ of a $3-$hypergraph H and we prove that if $L_3^2(H)$ is Hamiltonian then H has a null label. We highlight that the results of this chapter have been published also in [41].

3.1 Problem definition and previous results

Consider a hypergraph $H = (V, E)$. We can assign a label l in the set $\{+1, -1\}$ to each edge, resulting in positive and negative ones. We can then define the following notions:

Positive signed degree $d_l^+(v) = |\{e \in E : v \in e \; and \; l(e) = +1\}|$, i.e. the number of positive edges that contain v;

Negative signed degree $d_l^-(v) = |\{e \in E : v \in e \; and \; l(e) = -1\}|$, i.e. the number of negative edges that contain v;

Signed degree $d_l(v) = d_l^+(v) - d_l^-(v)$.

An assignment of ± 1 to the edges of H result in a *null labelling* if $d(v) = 0$, for all vertices v and the relative hypergraph is said to be *null*. Note that an obvious necessary condition for a hypergraph to have a null labelling is that each vertex must have even degree, i.e., it is an *even hypergraph*. In this chapter we consider the following problem.

3-Hypergraph Null Labelling Problem (3-NLP): Let H be a connected, even 3-hypergraph. Does there exist an assignment of ± 1 to the edges of H that produces a null label?

Interestingly, we can consider $3-NLP$ also from a tomographical perspective. Consider two hypergraphs H_1 and H_2 with the same degree sequence and assign $+1$ to each hyperedge of H_1 and -1 to the edges of H_2. It's easy to see that the symmetric difference

$H_1 \ominus H_2$ produces a null hypergraph. Vice versa, given a null hypergraph H, we can obtain two hypergraphs with the same degree sequence following this procedure: call H_1 the hypergraph with the same vertices as H but containing only the positive edges of H. Similarly, H_2 contains only the negative edges of H. As said, it's easy to see that H_1 and H_2 have the same degree sequence.

Null hypergraphs have been firstly used in [23] to study a set of operations that allow one to move through all the 3-hypergraphs with the same degree sequence. Moreover, in a follow-up paper [42], the following conjecture has been proposed.

Conjecture 1. All connected, even $3-$hypergraphs on n vertices with more than $\frac{1}{2}\binom{n}{3}$ edges have null labelling.

In the appendix we provide some *Matlab* code that generates $3-$hypergraphs and checks if they are null. So far, the empirical results obtained through computations support the previous statement.

In fact, in the case of $n \leq 7$ and even hypergraphs, the exhaustive computations show that:

$i)$ if $|E| \geq \frac{1}{2}\binom{n}{3}$ then all the hypergraphs admit null labelling;

$ii)$ if $|E| < \frac{1}{2}\binom{n}{3}$ then there exist hypergraphs both admitting and non-admitting null labelling.

However, no formal proof is present at the moment, even if some results (also the one presented in this chapter) seem to suggest its validity. Interestingly, the conjecture also implies that dual hypergraphs (i.e. that have complementary edges set) do not preserve the *null* property.

For example, consider the following $3-$hypergraphs, whose edges are arranged as a matrix.

$$H = \begin{bmatrix} 1 & 5 & 6 \\ 3 & 4 & 6 \\ 2 & 4 & 5 \\ 1 & 2 & 3 \end{bmatrix}$$

It is easy to see that H has no null label. However, its dual is a null $3-$hypergraph. The interested reader can utilize the *Matlab* code provided in the appendix to check the previous statement.

In [42], the following lemma is proved.

Lemma 1. *[42] A graph G is null if and only if every connected component is an Eulerian graph with an even number of edges.*

In particular a consequence of the result is that, when considering a graph G, the Null Label problem simply reduces to search for an Eulerian Tour in G or a Hamiltonian cycle in its Line Graph $L_2^1(G)$. This lemma also characterizes the graphs with even degrees and an even number of edges that do not have null labelling: they must be disconnected graphs such that at least two connected components have an odd number of edges.

However, moving to $3-$hypergraphs, the situation becomes more complex. In particular, in [42] it is shown that the problem of finding null labelling even for the simplest case of 3-hypergraphs is NP-complete.

Therefore, we are interested in finding suitable subclasses of $3-$hypergraphs that are null. Note that this problem also connects to the previous Conjecture 1.

The first results in this direction used the notion of *intersection graph* of a $3-$hypergraph. Formally it is defined as $L_3^1(H)$ (see Chapter 2). Importantly, this is a first extension of the idea of line graphs to 3-hypergraphs. In [42] the following result was proved.

Theorem 3. *[42] Let H be a connected, even 3-hypergraph, in which every vertex has degree two. Then H has a null labelling if and only if $L_3^1(H)$ is bipartite.*

However, the inspection of $L_3^1(H)$ does not provide useful hints, in general, to the existence of null labelling in the related $3-$hypergraph as shown in the following example.

Example 1. Consider the following 3-hypergraphs H_1 and H_2 on six vertices whose edges, arranged in matrix form, are:

$$H_1 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 5 \\ 2 & 4 & 6 \\ 3 & 5 & 6 \end{bmatrix} \qquad H_2 = \begin{bmatrix} 1 & 2 & 5 \\ 2 & 3 & 5 \\ 2 & 3 & 4 \\ 1 & 2 & 4 \end{bmatrix}$$

It is easy to check that the vector of labels $l = (1, -1, 1, -1)$, where $l(i)$ is the label of the i-th edge of the 3-hypergraph or, equivalently, of the i-th row in the matrix arrangement of its edges, is a null label for H_2. On the other hand, an easy check reveals that H_1 has no null labelling. However, H_1 and H_2 have the same intersection graph K_4, i.e. the complete graph on four vertices.

3.2 The 2-intersection graph and its connection to the null label problem

In the previous section, we recalled some simple results about null labelling, highlighting that $L_3^1(H)$ does not provide enough information about the null label of hypergraphs in general. Relying on this fact, we decide to use graphs in $\mathcal{L}_3^2$. In particular, for the sake of

simplicity, we call these graphs $2-$*intersection graphs* of 3-hypergraphs. Interestingly, they reveal to be a valuable tool for a deeper investigation of **3-NLP**.
Example 2 and the related Fig. 3.1 shows the $2-$intersection graph of a $3-$hypergraph. Note also that the hypergraphs considered in Example 1 have different $L_3^2(H)$, contrary to the case of $L_3^1(H)$. We are interested in studying some properties of the 2-intersection graph that are relevant to obtain information about the existence of null labelling in the related 3-hypergraph. In particular, we prove that the existence of a Hamiltonian cycle in $L_3^2(H)$ is sufficient for a $3-$hypergraph to be null. This idea comes from the following observation: we know that a connected, even graph G is Eulerian and an Euler tour in G corresponds to a Hamiltonian cycle in its line graph $L_2^1(G)$ (and viceversa). From Lemma 1 we know that it also corresponds to a null labelling of G. Therefore, Hamiltonian cycles in $L_2^1(G)$ can be used to determine null labellings of G.

Since the notion of $2-$intersection graph is similar to that of the line graph of a graph, we suppose that also similar properties hold. It is well known (see [43]) that a line graph uniquely decomposes into maximal cliques, with at most a vertex in common. A similar result holds also in $L_3^2(H)$, as shown in the next chapters.

Example 2. Consider the following hypergraph $H = (V, E)$ in which $V = \{1, \dots, 6\}$ and $E = \{\{3, 4, 5\}, \{3, 5, 6\}, \{1, 3, 5\}, \{3, 4, 6\}, \{2, 4, 6\}, \{4, 5, 6\}, \{1, 2, 6\}, \{2, 3, 5\}, \{1, 2, 5\}, \{1, 2, 4\}, \{1, 2, 3\}, \{1, 4, 6\}\}$.

The 2-intersection graph of H is shown in Figure 3.1. One can check that the edges with the same label form a clique.

A first naive strategy to produce a null labelling of a $3-$hypergraph starting from a Hamiltonian cycle C consists of alternately labelling ± 1 the vertices of C to obtain an initial label of H. Note that this strategy leads to a null label for line graphs of even graphs.

However, the alternating labelling of C does not always provide a null labelling of H, as witnessed by the following example. Note that we indicate with v_{e_i} the node relative to edge e_i.

Example 3. Consider the 3-hypergraph $H = (V, E)$ on six vertices and $E = \{e_1, \dots, e_8\}$, where $e_1 = \{1, 2, 3\}, e_2 = \{1, 2, 4\}, e_3 = \{1, 2, 5\}, e_4 = \{1, 2, 6\}, e_5 = \{1, 3, 4\}, e_6 = \{1, 3, 5\}, e_7 = \{2, 3, 5\}, e_8 = \{2, 5, 6\}$.

The related 2-intersection graph $L_3^2(H)$ in Fig. 3.2 has the Hamiltonian cycle $C_1 = (v_{e_1}, v_{e_3}, v_{e_2}, v_{e_4}, v_{e_8}, v_{e_7}, v_{e_6}, v_{e_5}, v_{e_1})$

It is easy to check that alternately labelling ± 1 the vertices of C_1, starting with $+1$, we obtain the null labelling $l_1 = (1, 1, -1, -1, -1, 1, -1, 1)$ on the eight edges of H such that $l_1(i)$ is the label of e_i, with $1 \leq i \leq 8$.

However, a second Hamiltonian cycle $C_2 = (v_{e_1}, v_{e_2}, v_{e_3}, v_{e_4}, v_{e_8}, v_{e_7}, v_{e_6}, v_{e_5}, v_{e_1})$ exists such that the alternating labelling $l_2 = (1, -1, 1, -1, -1, 1, -1, 1)$ is not null on H, as $d(v_4) = -2$ and $d(v_5) = +2$.

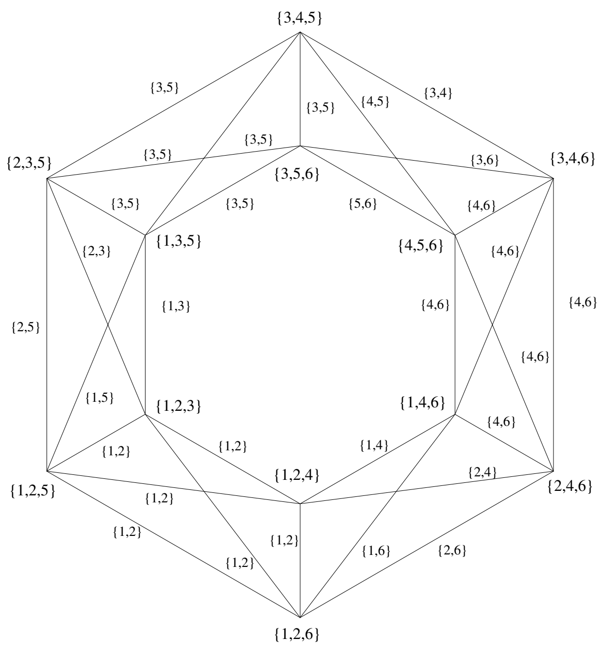


Figure 3.1: The 2-intersection graph of the 3-hypergraph H in Example 2. The edges are labelled according to the pairs shared by their vertices.

Let us introduce some notation: suppose that v_{e_i} and v_{e_j} are two consecutive vertices of the Hamiltonian cycle C of $L_3^2(H)$, with $e_i = \{u, x, y\}$ and $e_j = \{v, x, y\}$. We say, by extension, that v_{e_j} belongs to node $v \in H$. We see that $v \notin e_i$. There may be several consecutive vertices of C that contain v. Denote by $p_v = (v_{e_{j_1}}, \dots, v_{e_{j_k}})$ the longest sub-path of C starting in v_{e_j} such that every vertex of p_v contains v. Let $l(p_v)$ denote the labels of the vertices of p_v, $\sigma(l(p_v))$ denote the sum of the elements of $l(p_v)$ and let $|p_v|$ denote the length $k-1$ of p_v, i.e., its number of edges. If p_v contains just a single vertex, we set $|p_v| = 0$.

In the case presented above, $e_i = \{u, x, y\}$ is clearly the last vertex in the path p_u. We define *next*$(p_u) = p_v$, i.e., given a path p_u, *next*(p_u) is the path beginning at the first vertex following the last vertex of p_u. In general, C may contain several different sub-paths of the form p_v, for each vertex v; we indicate them by $p_v^1, \dots, p_v^n$. The following example clarifies the definitions.

Example 4. Consider the $2-$intersection graph depicted in Figure 3.3.

Consider the path $p_5 = \{\{1,3,5\}, \{1,2,5\}, \{2,3,5\}\}$. We have $l(p_5) = \{-, +, -\}$ and therefore $\sigma(l(p_5)) = -$. Moreover, $|p_5| = 3$ and $next(p_5) = p_6$.

Finally, note that some distinct subpaths are associated with the same node (e.g. nodes 3 and 5).

Using the subpath notation we are able to characterize null labels from a different

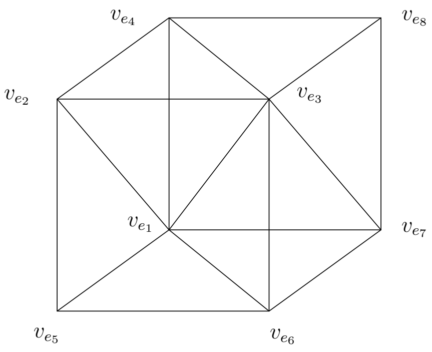


Figure 3.2: The 2-intersection graph of the 3-hypergraph considered in Example 3.

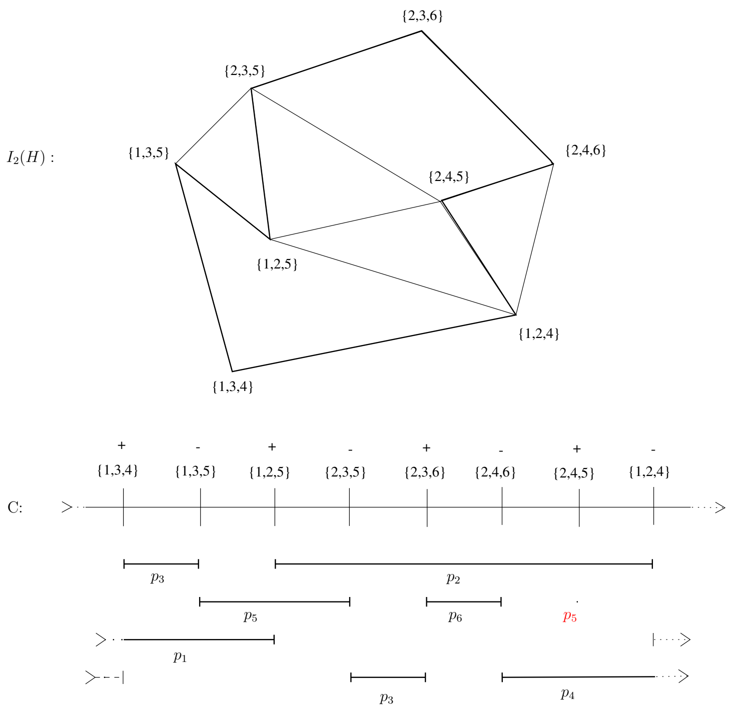


Figure 3.3: A 2-intersection graph $L_3^2(H)$ and one of its Hamiltonian cycles C are shown. The sub-paths p_v related to the vertices of H are highlighted. Note that in each vertex of $L_3^2(H)$ starts (resp. ends) a sub-path related to two, non-necessarily distinct, vertices of H. For example, a path p_5 that reduces to the single vertex, i.e. starting and ending in $\{2,4,5\}$ is highlighted in red.

perspective.

Property 1. Consider an alternating labelling ± 1 on the vertices of a Hamiltonian cycle

C of $L_3^2(H)$. For each sub-path $p_v = (v_{e_{j_1}}, \ldots, v_{e_{j_k}})$, the following holds:

- if p_v has odd length, then $\sigma(l(p_v)) = 0$, so the labels of the edges $e_{j_1}, \ldots, e_{j_k}$ containing v sum to zero in H. In this case, the first and the last vertex of p_v have different labels;

- if p_v has even length, then $\sigma(l(p_v)) \neq 0$ and the sum of the labels of the edges $e_{j_1}, \ldots, e_{j_k}$ containing v contribute $+1$ or -1 to the signed degree of v. In this case, the extremal vertices of p_v have the same label.

The proof of this property is straightforward. Figure 3.3 shows the 2-intersection graph of a 3-hypergraph on six vertices and eight edges. One of its Hamiltonian cycles and the sub-paths related to the vertices of H are highlighted. Note that the paths in the figure circle around the right edge of the diagram back to the left edge.

Let us continue analyzing the properties of the alternating labelling $l(C)$ of a Hamiltonian cycle C of $L_3^2(H)$. Property 1 assures that, if the labelling $l(C)$ produces a signed degree $d_l(v) = d \neq 0$ for vertex v, then there exists at least $p_v^1, \ldots, p_v^{|d|}$ subpaths with the same sum of labels (i.e. $+1$ or -1 depending on the sign of d).

We define the *distance* $d(p_u, p_v)$ between two paths p_u and p_v as the distance along C between the last point of p_u and the first point p_v. Note that any two of the previous $|d|$ subpaths have even distance. The above observations lead to the following lemmas.

Lemma 2. *Let H be an even 3-hypergraph and $L_3^2(H)$ its 2-intersection graph. If $L_3^2(H)$ has a Hamiltonian cycle C, an alternating ± 1 labelling $l(C)$ defines a null label of H if and only if, for each $v \in V$:*

$i)$ *each subpath p_v has odd length;*

OR

$ii)$ *the number of subpaths of v having even length is even and the sum of their labels is zero.*

The proof is straightforward. We emphasize that $ii)$ expresses the condition that, for each vertex v of H, there is the same number of subpaths of p_v having label $+1$ as -1.

We also have the following lemma.

Lemma 3. *Let H be a 3-hypergraph and C a Hamiltonian cycle of $L_3^2(H)$, and let $v_e = \{u, x, y\}$ be a vertex of $L_3^2(H)$. There are exactly three subpaths containing v_e, namely p_u, p_x and p_y. One of them begins at v_e and one of them ends at v_e (possibly the two paths may collapse into a single one having a vertex only).*

Proof. Let us assume w.l.o.g. that the next vertex of C intersects v_e in two vertices, say x and y. Suppose w.l.o.g. that $e' = \{v, x, y\}$ is the next vertex of C. The previous vertex, say e'', also intersects v_e in two vertices, wlog, either x, y or u, x. Then either $e'' = \{w, x, y\}$ or $e'' = \{u, x, w\}$, for some w. In the first case, we have p_u begins and ends at v_e. In the second case, we have p_u ends at v_e and p_y begins. $\square$

In the sequel, we describe an algorithm that modifies an alternating ± 1 labelling of a Hamiltonian cycle C not satisfying the conditions of Lemma 2 in order to obtain a null labelling of H. This algorithm relies on the *Switch* operator defined as follows.

Definition 6. Given two sub-paths $p_u = (v_{e_{i_1}}, \ldots, v_{e_{i_k}})$ and $p_v = (v_{e_{j_1}}, \ldots, v_{e_{j_{k'}}})$, where $p_v = \textit{next}(p_u)$, and $e_{i_k} \neq e_{j_1}$, the operator *Switch*(p_u, p_v) produces a new labelling $l'(C)$ by changing the signs of e_{i_k} and e_{j_1}: $l'(e_{i_k}) = -l(e_{i_k})$ and $l'(e_{j_1}) = -l(e_{j_1})$; at the same time, it keeps the remaining labels of $l(C)$ unchanged.

Figure 3.4 shows an example of the action of *Switch*(p_2, p_5).

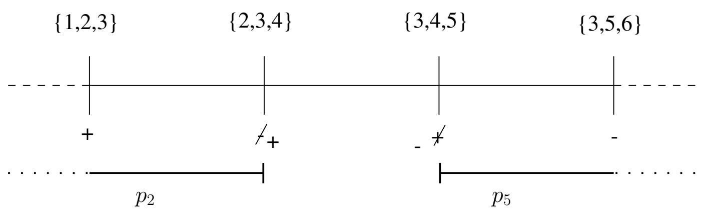


Figure 3.4: Example of *Switch*(p_2, p_5) between the two consecutive paths p_2 and p_5, i.e., such that p_5 =*next*(p_2). The switch operator changes $l(\{2, 3, 4\})$ from $-$ to $+$ and $l(\{3, 4, 5\})$ from $+$ to $-$.

We will start with an alternating labelling $l(C)$, and gradually change it using *Switch*() to produce a null labelling of H.

Property 2. Let H be a 3-hypergraph, C a Hamiltonian cycle of $L_3^2(H)$, and l a ± 1 labelling of C. Consider a sub-path p_u of C whose last element v_{e_i} with label $+1$, and the sub-path $p_v = \textit{next}(p_u)$ whose first element v_{e_j} with label -1. The operator *Switch*(p_u, p_v) updates l into l' so that $d_{l'}(u) = d_l(u) - 2$, $d_{l'}(v) = d_l(v) + 2$ and all the remaining signed degrees are left unchanged.

Proof. W.l.o.g., assume that $e_i = \{u, x, y\}$ and $e_j = \{v, x, y\}$. It is immediate to realize that the change of the opposite labels of e_i and e_j preserves the signed degrees of x and y, while it subtracts 2 from u and adds 2 to v. As the starting labels of e_i and e_j are opposite, a symmetric result holds. $\square$

The algorithm $Balance()$ defined below modifies a labelling $l(C)$ of a Hamiltonian cycle C of $L_3^2(H)$ in order to change, after a sequence of successive applications of the *Switch*() operator, the signed degree of two input vertices u and v of H, if possible, otherwise it gives failure.

Algorithm 1 Balance$(u, v, l(C))$

```
Input: the label $l(C)$ of a Hamiltonian cycle $C$, and two vertices $u$ and $v$ of $H$ with
signed degrees $d_l(u) > 0$ and $d_l(v) < 0$.
Output: The label $l'(C)$ such that $d_{l'}(u) = d_l(u) - 2$ and $d_{l'}(v) = d_l(v) + 2$.

$p_i = p_u$ such that $|p_u|$ is even and $\sigma(l(p_u)) = +1$
while true do
  $p_j = next(p_i)$
  $Switch(p_i, p_j)$
  if $|p_j|$ is odd then
    $p_i = p_j$;
  else if a non already considered $p'_j$ exists such that $|p'_j|$ is even and $p'_j$ starts with $+1$
  label then
    $p_i = p'_j$
  else
    FAILURE;
  end if
end while
return the final $l'(C)$ as OUTPUT.
```

First, we prove that the algorithm $Balance()$ computes a null labelling starting from the alternating labelling $l(C)$ in the easiest case of having only two signed degrees u and v different from zero, in particular $+2$ and -2, respectively.

Lemma 4. *Let H be a 3-hypergraph, C a Hamiltonian cycle of $L_3^2(H)$, and l an alternating labelling of C. If u and v are the only nodes of H with signed degree different from zero, in particular $d_l(u) = +2$ and $d_l(v) = -2$, then $Balance(u, v, l(C))$ returns a null labelling $l'(C)$ of H.*

Proof. Since $d_l(u) = 2$, there exists at least one subpath p_u such that $|p_u|$ is even and $\sigma(l(p_u)) = +1$, i.e. it starts and ends with two elements labelled with $+1$.

The $While$ cycle in $Balance(u, v, l(C))$ starts by performing the switch between p_u and $p_j = next(p_u)$ and l is updated to l'. Since l is an alternating labelling, the first element of p_j has label -1, so after *Switch*(p_u, p_j), we have $d_{l'}(u) = 0$, $d_{l'}(j) = d_l(j)+2$, and by Property 2, the other signed degrees do not change. Now, if $|p_j|$ is even and $j = v$, then $d_{l'}(p_v) = 0$ and l' is the desired null labelling.

On the other hand, if $|p_j|$ is odd, then it ends with a $+1$ label and its successor *next*(p_j) starts with a -1. So, after updating $p_i = p_j$ and $p_j =$ *next*(p_i), *Switch*(p_i, p_j) is again performed and the new l' changes back the signed degree of i to zero, while $+2$ is added to the signed degree of the new j. Note that in this case, even if $j = v$, the algorithm continues applying $Switch$ until reaching a $|p_v|$ of even length and such that $\sigma(l'(p_v)) = -1$.

If the last case $|p_j|$ is even and $j \neq v$ occurs, then we move to another new even p'_j such that $\sigma(l(p'_j)) = +1$. Such a p'_j always exists since $d_l(j) = 0$, and consequently the number of even sub-paths containing j whose labels sum up to $+1$ equals those whose labels sum up to -1. So, FAILURE never occurs starting from an alternating labelling l.

Finally, the result is obtained by observing that $Balance(u, v, l(C))$ does not loop since the number of subpaths of C is finite and each of them is involved in the $while$ cycle at most once since the procedure always switches a sub-path that ends with a $+1$ with a sub-path that ends with -1. □

Lemma 5. *Let $H = (V, E)$ be a 3-hypergraph, C a Hamiltonian cycle of $L^2_3(H)$ and l an alternating labelling of C. If v_1 and v_2 are the only nodes of H with signed degree different from zero with respect to l, say $d_l(u) = +2k$ and $d_l(v) = -2k$, where $k \geq 1$, then H admits a null labelling.*

Proof. This is obtained by k successive runs of $Balance(u, v, l^i(C))$, with $0 \leq i < k$, where $l^{i+1}(C)$ is the labelling obtained as output of $Balance(u, v, l^i(C))$. We set $l^0 = l$; the output l^k of $Balance(u, v, l^{k-1}(C))$ provides a null labelling of H.

We emphasize that from the second run of $Balance()$ until the last one, i.e., the k-th run, the choice of a new starting sub-path p_u is always possible. In fact, from $d_l(u) = +2k$ it follows that in $l(C)$ the number of sub-paths of u whose labels sum up to $+1$ exceeds exactly by k those whose labels sum up to -1. A last remark is required: since two different runs of $Balance()$ start from different sub-paths p_u, their computations do not involve the same sub-path twice. So, each call of *Switch*() in the k runs of $Balance()$ always modifies two elements of C whose labels are alternate, as set by $l(C)$.

□

This same reasoning can be generalized when more than two vertices of H have non-null signed degrees leading to our main result

Theorem 4. *Let H be a 3-hypergraph. If the 2-intersection graph $L^2_3(H)$ is Hamiltonian, then H admits a null labelling.*

The proof of this theorem and the related computation of the null label of H directly follow from the proofs of Lemmas 4 and 5, after observing that we can iterate the calls of $Balance(u, v, l(C))$ varying u among all the vertices with signed degree greater than zero until reaching the first vertex v among those having signed degree less than zero.

Note that this result provides evidence of Conjecture 1. In fact, it is easy to see that the $2-$intersection graph of $3-$hypergraphs having a large number of edges tend to be complete graphs. In this case, we can obviously apply Theorem 4. Therefore, from a certain threshold, it is true that all the $3-$hypergraphs are null. Further studies could provide a closed form to this threshold and study its asymptotic properties. To help the reader in the visualization of this property, we provide some *Matlab* code in the appendix to compute and show the $2-$intersection graph of a given $3-$hypergraph.

We conclude with an example that shows the computation of a null labelling with a sequence of applications of $Balance$, in accord with Theorem 4.

Example 5. Consider the following $3-$hypergraph $H = (V, E)$ with $V = \{1, \dots, 8\}$ and E = {{2,3,5}, {2,5,8}, {2,4,8}, {1,4,8}, {1,4,7}, {1,6,7}, {1,4,6}, {1,5,6}, {5,6,7}, {1,5,7}, {1,2,7}, {1,2,3}, {2,3,6}, {3,6,8}, {3,7,8}, {3,5,8}}

Figure 3.5 shows a Hamiltonian cycle C of $L_3^2(H)$, and one of its alternating labellings $l(C)$.

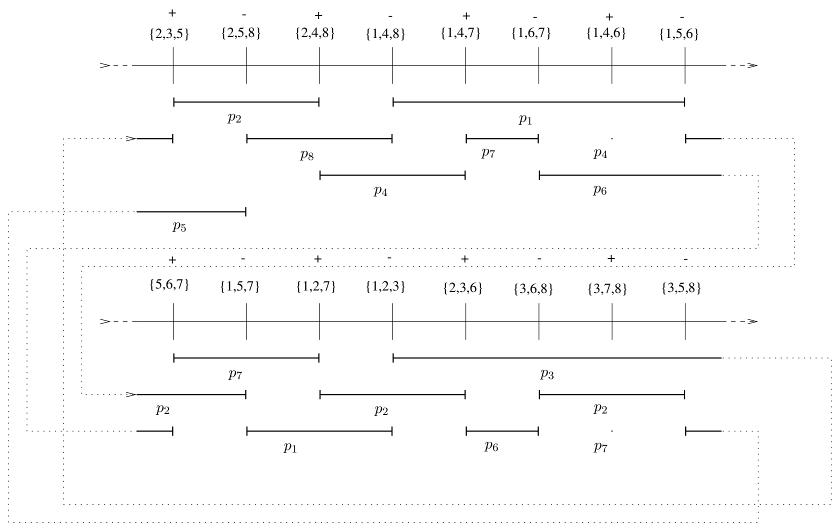


Figure 3.5: A Hamiltonian cycle of $L_3^2(H)$ and its labelling.

The chosen labelling is not a null labelling of H. The vector of the signed degrees of the vertices of H is

$$d = (-2, 2, 0, 2, -2, 0, 2, -2).$$

Let us perform a sequence of runs of $Balance()$ to compute a null labelling of H starting from $l(C)$.

Let us start, as an example, the run $Balance(2, v, l(C))$ in the p_2 sub-path having $\{2, 3, 5\}$ as first element. It is performed the call *Switch*(p_2, p_1), with $p_1 =$ *next*(p_2) and $|p_1|$ even. Since $d_l(1) = -2$, we perform the choice $v = 1$, and the switchings of $\{2, 4, 8\}$ and $\{1, 4, 8\}$ leading to the labelling $l^1(C)$ such that $d_{l^1}(1) = d_{l^1}(2) = 0$, leaving the remaining labels unchanged.

Let us now arbitrarily choose the vertex 7 such that $d_{l^1}(7) = +2$ and run $Balance(7, v, l^1(C))$ with the starting p_7 sub-path whose first element is $\{5, 6, 7\}$. The sub-path $p_3 =$ *next*(p_7) has odd length so the labels of $\{1, 2, 7\}$ and $\{1, 2, 3\}$ are switched and we obtain $d(7) = 0$ and $d(3) = +2$. Now $p_8 =$ *next*(p_3) and the labels of $\{2, 3, 5\}$ and $\{2, 5, 8\}$ are switched obtaining $d(3) = d(8) = 0$. Since $|p_8|$ is even, the run $Balance(7, v, l^1(C))$ ends setting $v = 8$. A new labelling $l^2(C)$ is returned as output.

Two more vertices with signed degrees different from zero are left, i.e., the vertices 4 and 5. A last run of $Balance(4, 5, l^2(C))$ is performed. Taking the p_4 subpath containing only $\{1, 4, 6\}$, we have $p_5 =$ *next*(p_4) with $|p_5|$ even. Therefore, switching the sign of $\{1, 4, 6\}$ and $\{1, 5, 6\}$ we obtain a new labelling l^3 such that $d_{l^3}(4) = d_{l^3}(5) = 0$ and $Balance(4, 5, l^2(C))$ ends. Therefore, the labelling

$$l^3 = (-1, 1, -1, 1, 1, -1, -1, 1, 1, -1, -1, 1, 1, -1, 1, -1)$$

is a null labelling of H. Note that the order of the calls of $Balance()$ is not relevant in order to obtain a null labelling of H.

Some further results can be obtained. In particular, note that provided an even graph G with n connected components and such that each component has an even number of edges, we can find a null labelling of G by considering, for each component, an Eulerian cycle and assigning alternatively ± 1 to its edges. This means that the line graph $L(G)$ has n components, each having a Hamiltonian cycle, where we can alternately assign ± 1 to the vertices of each Hamiltonian cycle to obtain a null labelling in the same fashion. It is easy to realize that a similar result holds also for $3-$hypergraphs.

Proposition 2. Let H be a 3-hypergraph whose 2-intersection graph $G = L_3^2$ has n connected components $G_1, \ldots, G_n$. If each connected component G_i, with $1 \leq i \leq n$, is Hamiltonian, and the sub-3$-$hypergraph $H_i \subset H$ such that $G_i = L_3^2(H_i)$ is even, then H admits a null labelling.

Proof. Since each G_i is Hamiltonian and it is the 2-intersection graph of an even sub-3-hypergraph H_i, then we can apply Theorem 4 to it, obtaining a null labelling for H_i. The null labelling of H is obtained by the union of the null labellings of these sub-hypergraphs. □

Note that H may also be a connected $3-$hypergraph, but the requirement of the previous theorem asks that $L_3^2(H)$ has to be disconnected. Furthermore, each of its components comes from an even sub$-3-$hypergraph of H.

To conclude this chapter, we also provide a simple generalization of Theorem 4 to $k-$hypergraphs. With similar arguments, it is possible to prove the following theorem.

Theorem 5. *Let $H = (V, E)$ be a $k-$hypergraph such that its $L_k^{k-1}(H)$ is Hamiltonian. Then, H admits a null labelling.*

Chapter 4

Reconstruction of the 2-intersection graphs

Due to the connections with the null labelling problem, our main focus in this chapter is to continue the study of the structural properties of graphs in L_3^2. In particular, we prove that given a graph $G = (V, E)$ it is NP-complete to decide the existence of a $3-$hypergraph such that $G = L_3^2(H)$. On the other hand, we prove the polynomiality of the reconstruction when restricting to triangulated claw-free graphs, while the $NP-$completeness is preserved when considering general claw-free graphs. We highlight that part of the results of this chapter have been published in [44].

4.1 Properties of graphs in $\mathcal{L}_3^2$

In this section we provide some structural properties of 2-intersection graphs that are related to 3-hypergraphs, with the aim of characterizing them and determining the computational complexity of their reconstruction starting from a graph G. If $G \in \mathcal{L}_3^2$ we say that G has the $2-$intersection property or it is reconstructible. In particular, for $G = L_k^l(H)$ we define a λ_k^l-labelling as a labelling of its vertices such that the label of a vertex $e \in V(G)$ is a set of k elements, representing the nodes $v \in V(H)$ contained in e.

Maximal cliques are relevant substructures when we deal with hypergraph reconstruction issues.

Property 3. The edges of a 3-hypergraph H sharing the same pair of vertices determine a clique in the 2-intersection graph $L_3^2(H)$.

However, we observe that Property 3 does not characterize the cliques of $L_3^2(H)$. In fact, there may exist K_3 subgraphs in it whose edges do not share a common label.

Let T be a K_3 clique in $G = L_3^2(H)$ with vertices a, b, c. Two different labels are allowed: either $a = \{1, 2, x\}, b = \{1, 2, y\}, c = \{1, 2, z\}$, with $x \neq y \neq z$ or $a = \{1, 2, 3\}, b = \{1, 2, 4\}, c = \{1, 3, 4\}$. The first case is indicated as $K-$triangle and the second as $T-$triangle.

Figure 4.1 (a), shows a configuration of a 2-intersection graph where a T-triangle and a K-triangle live together and share an edge. The T-triangle is shaded in grey.

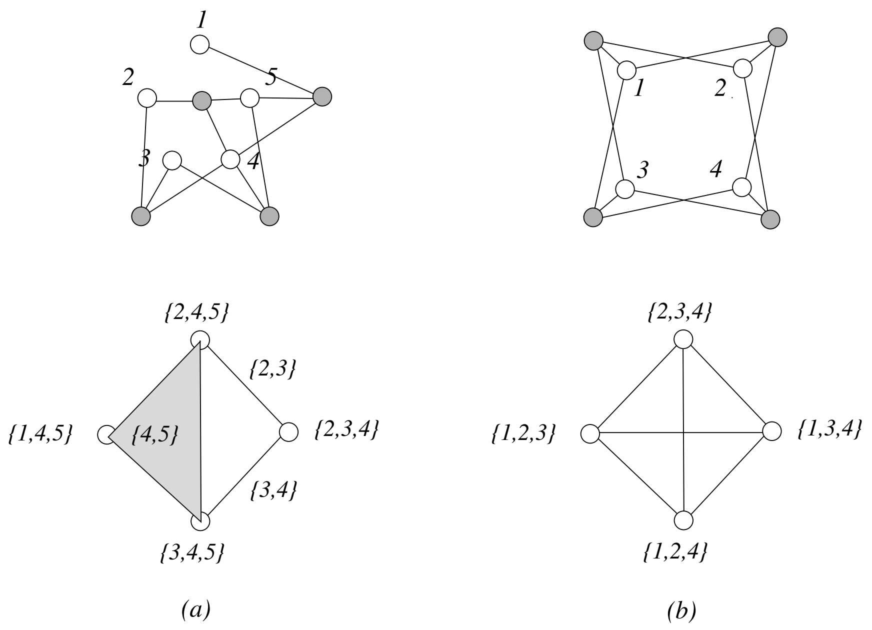


Figure 4.1: (a) example of a T and a K sharing 2 vertices. (b) example of S.

A similar situation occurs with K_4 cliques, say *square*, in G. In fact, if we consider a clique with four vertices a, b, c, d, then again two cases appear: either $a = \{1, 2, x\}, b = \{1, 2, y\}, c = \{1, 2, z\}, d = \{1, 2, t\}, x \neq y \neq z \neq t$ or $a = \{1, 2, 3\}, b = \{1, 2, 4\}, c = \{1, 3, 4\}, d = \{2, 3, 4\}$. Similarly, we denote the first case as $K-$square and the second as $S-$square. Figure 4.1 (b) shows the latter.

No similar ambiguities arise for bigger cliques. Let $K_p, p \geq 5$, the clique with p vertices $a_1, \ldots, a_p$. Then it is trivial that, w.l.o.g. $a_i = \{1, 2, x_i\}, 1 \leq i \leq p$ must hold. Any other configuration in which a couple is not shared by all the nodes will not lead to a K_p. In the sequel we will refer to *positive cliques* whenever their labels share a couple of vertices, *negative* otherwise. Therefore, all the cliques bigger than 4 are positive.

To set the notation, when a clique K is positive (resp. negative) we denote it by K^+ (resp. K^-). For convenience the clique of two vertices K_2 is both positive and negative.

Proposition 3. If K_i, K_j are two cliques such that $|K_i \cap K_j| = 2$ then we have K_i^+ and K_j^- (or vice versa).

Proof. W.l.g. let us consider $K_i \cap K_j = \{u, v\}$ with $u = \{1, 2, 3\}, v = \{1, 2, 4\}$ and $s \in K_i \setminus K_j, t \in K_j \setminus K_i$ be such that $st \notin E$. By contradiction, we assume that K_i^+, K_j^+ or K_i^-, K_j^-. Two cases arises: if K_i^+, K_j^+, then it holds $s = \{1, 2, 5\}, t = \{1, 2, 6\}$ that is not possible. Lastly, if K_i^-, K_j^-, then w.l.g., it holds $s = \{1, 3, 4\}$, again a contradiction when labelling t. □

A simple consequence of the previous proposition is the following.

Corollary 1. *Let e be an edge of $G \in \mathcal{L}_3^2$. Then e is shared by two cliques at most.*

With the previous definitions, the first obvious necessary condition for a graph to be reconstructible in L_3^2 is described in the following lemma.

Lemma 6. *If $G \in L_3^2$, then every vertex of G must belong to at most three positive maximal cliques. Furthermore, these maximal cliques can be joined by at most two non-adjacent T-triangles as shown in Figure 4.2.*

Proof. A clique in $L_3^2(H)$ can be obtained from hyperedges sharing the same couple of vertices or it can be a triangle (in which two pairs are used). In all cases, since a vertex of $L_3^2(H)$ has only three pairs, it cannot belong to more than three cliques. However, note that the configuration in which a point is shared by four triangles is reconstructible (see Figure 4.2).

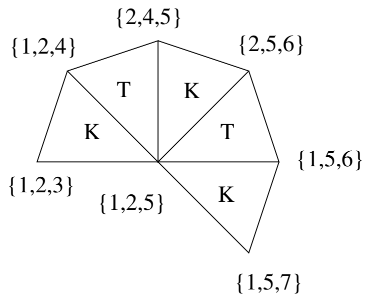


Figure 4.2: Reconstruction of the configuration in which a point belongs to 5 triangles.

The reason is simple: the edges of the T triangles shared with the K triangles count as a unique pair containing the central point. Therefore, it shares exactly three pairs, as desired. Note that the configuration is reconstructible even if the K triangles are replaced by any clique of order at least four. □

The following property partially characterizes the possible configuration of the cliques in $L_3^2(H)$.

Property 4. Let C_1 and C_2 be two cliques of $L_3^2(H)$. If $| C_1 \cap C_2 | \geq 3$ then $C_1 = C_2$.

Proof. Let $C = \{v_1, v_2, v_3\} \subseteq C_1 \cap C_2$. We proceed by contradiction assuming that there exist two non-adjacent vertices $x \in C_1$ and $y \in C_2$ such that $C'_1 = C \cup x$ and $C'_2 = C \cup y$ are two different cliques. The following two cases arise (see cases (a) and (b) in Figure 4.1).

- C is a T-triangle: we suppose w.l.g. $v_1 = \{1, 2, 3\}$, $v_2 = \{1, 2, 4\}$, and $v_3 = \{1, 3, 4\}$. Since C'_1 is a clique, then $x = \{2, 3, 4\}$ is the only possibility for x. The same reasoning holds for y, so it follows $x = y$, against the hypothesis.
- C is a K-triangle: we suppose w.l.g. $v_1 = \{1, 2, 3\}$, $v_2 = \{1, 2, 4\}$, and $v_3 = \{1, 2, 5\}$. Both x and y have to contain the pair $\{1, 2\}$, shared by all the elements of C, so there exists an edge joining them, against the hypothesis.

□

In the proof of Property 4, we note that if C is a K-triangle, then only two of its elements are enough to determine the common pair $\{1, 2\}$. The following corollaries hold.

Corollary 2. *Let C_1 and C_2 be two cliques of $L^2_3(H)$ not containing any T-triangle. If $\mid C_1 \cap C_2 \mid \geq 2$ then $C_1 = C_2$.*

Corollary 3. *Let C_1 and C_2 be two (maximal) cliques of $L^2_3(H)$. If $\mid C_1 \cap C_2 \mid = 2$, then either C_1 or C_2, but not both, are T-triangles.*

We emphasize that if both C_1 and C_2 are T-triangles such that $\mid C_1 \cap C_2 \mid = 2$, then they are not maximal, i.e., they form an S-square as in Figure 4.1, (b).

The following example shows that S is the biggest clique that does not admit a common pair in all of its edges.

Example 6. Consider a clique K_5 of a 2-intersection graph. It is not possible to label its vertices without a common pair. In fact, w.l.o.g. let us start by labelling a vertex with $\{1, 2, 3\}$ (see Figure 4.3) and one of its neighbours with $\{1, 2, 4\}$. Suppose that we label a third vertex with a pair different from $\{1, 2\}$, say $\{1, 3, 4\}$. Then we have only one label possible for the fourth vertex, namely $\{2, 3, 4\}$. We observe that there does not exist any label for the fifth vertex, since all possible pairs have been already used. So a labelling of the vertices of K_5 without a common pair is not possible.

The same argument can be used to prove that any clique of order greater than four has edges labelled with the same pair of indices.

Theorem 6. *Let G be a graph containing two distinct cliques only of order n and m and intersecting in $2 \leq i \leq \min(m, n) - 1$ vertices. Moreover, assume that the condition of Lemma 6 holds. Then the following statements hold:*

(i) *if $n = 3$ and $m \geq 3$, then $G \in L^2_3$;*

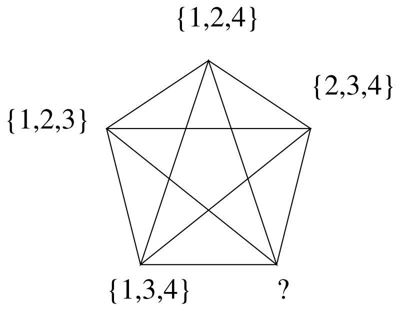


Figure 4.3: The labelling of the clique K_5 used in Example 6.

(ii) *let* $n = 4$ *and* $m \geq 4$, *then* $G \in L_3^2$ *if and only if* $i = 2$;

(iii) *if* $n > 4$ *and* $m > 5$, *then* $G \notin L_3^2$.

Proof. (i): by hypothesis, $i = 2$, so the two cliques must intersect in two vertices. From Property 4 and the corollaries, a clique K_m, with $m \geq 3$ shares a common edge, say with label $\{x, y\}$, with a T-triangle. We proceed in labelling the three vertices of T by $\{x, y, z_1\}, \{x, y, z_2\}, \{x, z_1, z_2\}$. Finally, we assign the labels $\{x, y, z_3\}, \{x, y, z_4\}, \ldots, \{x, y, z_m\}$ to the remaining vertices of K_m. The labels turn out to be the hyperedges of a 3-hypergraph H on $\{x, y, z_1, \ldots, z_m\}$ vertices such that $L_3^2(H) = G$, as desired.

(ii): from Property 4 and corollaries, if $i = 2$, then the edge shared by the two cliques must intersect a triangle T. Let T be contained in the first clique whose configuration turns out to be S. The other clique is forced to have a common label of the edges since it can not contain a second T-triangle. So, a labelling similar to that defined in (i) is allowed, providing a 3-hypergraph whose 2-intersection graph is G.

Finally, the case $i = 3$ causes the collapse of the two cliques into one single clique.

(iii): consider two cliques of order greater than four. Property 4 states that if they share two or more vertices, then they are the same clique, so G is not a 2-intersection graph.

□

The above result can be naturally extended to graphs having more than two cliques, by inspecting the intersections between pairs of them. A labelling of the edges and the determination of a 3-hypergraph compatible with them can be performed.

We now provide an example of a class of graphs contained in $\mathcal{L}_3^2$.

Cycle graphs: first, we observe that a cycle of length k admits labelling involving k integer indices at most. In fact, we start by labelling a randomly chosen vertex v_1 with $\{1, 2, 3\}$, then we visit the remaining vertices $v_2, \ldots, v_k$ of the cycle according to a chosen direction and we label the i^{th} one, where $1 < i < k - 1$, with $\{i, i+1, i+2\}$, so that $\{i, i+1\}$ is the pair shared by v_{i-1} and v_i, i.e., the label of the edge joining them.

Finally, vertices v_{k-1} and v_k are labelled with $\{1, k, k+1\}$ and $\{1, 2, k\}$, respectively. An example is shown in Figure 4.4.

Note that the labelling of the first $k-1$ vertices of the cycle defined above can be used to label a path of length $k-1$ involving k integer indices.

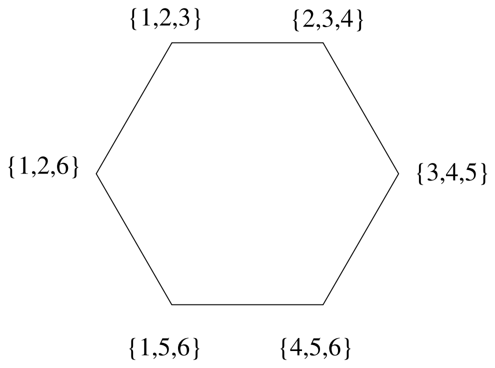


Figure 4.4: The labelling of a cycle of length 6.

4.2 Computational complexity of 2-intersection property

After having shown some structural properties of $L_3^2(H)$, here we are ready to prove that is $NP-$complete to determine whether an arbitrary graph G is the 2-intersection graph of a 3-hypergraph. We refer to this problem as the $2INT$ problem. Note that is clear that $2INT$ is in NP. We reduce from the following variant of the 3-SAT problem (L02 in [45])

MAX-3-SAT:

INSTANCE: a set U of *variables* and a set C of *clauses* over U such that each clause $c \in C$ has $|c|= 3$ *literals*, where a literal corresponding to a variable x is either x or $\overline{x}$. Each variable, as represented by a literal, appears at most three times in the clauses. Furthermore, no three occurrences of a variable are the same literal.

QUESTION: Is there a satisfying truth assignment for C?

First of all, we prove that this problem is $NP-$complete.

Lemma 7. *MAX-3-SAT is NP-complete.*

Proof. We reduce 3-SAT to MAX-3-SAT. Consider an instance of 3-SAT, in which a variable x appears k times, either as x or $\overline{x}$. Replace the first instance with x_1, the second with x_2 and so on up to x_k (possibly the complements). Then add $(\overline{x}_1 \vee x_2) \wedge$

$(\overline{x}_2 \vee x_3) \ldots (\overline{x}_k \vee x_1)$. This ensures that either all x_i are true, or all are false. Note that each x_i has two literals occurring in this expression, and one more literal occurring in the clauses of 3-SAT, providing a total of three occurrences.

Acting similarly for the remaining literals, when needed, we obtain an instance of $3-$SAT, where each variable appears at most three times, all of them not the same literal. □

Given an instance C of MAX-3-SAT, we construct a graph G_C so that there is a solution of the MAX-3-SAT instance if and only if G_C is a 2-intersection graph. This will imply that *2INT* is NP-complete. We remark that each solution of C will be achieved by one of the possible labellings of G_C that determines a 3-hypergraph H such that $G = L_3^2(H)$.

To achieve the thesis, we need to define some graph configurations that have the 2-intersection property and that are useful to model variables and clauses of 3-SAT. Each vertex of a 2-intersection graph G corresponds to a hyperedge of its related hypergraph H. So each vertex v of G contains three vertices of H. In order to avoid confusion with the use of the word vertex, we will use the term *indices* for the vertices of H contained in a vertex of G. In the following properties, we consider subgraphs of G determined by triangles and show that when G is a 2-intersection graph, the labels of G have to satisfy certain requirements, i.e., local properties of G extend to properties of H.

We will call two triangles sharing one vertex, with no edges between them, a *ribbon* configuration. Figure 4.5 shows the labels of two one-vertex intersecting triangles when at least one of them is a K-triangle.

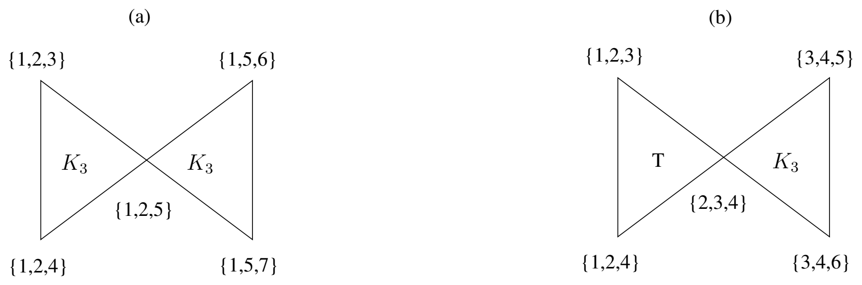


Figure 4.5: Two possible labels of a ribbon configuration. In each of them at most one triangle is a T-triangle.

The following result holds.

Property 5. A *ribbon* configuration belongs to $\mathcal{L}_3^2$ if and only the two triangles are not $T-$triangles both.

Proof. Observe that the three vertices of a T-triangle use just four different indices from H. If the two triangles are T-triangles, then the common vertex has to share four different pairs of indices with the four adjacent vertices, and this is not possible (see Figure 4.5). □

Property 6. Let T_1 and T_2 be two triangular cliques. Suppose there are just two edges joining a vertex of T_1 to a vertex of T_2, without common endpoints. Then the obtained configuration has the 2-intersection property. Furthermore, T_1 and T_2 cannot both be T-triangles.

Proof. Let x_1 and y_1 be the vertices of T_1 adjacent to the vertices x_2, and y_2 of T_2 by the edges e_x and e_y, respectively. We first show that T_1 and T_2 cannot be both T-triangles. Suppose, w.l.g., that T_1 is a T-triangle whose vertices have labels $x_1 = \{1,2,3\}$, $y_1 = \{2,3,4\}$, and $z_1 = \{1,2,4\}$ (see Figure 4.6). Since x_2 is adjacent to only x_1 in T_1, then the label of e_x must be $\{1,3\}$. With the same argument, the label of e_y is $\{3,4\}$. Since x_2 and y_2 belong to T_2, they have a common pair of labels, say $\{3,5\}$. So their labels must be $x_2 = \{1,3,5\}$ and $y_2 = \{3,4,5\}$. If we assume T_2 to be a T-triangle, then $z_2 = \{1,4,5\}$. This leads to a contradiction since z_2 is not adjacent to $z_1 = \{1,2,4\}$. Therefore the only possible label is $z_2 = \{3,5,6\}$, showing that T_2 is a K-triangle.

To show that T_1 and T_2 can both be K-triangles, we set $x_2 = \{1,3,5\}$ and $y_2 = \{3,4,5\}$ and $z_2 = \{3,5,6\}$ in T_2 (see Figure 4.6). We can then take $x_1 = \{1,2,3\}$, $y_1 = \{2,3,4\}$ and $z_1 = \{2,3,7\}$ in T_1. □

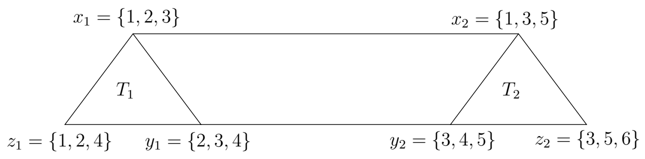


Figure 4.6: The configuration obtained by two triangles with joined by two edges. One possible labelling is shown.

Representing the variables of U

Define a *variable gadget*, denoted G_x, to represent a variable x in U. The gadget is a 2-intersection graph obtained by the union of different configurations and its definition can be checked in Figure 4.7.

The graph G_x consists of two gadgets G_x^1 and G_x^2 that grow around two K_5 cliques C_1 and C_2 that are connected by a ribbon configuration, say R.

The gadget G_x^1 includes the clique C_1 whose vertices are in common with five different K_3 triangular cliques, counter-clockwise denoted by T_i^1, with $1 \leq i \leq 5$, and starting from the triangle facing the ribbon that joins the two gadgets. These triangles are then connected in pairs by their two remaining free vertices, forming a 10-cycle, as shown in Figure 4.7.

The triangles T_2^1 and T_5^1 will be associated with the two (at most) occurrences of the variable x, while T_3^2 will be associated with the single occurrence of the variable $\overline{x}$.

We show that the graph G_x has the 2-intersection property. Furthermore, it allows very few possibilities when setting the triangles as K or T.

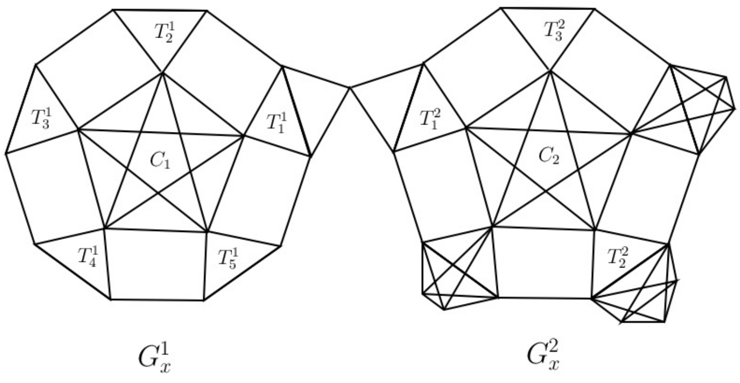


Figure 4.7: The graph G_x. It is used to model the occurrences of the literals x and $\overline{x}$ in the clauses of an instance of MAX-3-SAT. The two parts G_x^1 and G_x^2 are indicated together with their connection through a ribbon configuration.

Property 7. The variable gadget G_x is a 2-intersection graph.

In Figures 4.8 and 4.9 we provide samples of labellings of G_x.

Corollary 4. *A labelling of the vertices of the graph G_x only produces the following possible types (K or T) of the triangles T_2^1, T_5^1 and T_3^2:*

$i)$ if $T_3^2 = T$ then $T_2^1 = T_5^1 = K$;

$ii)$ if $T_5^1 = T$ (resp. $T_2^1 = T$) then $T_3^2 = K$.

Proof. $i)$ let us assume $T_3^2 = T$. Property 4.2 assures that triangle T_1^2 is a K-triangle. So, by Corollary 3, the triangle adjacent to T_1^2 is a T-triangle, and the triangle adjacent to T_1^1 is a K-triangle by Property 5. Continuing with the configurations, we get $T_1^1 = T$, and, again by Property 4.2, we obtain that both T_2^1 and T_5^1 are K-triangles as desired (see a possible labelling in Figure 4.8).

$ii)$ let us assume $T_5^1 = T$ (resp. $T_2^1 = T$). Property 4.2 assures that $T_1^1 = K$. So, by Corollary 3, the triangle adjacent to T_1^1 is a T-triangle, and the triangle adjacent to T_1^1 is a K-triangle by Property 5, and finally $T_1^2 = T$. Again Property 4.2 assures that T_3^2 is a K-triangle as desired (see a possible labelling in Figure 4.9). □

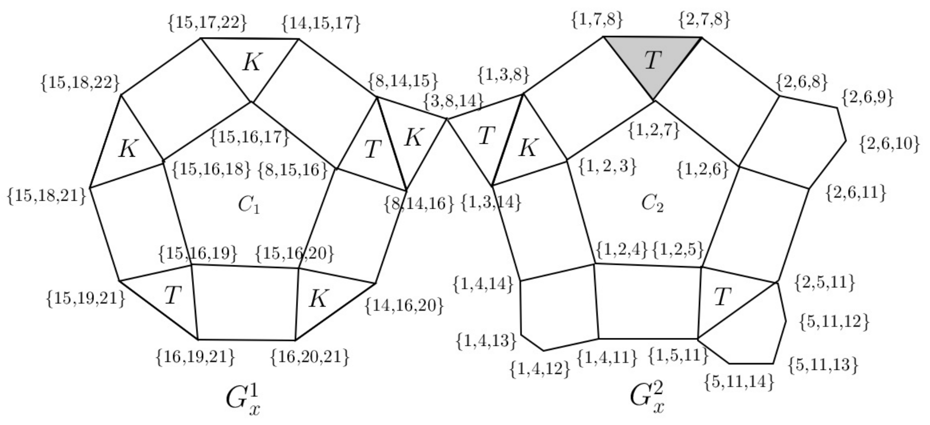


Figure 4.8: One of the labellings of G_x related to case $i)$ of Corollary 4. The starting triangle $T_3^2 = T$ is shaded. The type (T or K) of each triangle is also shown.

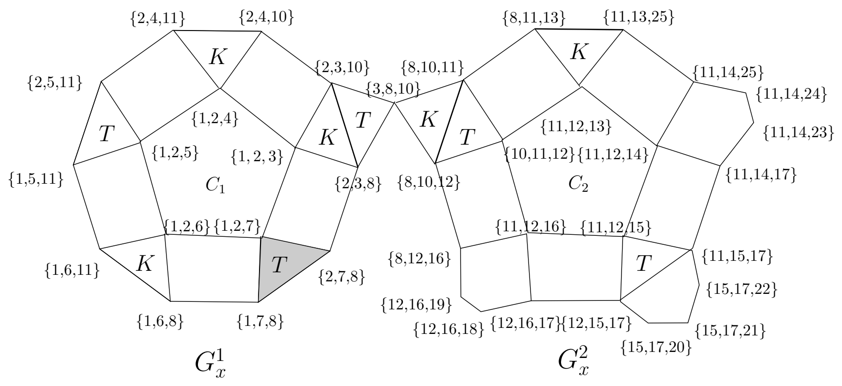


Figure 4.9: One of the labellings of G_x related to case $ii)$ of Corollary 4. The starting triangle $T_3^2 = T$ is shaded. The type (T or K) of each triangle is also shown

Comment 1. We point out that in case $ii)$ of Corollary 4 no assumption involves the type of T_2^1, that can be either T or K, when $T_5^1 = T$ and vice versa. Figure 4.9 shows an example of a labelling where $T_1^2 = K$. An easy check reveals that exchanging the types K and T between T_1^3 and T_1^2 produces a new admissible labelling.

Representing the clauses of U

The *clause gadget G_c*, represents a single clause $c \in C$. It consists of a central clique K_6 whose vertices also belong to six different K_3 cliques, called *boundary triangles*, as shown in Figure 4.10. The boundary triangles are then connected in pairs via the two remaining free vertices, as shown in Figure 4.12, forming a 12-cycle. In order to have the 2-intersection property, the clause gadget admits a maximum number of T-triangles among the six boundary ones, as stated in the following

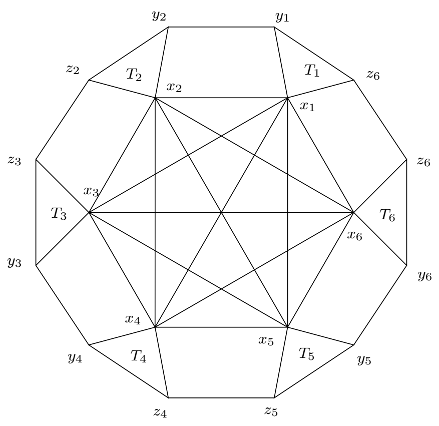


Figure 4.10: The clause gadget G_c with its triangles. By Property 4.2 no more than three T-triangles are allowed in the boundary.

Lemma 8. *A clause gadget G_c is a 2-intersection graph if and only if the boundary triangles do not contain either exactly three or exactly one T-triangles.*

Proof. W.l.g. let $T_1, \dots, T_6$ be the boundary triangles (labelled counter-clockwise) of a clause gadget. Without loss of generality, let T_i have vertices $\{x_i, y_i, z_i\}$, such that x_i is in common with the central clique K_6, while y_i and z_i connect T_i with the neighbour triangles T_{i-1} and T_{i+1}. Note that the indices $i-1$ and $i+1$ are reduced to the range $1 \dots 6$. We first show that there is no labelling when three (non-consecutive, according with Property 4.2) boundary triangles are T-triangles. Let T_1, T_3 and T_5 be T-triangles that alternate with three K-triangles T_2, T_4 and T_6. We show that such a configuration does not have the 2-intersection property, i.e., it does not allow any labelling. Assume that the edges of the central clique K_6 have the common label $\{1, 2\}$. Therefore, label every vertex $\{x_1, \dots, x_6\}$ of K_6 with $\{1, 2, i\}$, with $3 \leq i \leq 8$, respectively, and let x_1 belong to a T-triangle.

W.l.g, its two remaining vertices have labels $y_1 = \{1, 3, 9\}$ and $z_1 = \{2, 3, 9\}$. Suppose T_1 is connected by the edges (y_1, y_2) and (z_1, z_6) with the triangles T_2 and T_6, respectively. So, the edge $\{y_1, y_2\}$ has label $\{1, 9\}$, and the edge $\{z_1, z_6\}$ has label $\{2, 9\}$ (check the labelling in Figure4.11, left graph).

So, the labels $y_2 = \{1, 8, 9\}$ and $z_6 = \{2, 4, 9\}$ are determined. Since T_2 is a K-triangle, the label of z_2 requires the introduction of a new index, i.e., $z_2 = \{1, 8, 10\}$. The same holds in T_6, where the label $y_6 = \{2, 4, 11\}$ can be assumed.

Continuing labelling the boundary triangles, we observe that the edge $\{z_2, z_3\}$ has two possible labels remaining, i.e., either $\{1, 10\}$ or $\{8, 10\}$. Since T_3 is a T-triangle, then z_3 has either label $\{1, 7, 10\}$ or $\{2, 7, 10\}$; the first one only being compatible with one of the possible labels of the edge $\{z_2, z_3\}$. As a consequence, $y_3 = \{2, 7, 10\}$.

Acting similarly on the T-triangle T_5, we get $y_5 = \{2, 5, 11\}$ and $z_5 = \{1, 5, 11\}$.

Finally, a problem occurs in the labelling of the K-triangle T_4: the edges $\{z_3, z_4\}$ and $\{y_4, y_5\}$ have labels $\{2, 10\}$ and $\{1, 11\}$, respectively, preventing any connection between the vertices y_5 and z_5.

Similar reasoning reveals that the presence of exactly one boundary T-triangle does not allow labelling of the clause gadget. The failure of labelling, in this case, is shown in Figure 4.11, right (T_1 is the T-triangle). □

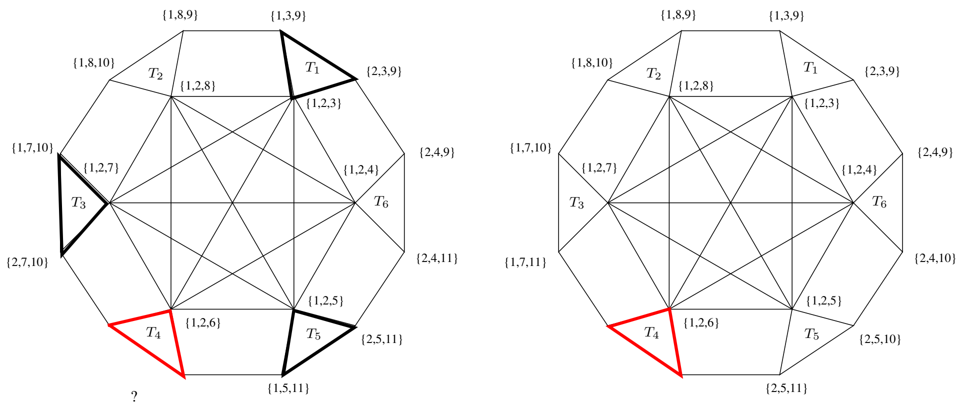


Figure 4.11: An attempt to define a labelling of a clause gadget with three (on the left) or one (on the right) boundary T-triangles. Both configurations prevent any labelling of the K-triangle T_4.

Finally, Figure 4.12 shows the possible labellings when either two or no T-triangles are present as boundary triangles.

The final NP-completeness reduction.

Let us consider the instance $C = \{c_1, \ldots, c_n\}$ of MAX-3-SAT involving the variables in the set $U = \{x_1, \ldots x_m\}$. Based on the gadgets already defined, we construct a graph G_C whose labels determine its 2-intersection property and express the desired valuations of C. The reader can follow an example of the construction of G_C in the case $C = (x_1 \vee \overline{x}_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \overline{x}_3)$ in Figure 3.5.

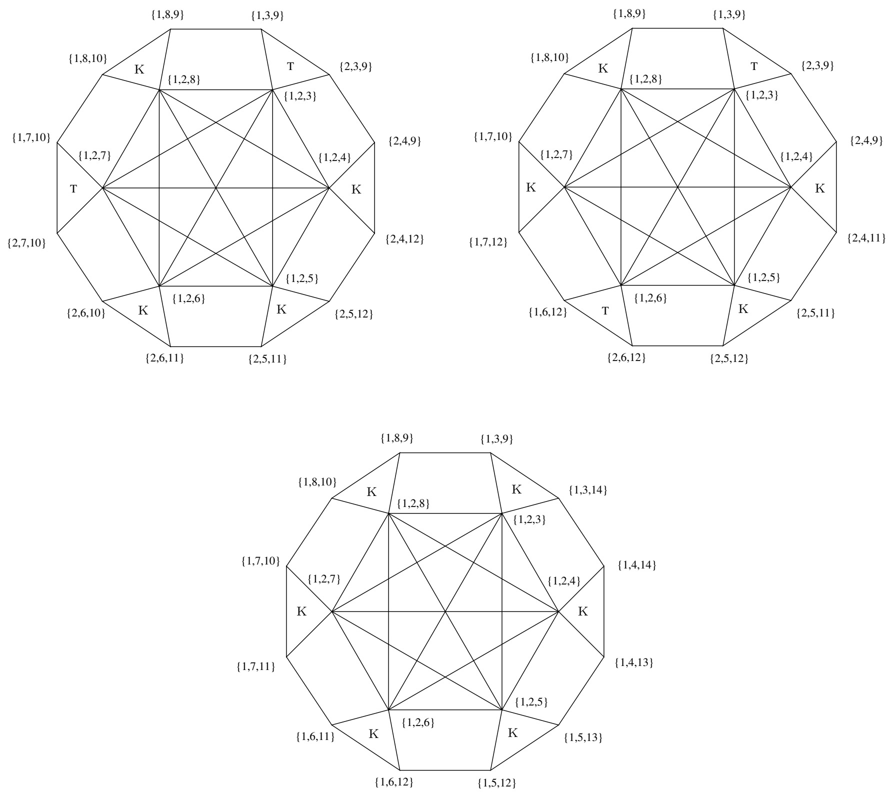


Figure 4.12: Reconstruction in the case of zero or two T-triangles.

First, from Lemma 7, we can suppose that each variable must appear at most three times: one in a form and two in the opposite form. For each variable $x_i \in U$, we define a variable gadget G_{x_i}, and associate the triangle T_3^2 with the single occurrences of a literal x_i. The at-most-two remaining occurrences of the opposite literal are associated with the triangles T_2^1 and T_5^1.

For each clause $c_j \in C$, we construct a clause gadget G_{c_j} and label its boundary triangles $T_1 \dots T_6$ as in Lemma 8.

Finally, we connect variable gadgets and clause gadgets together as follows: for each clause c_j, with $1 \leq j \leq n$, we use a ribbon to the triangle T_{2i-1} of the clause gadget G_c, to the corresponding triangle associated with the i^{th} literal of c in the G_x gadget of its variable, as in Figure 4.13.

Now, we prove the main theorem of this section.

Lemma 9. *Given an instance C of MAX-3-SAT, the graph G_C has the 2-intersection property if and only if the instance C has a solution.*

Proof. Let us assume that there exists a valuation for the MAX-3-SAT instance C. Given a variable $x \in U$, for each literal with value *true* associated with x, we assign the triangles associated with it to be T-triangles, and we assign the triangles associated with its negation to be K-triangles. Corollary 4 assures that the variable gadget G_x has a labelling. Then, for each clause $c_j \in C$, in its clause gadget G_{c_j}, we assign the triangles associated with the literals having valuation *true* to be K-triangles, but T-triangles for the literals having valuation *false*. Since three or more T-triangles are not allowed in G_{c_j} by Lemma 8, then also the clause gadget has a labelling. The labelling of the connecting ribbons is straightforward. The reader can benefit from an example of such a construction in Figure 4.13.

On the other hand, suppose G_C has the 2-intersection property, i.e., there is a labelling of G_C that determines a 3-hypergraph H such that $L_3^2(H) = G_C$. For each clause gadget G_{c_j}, with $c_j \in C$, there exists at least one triangle among T_1, T_3 and T_5, say T' of type K, by Lemma 8. Property 5 assures that T' having type K leads to a T-triangle in the variable gadget G_x to which a literal, say l, is associated (note that the opposite does not hold, as shown by the red triangle in Figure 4.13). We assign such a literal the truth value *true*. The opposite literal $\bar{l}$ is then associated with *false*. Due to Corollary 4 the triangles (one or two) in G_x associated with $\bar{l}$ are of type K.

We emphasize the following situation: it may happen that there exists a G_x labelling where all three triangles associated with the literals are of type K. In such a case, the value assigned to the variable does not affect the truth value of the clauses of C, so it can be arbitrarily assigned.

So, the valuation defined is a solution of the MAX-3-SAT instance C: each clause gadget G_{c_j} has at most one triangle among T_1, T_3 and T_5 of type K, so at least one literal in the clause has value *true*. Furthermore, in each variable gadget, if one literal is set to *true*, i.e. at least one of the corresponding triangles is of type T, then the opposite literal has the logical value *false*. □

Theorem 7. *The $2INT$ Problem is NP-complete.*

The proof directly follows from Lemma 4.2, after checking that the construction of the graph G_C associated with the instance C can be performed in polynomial time.

Comment 2. In Figure 4.13, the two highlighted triangles in different colors indicate the following situation:

- the red triangle in the ribbon connecting T_1 of G_{c_1} to T_2^1 of G_{x_3} shows that the T-triangle related to the value *false* in a clause may produce the truth value *true* in the associated literal. This is not a contradiction: it simply means that in the clause, here c_1, there exists at least a second triangle of type K, i.e. the clause is already satisfied by at least one different literal;

- the blue triangle T_1^5 in G_{x_2} is of type K which is different from the T-triangle T_2^1 although they are associated with the same literal l. This means that l may have a value *false* in a second clause in which it is possibly present. As a consequence, a different literal having truth value *true* is present in that clause. What is relevant in G_{x_2} is that the triangle associated with $\bar{l}$, i.e. T_3^2, has to be of type K, assuring that $\bar{l}$ has truth value *false* in all its occurrences.

Example 7. Consider the formula $F = (x_1 \vee \overline{x}_1 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$. Figure 4.13 shows the construction of its corresponding gadget. The labelling shown in the same figure corresponds to the valuation x_1 = *false*, x_3 = *false*. The value of x_2 doesn't matter since, in any case, the assignment is a solution for F.

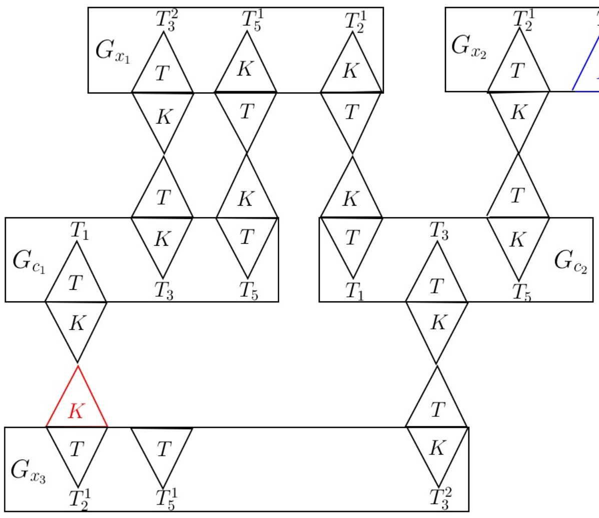


Figure 4.13: Example of the construction of the gadget for $C = \{c_1, c_2\}$, $c_1 = (x_3, x_1, \overline{x}_1)$, $c_2 = (\overline{x}_1, \overline{x_3}, x_2)$. One of the valuations obtained by the labelling of the corresponding G_C graph is $x_1 = true$, $x_2 = true$ and $x_3 = true$. The valuation can be obtained by the position of (at least one of) the T configurations in the triangles T_2^1, T_5^1 and T_3^2 of the variable gadget.

4.3 Reconstruction of Claw-Free Graphs

After having shown that decide if $G \in L_3^2$ is $NP-$complete in general, we consider some subclasses of interest. In particular, since graph in L_3^2 are $K_{1,4}-$free, we decide to focus on $K_{1,3}-$free graphs, known also as *claw-free* graphs. First, we provide some examples of claw-free graphs in $\mathcal{L}_3^2$ and then we discuss some necessary conditions to belong to the class.

We prove that, in general, determining if a claw-free graph belongs to $\mathcal{L}_3^2$ is NP-complete but it is polynomial for triangulated claw-free graphs. In the sequel we will indicate with small, medium and big a clique according to its cardinality $|K|$, i.e, $|K| \leq 2, 3 \leq |K| \leq 4, |K| \geq 5$, respectively.

4.3.1 Claw-free graphs in $\mathcal{L}_3^2$

In this section we will show some examples of claw-free graphs belonging in $\mathcal{L}_3^2$. We will use these graphs in subsequent proofs.

First of all, Figure 4.14(a) shows a claw-free graph belonging to $\mathcal{L}_3^2$. This is not always the case, as witnessed by Figure 4.14(b). In fact, using Proposition 11 it directly follows that the graph $K_5 - e$ has no realization.

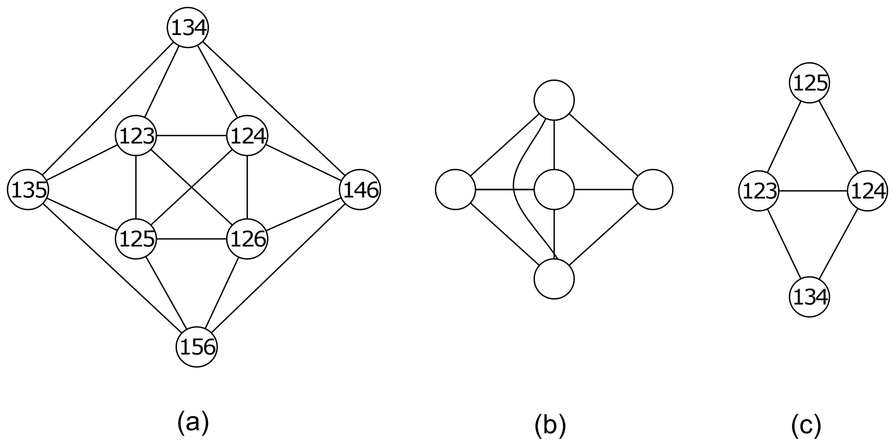


Figure 4.14: (a) A claw free graph in $\mathcal{L}_3^2$. (b) $K_5 - e$ has no realization. (c). A realization of the diamond.

Corollary 5. *If $G \in \mathcal{L}_3^2$ contains, as an induced subgraph, a diamond D consisting of the two triangles T_1, T_2 then we have either T_1^-, T_2^+ or T_1^+, T_2^- (see Figure 4.14)*(c).

Corollary 6. *If $G \in \mathcal{L}_3^2$ contains, as an induced subgraph, a 4-wheel W_4 consisting of the four triangles $T_1 = \{a, b, c\}, T_2 = \{a, c, d\}, T_3 = \{a, d, e\}, T_4 = \{a, b, e\}$ then we have either $T_1^-, T_2^+, T_3^-, T_4^+$ or $T_1^+, T_2^-, T_3^+, T_4^-$ (see Figure 4.15).*

Corollary 7. *If $G \in \mathcal{L}_3^2$ then it cannot contain the 5-wheel W_5 as an induced subgraph (see Figure 4.15 (a)).*

The prism P consists of two vertex disjoint triangles $T_1 = \{a, b, c\}, T_2 = \{d, e, f\}$ plus the three edges ad, be, cf.

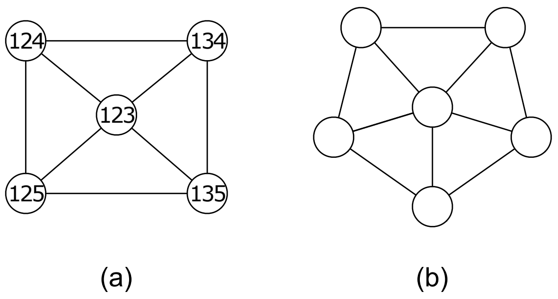


Figure 4.15: (a) A realization of the 4-wheel W_4. (b) the 5-wheel W_5

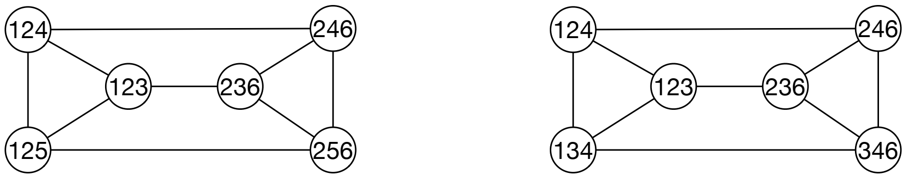


Figure 4.16: Two realizations of the prism.

Fact 1. If $G \in \mathcal{L}_3^2$ contains, as an induced subgraph, a prism P then we have T_1^+, T_2^+ or T_1^-, T_2^- (see Figure 4.16(b)).

Proof. By contradiction, assume that T_1^+, T_2^-. Let $a = \{1,2,3\}, b = \{1,2,4\}, c = \{1,2,5\}$. Without loss of generality $d = \{1,3,6\}$. Then we have $e = \{1,6,4\}$. It follows that $f = \{1,5,6\}$, so T_2 is positive, a contradiction. $\square$

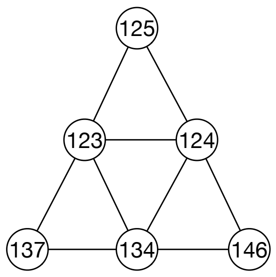


Figure 4.17: A realization of the sun S_3.

Fact 2. If $G \in \mathcal{L}_3^2$ contains, as an induced subgraph, a sun S_3 consisting of the four triangles $T_1 = \{a,b,c\}, T_2 = \{a,b,d\}, T_3 = \{a,c,e\}, T_4 = \{b,c,f\}$ then we have

$T_1^-, T_2^+, T_3^+, T_4^+$ with $a = \{1,2,3\}, b = \{1,2,4\}, c = \{1,3,4\}, d = \{1,2,5\}, e = \{1,3,6\}, f = \{1,4,7\}$ (see Figure 4.17).

Proof. From Corollary 1 we have either $T_1^-, T_2^+, T_3^+, T_4^+$ or $T_1^+, T_2^-, T_3^-, T_4^-$. The figure 4.17 shows a realization with $T_1^-, T_2^+, T_3^+, T_4^+$. Now we assume that $T_1^+, T_2^-, T_3^-, T_4^-$ with $a = \{1,2,3\}, b = \{1,2,4\}, c = \{1,2,5\}$. Then, w.l.o.g., $d = \{1,3,4\}$. It follows that $e = \{2,4,5\}$ but f cannot be labelled. □

Fact 3. If $G \in \mathcal{L}_3^2$ contains, as an induced subgraph, a path on three vertices uvw with $u = \{a,b,c\}, w = \{d,e,f\}$ then $\{a,b,c\} \cap \{d,e,f\} \neq \emptyset$.

As mentioned before, we denote with $K_4{+}v$ is the graph with five vertices $\{a,b,c,d,e\}$ where $\{a,b,c,d\}$ is complete and e is connected to exactly one vertex, say a.

Fact 4. If $G \in \mathcal{L}_3^2$ contains $K_4{+}v$, as an induced subgraph, then the clique K_4 of $K_4{+}v$ is positive.

Proof. By contradiction, assume that the clique on four vertices $G[\{a,b,c,d\}]$ is negative: $a = \{1,2,3\}, b = \{1,2,4\}, c = \{1,3,4\}, d = \{2,3,4\}$. Then, without loss of generality, $e = \{1,2,5\}$. So $be \in E$, a contradiction. □

As previously underlined, all the graphs considered in this section are claw-free.

4.3.2 Complexity of recognizing claw free graphs in $\mathcal{L}_3^2$

In Section 4.2 we show that reconstructing a graph in $\mathcal{L}_3^2$ is $NP-$complete. Since a graph in $\mathcal{L}_3^2$ is $K_{1,4}$-free we are concerned with the subclass of claw-free graphs ($K_{1,3}$-free graphs).

We will prove that the problem of deciding if $G \in \mathcal{L}_3^2$ is NP-complete. To reach our goal we need an intermediate problem that is defined and proved NP-complete below.

The 2-labelling intersection ($2LI$) problem

Let us consider a simple graph $G = (V,E)$ and a partition of its edge-set E into two subsets E_w and E_s, i.e. $E = E_w \cup E_s$ and $E_w \cap E_s = \emptyset$. We call *weak edges* the edges in E_w and *strong edges* those in E_s.

We define a function φ, say a 2-*labelling*, that associates to each vertex $v \in V$ a pair of labels $\{a_v, b_v\}$ such that:

$i)$ if $v_1 \neq v_2$, then $\varphi(v_1) \neq \varphi(v_2)$;

$ii)$ if $v_1v_2 \in E_w$, then $\varphi(v_1) \cap \varphi(v_2) = \emptyset$;

$iii)$ if $v_1v_2 \in E_s$, then $|\varphi(v_1) \cap \varphi(v_2)| = 1$.

See Fig.4.18 for an example.

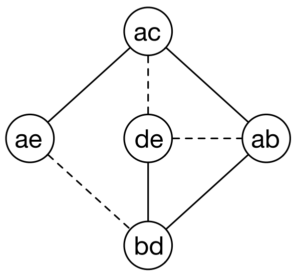


Figure 4.18: A graph with a 2-labelling φ. Weak and strong edges are represented by dotted and straight lines, respectively, while the nodes show the pair of elements associated by φ.

The $2LI$ problem, provided in its decision form, follows

2-Labelling Intersection (2LI):

INSTANCE: a simple graph $G = (V, E)$ and a partition of its edges into E_w and E_s.

QUESTION: does G admit φ a 2-labelling of its vertices?

We show the NP-completeness of $2LI$ by a reduction that involves the problem $3 - SAT$ (LO2 in [45]).

3-SAT:

INSTANCE: a set U of variables, a collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$.

QUESTION: Is there a satisfying truth assignment for C?

Given an instance A of $3 - Sat$, we construct a graph $G_A = (V_A, E_A)$ and a partition of its edges into weak and strong edges $E_{A,w}$ and $E_{A,s}$ such that the $3 - Sat$ instance admits a solution if and only if G_A admits a 2-labelling. This will imply the NP-completeness of $2LI$.

Hence, we start by providing two graphs' prototypes to model variables and clauses of the 3-Sat instance A, then we show how to use them to reach the graph G_A.

Representing the variables and the clauses of A

Let us define the graph $G_x = (V_x, E_x)$ that will be used to represent each variable $x \in U$. The set V_x consists of three vertices v_1^x, v_2^x, and v_3^x, while E_x is partitioned into the weak edges $E_{x,w} = \{v_2^x v_3^x\}$ and the strong edges $E_{x,s} = \{v_1^x v_2^x, v_1^x v_2^x\}$ (see Fig.4.19, (a)).

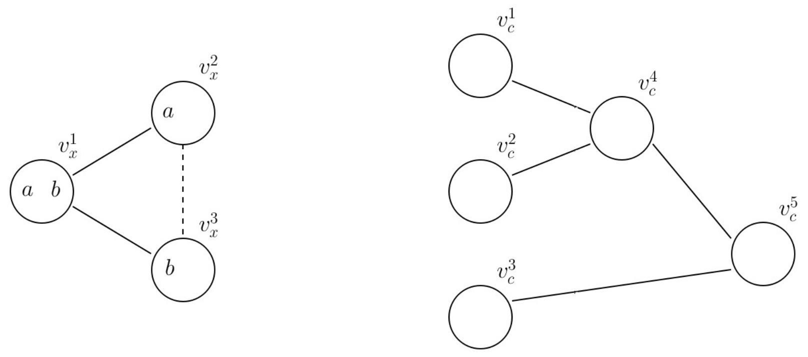


Figure 4.19: (a) the graph defined for a variable $x \in U$; (b) the graph defined for a clause $c \in C$. In (a) the nodes show the pair of elements associated by φ.

Lemma 10. *Let G_x be the graph related to variable x and $\varphi(v_1^x) = \{a, b\}$. It holds, w.l.o.g., that $a \in \varphi(v_2^x)$ and $b \in \varphi(v_3^x)$.*

The proof is immediate by definition of weak and strong edges.

The gadget G_x will also be used to represent the dichotomy of the truth values in the final graph. In particular, from now on, we consider the truth values labels T and F. We define G_{TF} similarly to G_x with the further assumption that $a = T$ and $b = F$.

On the other hand, we associate to each clause $c \in C$ a graph $G_c = (V_c, E_c)$ having five vertices $V_c = \{v_c^1, v_c^2, v_c^3, v_c^4, v_c^5\}$ and four strong connections, i.e., $E_c = E_{c,s} = \{v_c^1 v_c^4, v_c^2 v_c^4, v_c^3 v_c^5, v_c^4 v_c^5\}$ (see Fig.4.19, (b)).

Putting things together

So, starting from an instance A of $3 - SAT$, we proceed in defining the graph $G_A = (V_A, E_A)$. The reader can follow the construction in Fig.4.20. We include in G_A $n = |U|$ graphs $G_{x_1}, \ldots, G_{x_n}$ representing the variables of U, $m = |C|$ graphs $G_{c_1}, \ldots, G_{c_m}$ representing the clauses of C, and the graph G_{TF}.

The connections between these graphs in G_A are set according to the following rules:

(1) the variables are connected by all the possible weak edges between the vertices v_x^1, i.e., for each couple of variable x and y in U, we set the weak edge $v_x^1 v_y^1 \in E_{A,w}$;

(2) let x be the i-th variable involved in the clause c, with $1 \leq i \leq 3$. We set the weak edge $v_x^1 v_c^i \in E_{A,w}$, and the strong edge $v_x^3 v_c^i \in E_{A,s}$ if x is negated, $v_x^2 v_c^i \in E_{A,s}$ otherwise;

(3) for each variable x, we set the strong edges $v_x^2 v_{TF}^1, v_x^3 v_{TF}^1 \in E_{A,s}$ to connect the variable to the truth values in G_{TF}. Furthermore, we set three more strong edges

$v_c^4 v_{TF}^1, v_c^5 v_{TF}^1, v_c^5 v_{TF}^2 \in E_{A,s}$ to connect also each clause c to the truth values in G_{TF}.

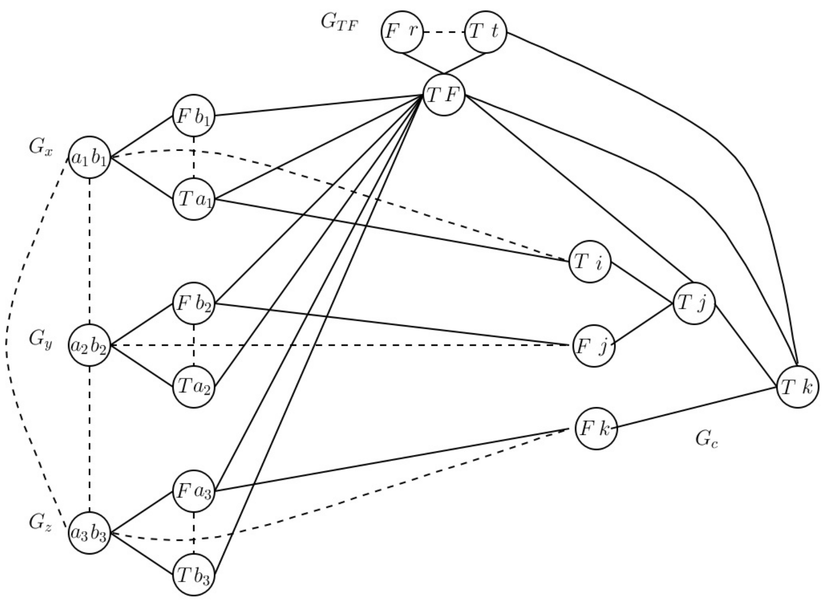


Figure 4.20: The graph G_A for a clause $C = \bar{x} \vee y \vee z$

Theorem 8. *Given an instance A of 3-Sat, the graph G_A admits a 2-labelling if and only if the instance A has a solution.*

Proof. Suppose that a 2-labelling φ exists. Let $\{a_i, b_i\}$ be the label associated to the node $v_{x_i}^1$ of G_{x_i}. The connections in (1) assure that the labels $\{a_i, b_i\}, 1 \leq i \leq n$, have no common elements.

By the edges in (2), for each variable x, one among v_x^1 and v_x^2 contains T, while F belongs to the other since their labels can not intersect by definition of G_x. Let us consider a clause c involving literals of the (distinct) variables x, y and z. $\varphi(v_c^1)$ contains, by the edges in (2), the truth value either in $\varphi(v_x^2)$ or in $\varphi(v_x^3)$ of G_x to whom it is strongly connected. Note that $\varphi(v_c^1)$ and $\varphi(v_x^1)$ does not intersect since a weak edge is set between them in (2).

Now, consider the node v_c^4: it is strongly connected both with v_{TF}^1, v_c^1, and v_c^2, so its label contains T or F according to the values of $\varphi(v_c^1)$ and $\varphi(v_c^2)$. More precisely, the following three cases arise:

- $T \in \varphi(v_c^1)$ and $T \in \varphi(v_c^2)$: it follows that $T \in \varphi(v_c^4)$;
- $T \in \varphi(v_c^1)$ and $F \in \varphi(v_c^2)$, or conversely: it follows that one among T or F, but not both, belongs to $\varphi(v_c^4)$;

- $F \in \varphi(v_c^1)$ and $F \in \varphi(v_c^2)$: it follows that $F \in \varphi(v_c^4)$.

Finally, consider the node v_c^5: since it is strongly connected to v_{TF}^1 and v_{TF}^2, its label contains, w.l.o.g. T. By the three cases above, if $T \in \varphi(v_c^5)$, then it holds $T \in \varphi(v_c^3)$ or $T \in \varphi(v_c^4)$. So, $F \in \varphi(v_c^1)$, $F \in \varphi(v_c^2)$, and $F \in \varphi(v_c^3)$ if and only if G_c does not admit a 2-labelling, and the same holds for G_A. Since the three truth values in $\varphi(v_c^1)$, $\varphi(v_c^2)$, and $\varphi(v_c^3)$ are the truth values of the literals in c, then a truth assignment for c exists if and only if a 2-labelling for G_c does.

It is clear that the converse holds, as the construction is reversible. $\square$

Theorem 9. *Let $G = (V, E)$ be a claw-free graph. Deciding whether there exists a 3-uniform hypergraph H such that $G = L_3^2(H)$ is NP-complete.*

Proof. The proof has two parts. Firstly, we define $\mathcal{C}$ a subclass of claw-free graphs we are interested here. Then we show that the problem of deciding whether $G \in \mathcal{C}$ is such that $G \in \mathcal{L}_3^2$ is equivalent to the problem $2LI$ which is NP-complete from Theorem 8.

First: the definition of $\mathcal{C}$. A graph $G \in \mathcal{C}$ consists of components $C_1, \dots, C_k$ each of the C_i's being a clique of size at least five, the C_i's form a partition of the vertex set of G. When two components C_i, C_j are linked they are connected by either a *strong link* or a *weak link*. A strong link consists of a C_4 of G with its two non-adjacent vertices $i_1, i_2 \in C_i$ and the two other non-adjacent vertices $j_1, j_2 \in C_j$. A weak link consists of a K_4 of G with two vertices $i_1, i_2 \in C_i$ and the two other vertices $j_1, j_2 \in C_j$. The links, weak or strong, have no common vertices. It follows that G is claw-free. Moreover, since $|C_i| \geq 5$, $C_i \cup C_j$ the union of two distinct components cannot be a clique.

When $G \in \mathcal{L}_3^2$ the components satisfy the following: Since each component $C_i = \{v_1^i, v_2^i, \dots, v_p^i\}$ has at least five vertices, we necessarily have $v_1^i = \{i, i', 1\}, v_2^i = \{i, i', 2\}, \dots, v_p^i = \{i, i', p\}$ and C_i is associated with its pair of common labels $\{i, i'\}$. For two distinct components C_i, C_j we have $|\{i, i'\} \cap \{j, j'\}| \leq 1$. When C_i, C_j are connected with a strong link then we have $|\{i, i'\} \cap \{j, j'\}| = 1$. When C_i, C_j are connected with a weak link then we have $\{i, i'\} \cap \{j, j'\} = \emptyset$.

Second: equivalence with the problem $2LI$. Given $G \in \mathcal{C}$ we define the graph G' as follows: to the vertices v_i of G' correspond the components C_i of G, and vice versa; to a strong (resp. weak) link of G corresponds a strong (resp. weak) edge of G', and vice versa.

We assume that there exists a 3-uniform hypergraph H such that $G = L_3^2(H)$. Since $|C_i| \geq 5$ the intersection of the labels of the pairs of vertices in the same component C_i is the same two labels says $\{i, i'\}$. Now, for two distinct C_i, C_j, since $C_i \cup C_j$ is not a clique we have that $|\{i, i'\} \cap \{j, j'\}| \leq 1$. When C_i, C_j are strongly connected then $|\{i, i'\} \cap \{j, j'\}| = 1$, when they are weakly connected then $|\{i, i'\} \cap \{j, j'\}| = 0$. Thus for each vertex v_i of G' when assigning the two labels i, i to v_i we obtain a positive answer for the problem $2LI$.

Now, we assume that the problem $2LI$ has a positive answer. Let i, i' be the two labels assigned to v_i in G'. We assign $\{i, i'\}$ to the component C_i of G. Let v_i, v_j be two vertices linked with a strong edge. Then the labels associated to C_i, C_j are $\{i, i'\}, \{i, j\}$, respectively. Let w_1^i, w_2^i and w_1^j, w_2^j be respectively the two vertices of the strong link between C_i and C_j. Then $w_1^i = \{i, i', a\}, w_2^i = \{i, i', b\}, w_1^j = \{i, j, a\}, w_2^j = \{i, j, b\}$. Let v_i, v_j be two vertices linked with a weak edge. Then the labels associated to C_i, C_j are $\{i, i'\}, \{j, j'\}$, respectively. Let w_1^i, w_2^i and w_1^j, w_2^j be respectively the two vertices of the weak link between C_i and C_j. Then $w_1^i = \{i, i', j\}, w_2^i = \{i, i', j'\}, w_1^j = \{j, j', i\}, w_2^j = \{j, j', i'\}$. Thereafter, when a vertex $w_k^i \in C_i$ is not contained in a link, weak or strong, we take $w_k^i = \{i, i', k\}$. Thus there exists H a 3-uniform hypergraph H such that $G = L_3^2(H)$. $\square$

4.3.3 Complexity of recognition for triangulated claw-free graphs in $\mathcal{L}_3^2$

Recall that a graph G is *triangulated* (or *chordal*) if it is C_k-free, $k \geq 4$.

It is known that to each triangulated graph $G = (V, E)$, we can associate, in linear time [46], a maximal clique tree $T = (C, S)$ where each maximal clique of G corresponds to a vertex $c \in C$ and $cc' \in S$ if $c \cap c' \neq \emptyset$ and $c \cap c'$ is a minimal separator of G which is a clique. An example is shown in Figure 4.21

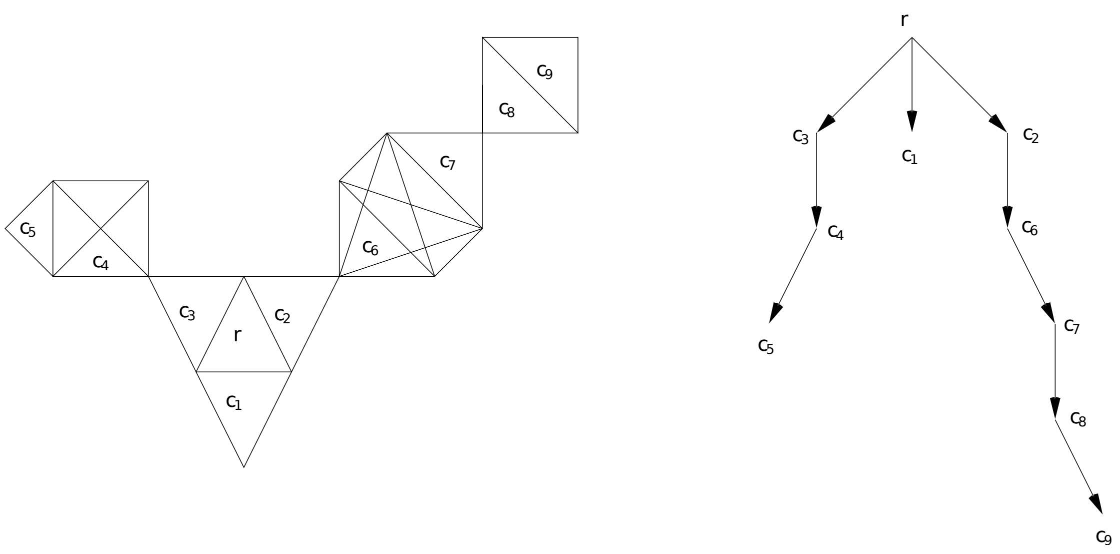


Figure 4.21: Example of a claw-free triangulated graph and its associated directed tree. We consider the clique r as the root of the tree. The three leaves are highlighted.

From Property 4 if $G \in \mathcal{L}_3^2$, then an edge $cc' \in S$ corresponds either to a strong intersection when $|c \cap c'| = 2$ or to a weak intersection when $|c \cap c'| = 1$. Moreover, when $cc' \in S, |c|, |c'| \geq 3$ we can easily obtain the following:

- c and c' weakly intersect: let $c \cap c' = \{a\}$; there exists $\{u, v\} \in c, \{u', v'\} \in c'$ such that $G[\{a, u, v, u', v'\}]$ is a ribbon, otherwise a cannot be a separator of G;
- c and c' strongly intersect: let $c \cap c' = \{a, b\}$; there exists $u \in c, u', \in c'$ such that $G[\{a, b, u, v\}]$ is a diamond, otherwise $\{a, b\}$ cannot be a separator of G.

The following lemma holds.

Lemma 11. *Let $G = (V, E)$ be a triangulated claw-free graph and K_t be a clique of G such that $t \geq 4$. If K_i, K_j are two (distinct) cliques that strongly intersect K_t then $K_i \cap K_j \cap K_t = \emptyset$.*

Proof. By contradiction, we assume $v \in K_i \cap K_j \cap K_t$. Let $K_i \cap K_t = \{v, v_i\}$, $K_j \cap K_t = \{v, v_j\}, v_i \neq v_j$. Since $t \geq 4$ there exists $k \in K_t \setminus \{v, v_i, v_j\}$. Let $w_i \in K_i \setminus \{K_j \cup K_t\}$ and $w_j \in K_j \setminus \{K_i \cup K_t\}$. We have $w_i w_j \in E$, otherwise $G[\{v, k, w_i, w_j\}]$ is a claw. We also have $v_i v_j \in E$ for a similar reason. But then $G[\{v_i, w_i, w_j, v_j\}] = C_4$ which is not possible since G is triangulated. □

Figure 4.22 shows the situation described in Lemma 11. The following theorem states the polynomiality of the reconstruction of a 3-uniform hypergraph H from a triangulated claw-free graphs G such that $G = L_3^2(H)$. To help the reader, the proof is divided into small steps that lead to the final result.

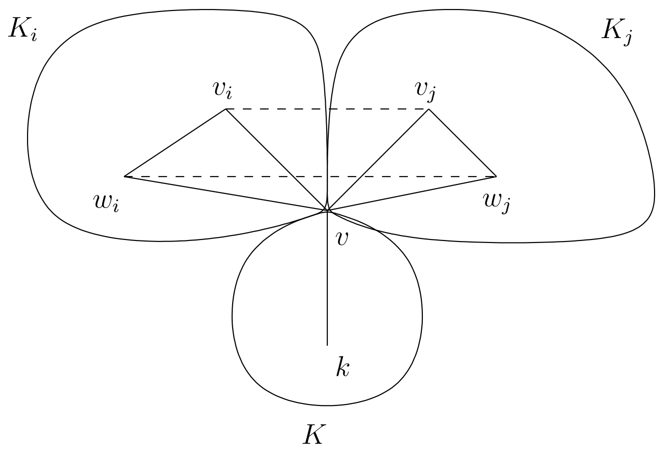


Figure 4.22: Visual example of the proof of Fact 11. Curves represent the set of nodes inside the same clique. Dashed edges are forced by claw-free property.

Theorem 10. *Let $G = (V, E)$ be a triangulated claw-free graph. Deciding whether a 3-uniform hypergraph H exists such that $G = L_3^2(H)$ and reconstructing one of them, when possible, can be performed in polynomial time.*

Proof. Let $G = (V, E)$ be a triangulated claw-free graph. We can assume that G is connected and $|V| \geq 3$. We first consider two cases.

G has an edge cut

Let $e = v_1v_2 \in E$ be an edge-cut of G. We call $\widetilde{G}_1, \widetilde{G}_2$ the two components of $\widetilde{G} = G - e$, with $v_1 \in \widetilde{G}_1, v_2 \in \widetilde{G}_2$.

We will denote $G_1 = \widetilde{G}_1 + v_2$ and $G_2 = \widetilde{G}_2 + v_1$.

Since G is claw-free, v_1 is a vertex of at most two cliques: the clique $G[\{v_1, v_2\}]$ and one clique K_n^1 of $\widetilde{G}_1$. The same holds for v_2, which can belong respectively to one clique K_m^2 of $\widetilde{G}_2$ or to $G[\{v_1, v_2\}]$.

First, we suppose that v_1 is a leaf of G (i.e. $deg_G(v_1) = 1$). In such a case, obviously $G = G_2$.

Fact 5. G has a realization if and only if $\widetilde{G}_2$ has a realization such that K_m^2 is positive when $m = 4$.

Proof. From Fact 4 if G has a realization then K_m^2 is positive when $m = 4$.

Conversely, assume that $\widetilde{G}_2$ such that K_m^2 is positive when $m = 4$. The following cases arise:

- if $m = 4$ suppose that $K_m^2 = \{v_2, v_3, v_4, v_5\}$ with $v_2 = \{1, 2, 3\}, v_3 = \{1, 2, 4\}, v_4 = \{1, 2, 5\}, v_5 = \{1, 2, 6\}$. Then we can take $v_1 = \{1, 3, 0\}$;

- if $m \geq 5$ then K_m^2 is positive and we do as before.

- if $m = 3$ we have two further cases.

  If K_m^2 is positive, we proceed as above. On the other hand, if it is negative, let $K_m^2 = \{v_2, v_3, v_4\}$ with $v_2 = \{1, 2, 3\}, v_3 = \{1, 2, 4\}, v_4 = \{1, 3, 4\}$. Then we take $v_1 = \{2, 3, 0\}$;

- Finally $K_m^2 = \{v_2, v_3\}$ with $v_2 = \{1, 2, 3\}, v_3 = \{1, 2, 4\}$. We take $v_1 = \{1, 3, 0\}$.

□

We now suppose both v_1 and v_2 are not leaves.

Fact 6. G has a realization if and only if G_1 and G_2 have a realization.

Proof. Obviously, if G has a realization also G_1 and G_2 have it. Therefore we focus on sufficiency.

The following cases arise:

- suppose that K_n^1 and K_m^2 are positive and that the vertices of K_n^1 have the labels $\{1, 2, a_k\}, 1 \leq k \leq n$ with $v_1 = \{1, 2, a_1\}$, while the vertices of K_m^2 have the labels $\{2, 3, b_k\}, 1 \leq k \leq m$ with $v_2 = \{2, 3, b_1\}$. Taking $a_1 = b_1$ and $a_i \neq b_j$ with $1 < i \leq n$ and $1 < j \leq m$, we obtain a realization for G;

- suppose K_n^1 is positive and K_m^2 is negative. Suppose also that K_n^1 vertices have labels $\{1, 2, a_k\}, 1 \leq k \leq n$ with $v_1 = \{1, 2, a_1\}$.

  From Fact 4 we have $m = 3$. W.l.o.g., the vertices of K_m^2 have the labels $\{2, 3, 4\}, \{2, 3, 5\}, \{3, 4, 5\}$ with $v_2 = \{2, 3, 4\}$. Taking $a_1 = 4$ we obtain a realization for G;

- suppose K_n^1 is positive and $m = 2$. Suppose also that the vertices of K_n^1 have labels $\{1, 2, a_k\}, 1 \leq k \leq n$ with $v_1 = \{1, 2, 3\}$. W.l.o.g., for the realization of G_1 we have $v_2 = \{1, 3, 4\}$, then up to a renaming of the labels for the realization of G_2' we have a realization for G;

- suppose K_n^1 and K_m^2 are negative. From Fact 4 we have $n = m = 3$. For the realization of G_1 let $v_1 = \{1, 2, 3\}, v_2 = \{2, 3, 5\}$ and let $\{1, 2, 4\}, \{1, 3, 4\}$ be the labels of the two other vertices of K_n^1. Then up to a renaming of the labels for the realization of G_2' with $\{3, 5, 6\}, \{2, 5, 6\}$ the labels of the two other vertices of K_m^2, we have a realization for G;

- Suppose $n = m = 2$. For the realization of G_1 let $v_1 = \{1, 2, 3\}, v_2 = \{1, 3, 5\}$ and let $\{1, 2, 4\}$ be the label of the other vertex of K_n^1. Up to a renaming of the labels for the realization of G_2' we have a realization for G.

□

G has no edge-cut:

If G has no edge cut, then the following fact is readily obtained.

Fact 7. G has no small cliques.

Proof. By contradiction we assume that $K = G[\{v_i v_j\}]$ is a clique with $v_i \in K_i, v_j \in K_j$, where $K_i \neq K, K_j \neq K$, are two distinct cliques. Since $v_i v_j$ is not a cut-edge, then there exists a path $v_i - v_k - \cdots - v_j$ where $v_k \neq v_j$. Assume that it is one of the shortest paths. Then $v_i - v_k - \cdots - v_j - v_i$ is an induced cycle of length greater than three, a contradiction. □

Therefore, we suppose without loss of generality that G contains only medium or big cliques.

Using the algorithm given in [46], we obtain a maximal clique tree $T = (C, S)$, a maximal clique tree of G in time $O(n)$ where a vertex $c \in C$ corresponds to a big or a medium clique of G. We show how to define a labelling.

First, we need the following fact.

Fact 8. Let $c \in C$ with $|c| = 3$. Then, c has at most 3 neighbours in T.

Proof. Suppose that c has more than 3 neighbours. Then, considering the previous facts, only two cases are possible: at least two cliques weakly intersect c in the same node v or one strongly intersects c in $\{v, w\}$ and, as last case, the other weakly intersects c, without loss of generality, in $\{v\}$. We consider these cases separately:

- assume that c has two neighbors c_1, c_2 that weakly intersect it in $v \in c$. Let $w \in c, w \neq v$. Since the cliques are distinct and of size greater than one, there must exist $v_i \in c_i, v_i \neq v, i \in \{1, 2\}$. Remember that, if c_1, c_2 are neighbours of c, it means that $\{v\}$ is a separator in G. Therefore, $wv_1, wv_2, v_1v_2 \notin E$, but $G[\{v, w, v_1, v_2\}] = K_{1,3}$, a contradiction;
- we assume that c has two neighbors c_1, c_2 such that $c_1 \cap c = \{v, v'\}$ and $c_2 \cap c = \{v\}$. Let $w \in c, w \neq v, v'$. Since $\{v, v'\}$ and $\{v\}$ are two separators $wv_1, wv_2, v_1v_2 \notin E$, but $G[\{v, w, v_1, v_2\}] = K_{1,3}$, a contradiction.

Thus c has at most three neighbours. □

Note that for the case where c has three neighbours, either c is the central triangle of a sun or it weakly intersects three cliques c_1, c_2, c_3 in each of their vertices.

Going back to the main problem, we will associate a label, $\lambda(c) \in \{+, -\}$ to each vertex c of T. The idea is that the label associated with a clique represents its negativity or positivity.

In a first stage, Algorithm 2 performs a partial labelling. In this step only the *local* forcings based on big cliques, Facts 4 and 2 are taken into account. The last if sentence of the algorithm consists of checking that no two negative cliques are adjacent in T.

Then, in the second stage, Algorithm 3 propagates, from the bottom to the top, the labelling from neighbour to neighbour of the cliques already labelled.

In particular, for the vertices not yet labelled (corresponding to some of the medium cliques) the algorithm proceeds as follows:

1. choose arbitrarily $c_r \in C$ as the root of the directed tree T_r. For each $c \neq c_r$ we orient the edges of the unique path $c_r - \cdots - c$ so that it becomes a directed path starting from c_r). Then we proceed from the leaves to the root of T_r, using the already given labels;
2. let c be a non labelled leaf and c' be its predecessor in T_r. Suppose c and c' strongly intersect. Using Proposition 3 we have that if $\lambda(c') = (+)$ (resp. $\lambda(c') = (-)$) then $\lambda(c) = (-)$ (resp. $\lambda(c') = (+)$). Suppose now that c and c' weakly intersect. From Property 5 we have that if $\lambda(c') = (-)$ then $\lambda(c) = (+)$. Note that in this case $|c'| = 3$ must hold;

Algorithm 2 Preprocessing Labelling

```
Require: Maximal Clique Tree $T = (C, S)$
Ensure: A partial labelling of $T$
  for all $c \in T$ do
    if $|c| \geq 5$ then
      $\lambda(c) = +$ {a big clique is positive}
    end if
    if $|c| = 4$ and there exists $c'$ that weakly intersects $c$ then
      $\lambda(c) = +$ {by Fact 4}
    end if
    if $|c| = 3$ and there are $c_1, c_2, c_3$ such that $|c_1| = |c_2| = |c_3| = 3$ and $c_1, c_2, c_3$
    strongly intersects $c$ in different edges then
      $\lambda(c) = +$ {by Fact 2 noticing that $c, c_1, c_2, c_3$ induce a sun}
      for i=1 to 3 do
        $\lambda(c_i) = -$
      end for
    end if
  end for
  for all $cc' \in S$ do
    if $\lambda(c) = \lambda(c') = -$ then
      return $G \notin \mathcal{L}_3^2$
    end if
  end for
```

Algorithm 3 Labelling

```
Require: A partially labelled tree $T = (C, S)$
Ensure: A labelling of $T$
  Let $c_r \in C$ as the root of the directed tree $T_r$
  for all $c \in T_r$ and $\lambda(c) = \emptyset$, from the leaves to $c_r$ do
    Let $c'$ be the predecessor of $c$ in $T_r$
    if $c$ is a leaf then
      if $c$ and $c'$ strongly intersect then
        if $\lambda(c') = +$ then
          $\lambda(c) = -$
        end if
        if $\lambda(c') = -$ then
          $\lambda(c) = +$
        end if
      end if
      if $c$ and $c'$ weakly intersect then
        if $\lambda(c') = -$ then
          $\lambda(c) = +$ {in this case $|c'| = 3$ must hold}
        end if
        if $\lambda(c') = +$ then
          $\lambda(c) = -$
        end if
      end if
    end if
    if $c$ is not a leaf then
      if ($c$ has two successors $c_1, c_2$ that are strongly connected with $c$ and $\lambda(c_1) \neq$
      $\lambda(c_2)$) or ($c$ has a successor $c_1$ that is strongly connected with $c$, $\lambda(c_1) = +$ and
      $c$ has a successor $c_2$ that is weakly connected with $c$, $\lambda(c_2) = -$ ) then
        return $G \notin \mathcal{L}_3^2$
      end if
      if $c$ has a successor $c_1$ strongly connected with $c$ then
        if $\lambda(c_1) = +$ then
          $\lambda(c) = -$
        end if
        if $\lambda(c_1) = -$ then
          $\lambda(c) = +$
        end if
      end if
      if $c$ has a successor $c_1$ weakly connected with $c$ and $\lambda(c_1) = -$ then
        $\lambda(c) = +$
      end if
      if $\lambda(c) = \lambda(c') = -$ then
        return $G \notin \mathcal{L}_3^2$
      end if
    end if
  end for
```

3. then, the cases in which c is not a leaf are considered. In the first if we are checking the cases in which a label is not compatible with the graph. If those conditions do not apply, we continue with the standard rules to label the other cliques (they rely on Proposition 3 and Property 5). Finally, the case in which two negative cliques are adjacent is checked. If this is the case, it means that $G \notin \mathcal{L}_3^2$.

The correctness of the algorithm simply follows from the fact that triangulated graphs allow considering a tree of cliques and Algorithm 3 checks the cases in which a graph does not belong to $\mathcal{L}_3^2$. If this is not the case, the labelling is admissible.

Note also that, at the end of Algorithm 3 it is possible to have cliques that do not have a label. Since they were not considered before, if some vertices are not labelled we can fix their labels to either $+$ or $-$ in such a way that the labels alternate for the cliques c, c' that strongly intersect. Finally, we find a 3-uniform hypergraph H such that $G = L_3^2(H)$.

Construction of the 3-uniform hypergraph

Fact 9. Consider the labelled tree T of a claw-free triangulated hypergraph. Then there exists H such that $G = L_3^2(H)$.

Proof. Given T with the labelling of its vertices, we construct a labelling of the vertices of G from the root c_r to the leaves of T_r. We assume w.l.o.g. that $\lambda(c_r) = +$. We label the vertices of c_r as follows: since $\lambda(c_r) = +$, c_r it is labelled positively with the labels $(1, 2, 3)$, $(1, 2, 4)$,. . .,$(1, 2, k_r)$. Let $c \in C, c \neq c_r$. We assume that the vertices of its predecessor c' in T_r are labelled.

Since T is a tree, up to a permutation of the labels, we assume that the labels of c' are taken into $\{1, 2, \ldots, k\}$. Consider the following two cases:

- Suppose that c' is labelled positively with $(1, 2, 3), (1, 2, 4), \ldots, (1, 2, k)$. If c strongly intersect c' then $\lambda(c) = -$. We set $(1, 2, 3), (1, 2, 4)$ the labels of $c \cap c'$. If c contains three vertices, it is labelled as $(1, 3, 4)$. If it contains four vertices, the last node is labelled as $(2, 3, 4)$.

  If c weakly intersect c' from Fact 4 we have $|c| = 3$. Let $(1, 2, 3)$ be the label of $c \cap c'$. If $\lambda(c) = -$ then the other labels of c are $(1, 3, k), (2, 3, k)$, with k different from any value in c and c'. If $\lambda(c) = +$ then the other labels of c are $(1, 3, k), (1, 3, t)$, with k, t different from any value in c and c'. In any case, we obtain a valid label for the two cliques;

- Suppose that c' is labelled negatively. Let's consider the case in which $|c'| = 4$ and their vertices are $(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)$: from Fact 4 c strongly intersect c' and $\lambda(c) = +$ must hold. Let $(1, 2, 3), (1, 2, 4)$ be the labels of $c \cap c'$. The other labels of c are $(1, 2, 5), \ldots, (1, 2, k)$.

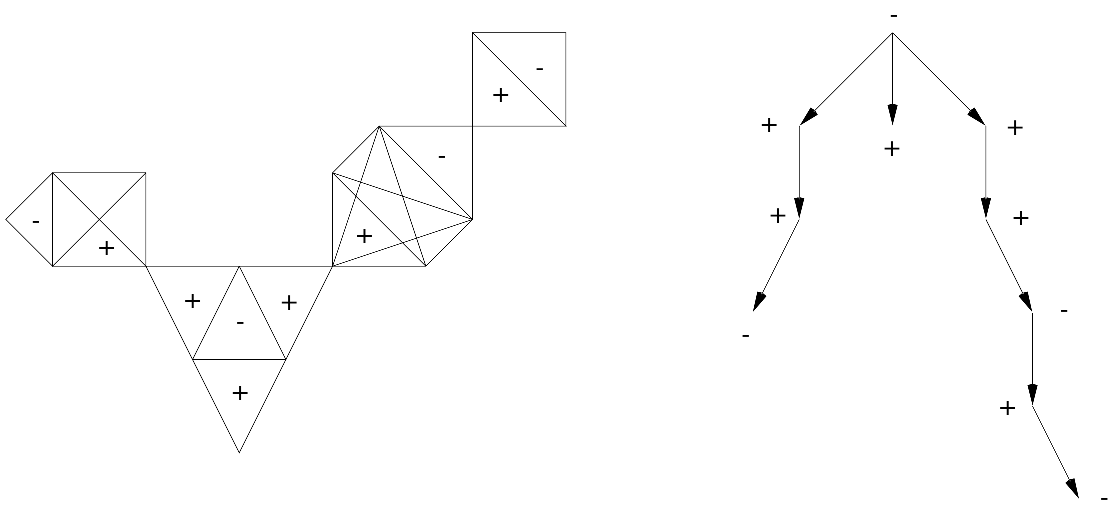


Figure 4.23: Labelling of the graph G and its associated tree.

Consider now the case in which $|c'| = 3$ and their vertices are $(1,2,3), (1,2,4), (1,3,4)$. If c strongly intersect c', let $(1,2,3), (1,2,4)$ be the labels of $c \cap c'$. The other labels of c are $(1,2,5), \dots, (1,2,k)$.

On the other hand, if c weakly intersects c', let $(1,2,3)$ be the label of $c \cap c'$. The other labels of c are $(2,3,5), \dots, (2,3,k)$.

In case c' has a second successor c'', such that the label of $c \cap c''$ is $(1,2,4)$, then the other labels of c'' are $(2,4,k+1), \dots, (2,4,k+l)$. Recall that $c' \neq c_r$ so c' has at most two successors in T_r. □

The proof is complete. □

As an example, consider the graph depicted G in Figure 4.21 and its associated tree cliques. Using the rules listed in the proof, we obtain the labelling shown in Figure 4.23. From that, it's easy to find an actual label of the vertices and conclude that $G \in L_3^2(H)$.

4.4 Further results

In this section, we prove some further results regarding graphs belonging to $\mathcal{L}_k^l$ classes, starting with a property valid for general graphs belonging to that class.

Fact 10. If $G \in \mathcal{L}_k^l$ then G is $K_{1,p+1}$-free, where $p = \binom{k}{l}$.

Proof. Suppose $G = L_k^l(H)$ and let $e \in V(G)$ a node, relative to an edge of H. Since H is a $k-$uniform hypergraph, it means e can be connected at most to $\binom{k}{l}$ other edges. □

4.4.1 NP-complete problems in the class L_k^1

Given a linear k-uniform hypergraph $H = (V, E)$ one can define a linear $(k+1)$-uniform hypergraph $H' = (V', E')$ as follows. First of all, we set $V' = V \cup_{1 \leq i \leq m} \{a_i\}$. Then, for each hyperedge $e_i \in E, 1 \leq i \leq m$, we create the hyperedge $e_i' \in E', e_i' = e_i \cup \{a_i\}$.

In [47] it is proved that recognizing whether a graph $G \in \mathcal{L}_3^1$ is NP-complete. Hence, using the previous construction, we obtain the following proposition.

Proposition 4. For any fixed $k \geq 3$, deciding whether a graph $G \in L_k^1$ is NP-complete.

Moreover, other works show the veracity of the following proposition.

Proposition 5. The problems Hamiltonian cycle [48], 3-coloring [49], Minimum domination [50] are NP-complete in L_2^1.

Using the previous construction it's easy to check that the following proposition is true.

Proposition 6. For any fixed $k \geq 2$, the problems Hamiltonian cycle, 3-coloring, and minimum domination are NP-complete in L_k^1.

4.4.2 Hamiltonian cycle detection in $\mathcal{L}_3^2$

In the previous chapter we study the null label problem and prove a sufficient condition for a $3-$hypergraph to be null. In particular, the result uses Hamiltonian graphs in $\mathcal{L}_3^2$. Here, we show that deciding if $G \in \mathcal{L}_3^2$ is Hamiltonian is NP-complete, limiting the possible application of the result.

Theorem 11. *The Hamiltonian cycle problem is NP-complete in $\mathcal{L}_3^2$ even for graphs $G = L_3^2(H)$ where $m(H) = 3$.*

Proof. We give a polynomial transformation from the Hamiltonian cycle problem in cubic graphs which is NP-complete [45]. From a cubic graph $G' = (V', E')$, we define $G \in \mathcal{L}_3^2$ as follows: to each vertex $v \in V'$ with neighbours u, w, and t corresponds K_v, i.e., the complete graph with the three vertices $(v, \overline{v}, u), (v, \overline{v}, w), (v, \overline{v}, t)$. For each edge $uv \in E'$, we add the edge $(v, \overline{v}, u)(u, \overline{u}, v)$. It is straightforward to verify that $G \in \mathcal{L}_3^2$, and that H, the hypergraph such that $G = L_3^2(H)$, satisfies $m(H) = 3$. Moreover, G' has a Hamiltonian cycle if and only if G has one. □

Remark 1. In [51] is proved that the Hamiltonian cycle problem remains NP-complete for cubic planar graphs. Since K_3, the subgraph replacing each vertex in our reduction is planar, it is straightforward, using the same transformation, that the Hamiltonian cycle problem is NP-complete in $\mathcal{L}_3^2$ even for planar graphs.

4.4.3 Recognition problem for trees

We are interested in the recognition problem for trees in $\mathcal{L}_3^2$. We denote with $\Delta(T)$ the maximum degree among all the vertices in the graph.

Proposition 7. Let T be a tree. $T \in \mathcal{L}_3^2$ if and only if $\Delta(T) \leq 3$.

Proof. Let T be a tree. If $\Delta(T) \geq 4$ then T contains $K_{1,4}$ as an induced subgraph and so $T \notin \mathcal{L}_3^2$. Now $\Delta(T) \leq 3$. We use induction on n, the number of vertices of T. The cases $n = 1$ and $n = 2$ are trivial. Let v be a leaf of T. By our induction hypothesis, $T - v$ has a λ_3^2-labelling. Let w be the neighbour of v in T. In $T - v$, w has degree at most two. Without loss of generality, let $w = \{1, 2, 3\}$ and $w' = \{1, 2, 4\}$ be the labelling of a neighbor w' of w in $T - e$. When w' is the unique neighbor of w in T_e then $v = \{2, 3, 5\}$. Else w'' is the second neighbour of w in T_e. Let $w'' = \{1, 3, 5\}$. Then $v = \{2, 3, 6\}$. $\square$

Chapter 5

Minimum Surgical Probing

In this chapter we consider a general class of problems that concern retrieving information and, at its best, reconstructing a physical discrete object from aggregate measurements on its points. In particular, we assume that each point of the unknown object has an assigned value and our aim is to determine these values. Such a framework models several situations where it is not easy or even not possible to obtain them through a precise inspection (called *surgical probe*), since it may damage the structure or may alter these values. A common alternative proposes the use of aggregate measuring techniques, whereby measurements are taken over a larger area or discrete lines and the values at each point are subsequently extracted by computational methods.

This problem is often referred to as the *Discrete Tomography Reconstruction* problem (DTR) (for a survey on the topic and the related problems see [21, 22]). The *Microscopic Image Reconstruction* problem (MIR) has been introduced in [34, 52] as a natural extension of DTR. In both problems, the object is represented by a subset $U \subseteq \mathbb{Z} \times \mathbb{Z}$ whose points have assigned non-negative integer values that can be thought as their weights. In the DTR problem, the projections are taken using an entire row or column. In contrast, in the MIR problem it is assumed that the microscope's scanning window is a subset of the plane (see [34, 53] for some examples). In [25] the authors extend this framework considering a generalized setting where the inspected object is represented by an undirected unweighted connected graph $G = (V, E)$. In this case, the vector $\ell \in \mathbb{R}^n$ is an assignment of values ℓ_v to each one of the n nodes in V. Moreover, in this context, a probe centred in v captures the sum of its neighbours' labels. Since in general there may exist several values assignments that satisfy the same vector ℓ, the authors of [25] studied the so-called *Minimum Surgical Probing* problem (MSP) in which we ask to find the minimum number of surgical probes (i.e. exact known values) that allows reconstructing ℓ uniquely. In the same paper the authors show that the problem can be solved in polynomial time using linear algebra tools. Also, a generalization for weighted graphs has been studied [54].

Moving from these studies, in the first section we recall MSP and some basic results about it. Then, in the second section, we study MSP on hypergraphs, proving the equivalence with the same problem on weighted graphs. We also consider some classes of hypergraphs whose $2-$intersection graphs have a specific form investigating the MSP problem of the related hypergraphs. We prove that the values assigned to the hypergraphs' nodes can be retrieved (in polynomial time) by using zero or one surgical probes, and we show how to detect them.

In the third section we step back to graphs, searching for convex solutions to the problem. In particular, although the problem is $NP-$hard in general, we provide some subclasses of graphs for which it is possible to obtain simple and elegant algorithms to reconstruct their labels and find the minimum number of surgical probes needed to do that.

Part of the results presented in this chapter are published in [55] and [56].

5.1 Definition of MSP and previous results

Consider a hypergraph $H = (V, E)$ such that $|V| = n$. For each $v \in V$ we assign to it a label $\ell_v \in \mathbb{R}$. We denote $\ell = (\ell_1, \ldots, \ell_n)$ as *label* vector of the nodes. For a node v we define its *probe* as

$$\mathcal{P}_v = \sum_{w \in V} F(v, w)\ell_w \tag{5.1}$$

where $F(v, w)$ is the *number of hyperedges that contain both v and w*. The value of $F(v, v)$ (either 0 or different) determines if we are considering exclusive or inclusive probes, i.e. if we are considering the node itself inside the probe. Suppose to collect the probes in a vector $\mathcal{P}$. We stress that the probe of a node v considers each neighbour's label with a coefficient counting its occurrences in different hyperedges (provided by $F(\cdot, \cdot)$). On the other hand, the choice of computing the probes of v without such coefficients allows the MSP problem to go back to the graph case. In fact, for a generic unweighted graph G, $F(v, w) = A_{v,w}$ where A is the adjacency matrix of G.

We define the *neighbourhood matrix* F of a hypergraph as the matrix whose generic entry $f_{i,j}$ equals $F(i, j)$. The first question that comes from the previous definition is the following.

Question 1. Is it possible to compute vector ℓ from the knowledge of $\mathcal{P}$?

Since in general there may exist several vectors ℓ that would yield the same probe $\mathcal{P}$ (see [25] for an example), we study a generalization of the problem described in [25]. Let us define *surgical probe* at node v to be the knowledge of its label l_v:

Question 2. What is the minimal number of surgical probes needed for a unique reconstruction of the label vector ℓ from the knowledge of $\mathcal{P}$?

We call this problem *Minimum Surgical Probing* problem (MSP). Given a hypergraph H and a vector $\mathcal{P}$, we aim at finding the minimum number of surgical probes needed to reconstruct the vector ℓ uniquely.

In [25], the authors prove that the MSP problem on graphs can be solved in polynomial time and present an efficient algorithm to perform the task.

In particular, solving Question 2 concerns the study of the solutions of the linear system:

$$F\ell = \mathcal{P}. \tag{5.2}$$

In the following, we denote with $rank(A)$ its rank and with $\phi_\lambda(A)$ the geometric multiplicity of each $\lambda \in \Lambda(A)$, where $\Lambda(A)$ is the set of the eigenvalues of A.

We note that F is a non-negative $n \times n$ symmetric matrix, and therefore $\phi_0(F) = n - rank(F)$ is the kernel dimension of F.

Theorem 12. *Let us consider a hypergraph H and let $\mathcal{P}$ be its probes vector and F be its neighbourhood matrix. It holds that*

1. *if F has full rank, then ℓ can be found in polynomial time without surgical probes;*
2. *otherwise the minimum number of surgical probes needed to compute ℓ is $s = n - rank(F) = \phi_0(F)$.*

The proof of Theorem 12 can be easily obtained from Theorem 2.2 in [25]. We stress that its proof only relies on the symmetry of the matrix, without imposing any constraint on its coefficients.

Observation 1. The minimum number of probes pointed out in Theorem 12 holds for a generic vector of labels ℓ, while it can be considered as a lower bound in case we require ℓ to be an integer vector.

Observation 2. In the framework we are setting up, we implicitly assume that the probe vector $\mathcal{P}$ is obtained by the machinery scanning of a real object. This implies that system (5.2) has always at least one solution. However, from a mathematical point of view, it may happen that $rank(F) < |V|$, preventing the problem from having a solution.

5.2 The MSP problem on classes of hypergraphs

In this section we suppose $F(v, v) = 0$ for each $v \in V$, i.e. the node itself is not included in the probe. Consider the following notion.

Definition 7. Let $H = (V, E)$ be a hypergraph, we define the graph $G_H = (V_H, E_H)$ such that $V_H = V$ and $E_H = \{\{v, w\} : \ exists\ e \in E\ such\ that\ \{v, w\} \subseteq e\}$. Furthermore, we define a *weight function* W_H on E_H such that $W_H(v, w) = F(v, w)$, being F the neighborhood matrix of H. G_H is called *weighted* $2-$*section* of H.

In words, starting from H we compute the graph G_H by replacing each hyperedge $e \in E$ with a complete graph on the same set of nodes of e. The weight function $W(\cdot, \cdot)$ indicates, for each edge (v, w) of G_H, the number of hyperedges of H including the two nodes v and w (see Figure 5.1).

This notion has been introduced in [57] and later studied in [58], where the authors consider the reconstruction problem of a hypergraph starting from its weighted $2-$section.

It is worthwhile noting that the MSP problem on hypergraphs H can be equivalently shifted to the same problem on the weighted graph G_H, where the probes are computed using the edges' weights so that F turns out to be the weighted adjacency matrix of G_H.

Example 8. Consider the hypergraph H in Figure 5.1 (a), whose hyperedges are $\{\{1, 2, 3\}\{2, 3, 4\}\}$. Its probes are $P_1 = l_2 + l_3$, $P_2 = l_1 + 2l_3 + l_4$, $P_3 = l_1 + 2l_2 + l_4$, and $P_4 = l_2 + l_3$.

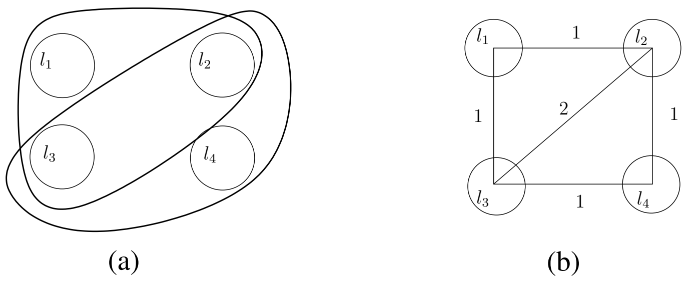


Figure 5.1: The hypergraph $H = \{\{1, 2, 3\}\{2, 3, 4\}\}$ is shown in (a). (b) represents the related G_H weighted graph.

Consider the $2-$section graph G_H depicted in Figure 5.1, (b). An easy check reveals that G_H satisfies the same probes $P_1 \ldots P_4$ as H.

So far the intersection graph has been a valuable tool to inspect structural properties of uniform hypergraphs, such as the existence of null labellings. Therefore, in what follows, we use the intersection graph to solve the MSP problem for some relevant classes of hypergraphs.

Observation 3. Let H be a hypergraph. If its 2-intersection graph $L_3^2(H)$ has no edges, then no two hyperedges of H share two nodes. So, it holds that the edges' weights of G_H have the same value 1, and the solution of the related MSP problem (on unweighted graph) has already been studied in [25].

Here, we focus on three classes of $3-$hypergraphs whose $2-$intersection graphs have specific, non-trivial, properties of regularity.

5.2.1 3-hypergraphs whose 2-intersection graph is a line

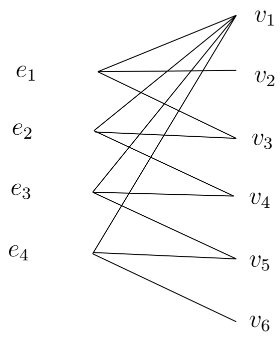
Let us consider a 3-hypergraph H whose $L_3^2(H)$ intersection graph is a line. Among them, we distinguish two types of hypergraphs: the *cluster hypergraphs* and the *path hypergraphs*. We begin our study from the former.

The MSP problem on cluster hypergraphs

Definition 8. A $k-$cluster hypergraph is a $3-$hypergraph such that $|V| = k+2$, $|E| = k$ and its hyperedges are defined, up to isomorphism, as

$$\begin{cases} e_1 = \{1,2,3\} \\ e_i = \{1, i+1, i+2\} \quad i = 2 \dots k. \end{cases} \tag{5.3}$$

An easy check reveals that the $2-$intersection graph of a $k-$cluster is a line of length k. Figure 5.2 depicts a $4-$cluster hypergraph (left), and its four-length line $2-$intersection graph (right).



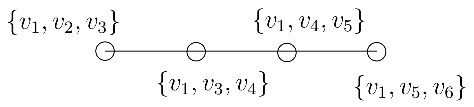


Figure 5.2: A 4-cluster hypergraph (left) and its 2-intersection (right)

Denote by F_k the neighborhood matrix of a $k-$cluster. Note that if $|E| = k$, then $|V| = k+2$.

In Table 5.1, we explicitly compute the neighborhood matrices of the $2-$cluster and $3-$cluster hypergraphs $H_2 = \{\{1,2,3\},\{1,3,4\}\}$ and $H_3 = \{\{1,2,3\},\{1,3,4\}\{1,4,5\}\}$.

The elements that are in common are in boldface, according to the successive Lemma 5.4 that highlights the relation between the generic matrices F_{n-1} and F_n.

Lemma 12. *Let F_{n-1} and F_n be two neighborhood matrices of the $(n-1)$-cluster and n-cluster, respectively. The following recursive equations define F_{n+1}*

$$F_2 = \begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{2} & 1 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 \\ \mathbf{2} & \mathbf{1} & \mathbf{0} & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad F_3 = \begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{2} & 2 & 1 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 \\ \mathbf{2} & \mathbf{1} & \mathbf{0} & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Table 5.1: the neighborhood matrices associated to the clusters of dimensions 2 and 3.

$$F_{n+1}(i,j) = F_n(i,j) \text{ \textit{if} } 1 \le i \le n+1 \text{ \textit{and} } 1 \le j \le n+1 \tag{5.4}$$

$$F_{n+1}(i,n+2) = F_{n+1}(n+2,i) = \begin{cases} 2 \text{ \textit{if} } i = 1 \\ 1 \text{ \textit{if} } i = n+1 \\ 0 \; otherwise \end{cases} \tag{5.5}$$

$$F_{n+1}(i,n+3) = F_{n+1}(n+3,i) = \begin{cases} 1 \text{ \textit{if} } i = 1 \text{ \textit{or} } i = n+2 \\ 0 \; otherwise. \end{cases} \tag{5.6}$$

Proof. Equation (5.4) states that the multiplicity of the edges $\{i,j\}$ of F_n is not affected by the addition of the new node.

Equation (5.5) states that adding node $n+3$ implies, by definition, the inclusion of the hyperedge $\{1, n+2, n+3\}$. Therefore, $\{1, n+2\}$ shares two hyperedges and $\{n+2, n+1\}$ shares one hyperedge.

Equation (5.6) set the values of the row and the column of the new node $n+3$. Since the new hyperedge is $\{1, n+2, n+3\}$ the equation holds. □

From Theorem 12, it follows that the MSP problem on cluster hypergraphs can be solved by studying the rank of the associated F_n matrices.

Theorem 13. *Let $n \ge 4$ and consider $n-$cluster hypergraph. If n is even, then the MSP problem can be solved with one surgical probes while, if n is odd, then no surgical probes are needed.*

Proof. Let us study $rank(F_{n-2})$ by inspecting its associated homogeneous system:

$$\begin{cases} x_2 + 2x_3 + \ldots + 2x_{n-1} + x_n = 0 \\ x_1 + x_3 = 0 \\ 2x_1 + x_{i-1} + x_{i+1} = 0 \text{ if } i = 3 \ldots n-1 \\ x_1 + x_{n-1} = 0. \end{cases}$$

From the second equation we get $x_3 = -x_1$, and from the last one, $x_{n-1} = -x_1$. By substituting the computed variables in the third equation, with i ranging from 3 to $n-1$, we obtain, for $i = 4$, the equation $x_5 = -x_1$ and, in general, the equation $x_i = -x_1$, with i odd. On the other hand, $x_{n-1} = -x_1$ implies $x_{n-3} = -x_1$ when $i = n-2$ and, in general $x_{(n-1-2k)} = -x_1$. We have two cases:

1. if n is odd we obtain each $x_i = -x_1$. Since the first equation is a sum of all the variables except x_1, we obtain $x_1 = 0$ and therefore $x_i = 0$. So F_{n-2} has maximum rank;

2. if n is even we obtain $x_i = -x_1$, with i odd, and then we have an additional equation that gives $0 = 0$ (i.e., x_1 is a free variable). Note that other equations involving even index i are of the form

$$x_{i-2} + x_i = -x_1$$

Therefore, assuming $i = 2k$, we obtain x_{2k} in terms of x_1 and x_2, so

$$x_{2k} = -x_1 - x_{2k-2} = \begin{cases} x_2 & \text{if } k \equiv_2 0 \\ -x_1 - x_2 & \text{if } k \equiv_2 1 \end{cases} \tag{5.7}$$

Since the first equation is only a sum of all the other variables except x_1, $rank(F_{n-2}) = n-1$. Therefore, from Theorem 12, we need one surgical probe to determine ℓ.

□

The MSP problem on path hypergraphs

Definition 9. A $n-$path hypergraph C_n, with $n \geq 1$, is a $3-$hypergraph H in which $|V| = n+2$, $|E| = n$ and whose hyperedges are

$$\begin{cases} e_1 = \{1,2,3\} \\ e_i = \{i, i+1, i+2\} \quad i = 2, \dots n \end{cases} \tag{5.8}$$

Also in this case, w.l.g. we suppose that v_1 is the starting node. See Figure 5.3 for an example.

Call F_n the neighbourhood matrix of a $n-$path. The following lemma holds.

Lemma 13. *Let H be C_n hypergraph. Then F_n is a symmetric pentadiagonal matrix of dimension $(n+2) \times (n+2)$ such that:*

$$\begin{cases} diag(F_n) = (0, \dots, 0) \in \mathbb{R}^{n+2} \\ diag_{\pm 1}(F_n) = (1, 2, 2, \dots, 2, 2, 1) \in \mathbb{R}^{n+1} \\ diag_{\pm 2}(F_n) = (1, \dots, 1) \in \mathbb{R}^{n} \end{cases} \tag{5.9}$$

where $diag_{\pm k}(F_n)$ indicates the k-th diagonals of F_n above and below the main diagonal.

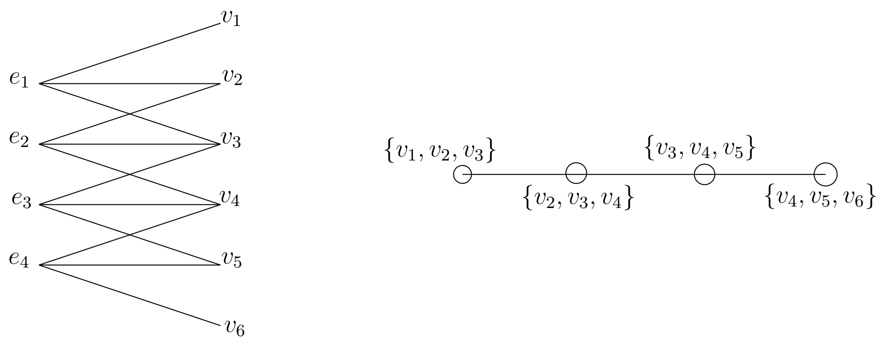


Figure 5.3: (a) a $4-$path hypergraph and in (b) its $2-$intersection graph.

$$F_2 = \begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{2} & 1 \\ \mathbf{1} & \mathbf{2} & \mathbf{0} & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \qquad F_3 = \begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{2} & 1 & 0 \\ \mathbf{1} & \mathbf{2} & \mathbf{0} & 2 & 1 \\ 0 & 1 & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Table 5.2: the neighborhood matrices associated to the paths of dimensions 4 and 5.

Proof. The first equation immediately follows by definition of neighborhood matrix. Then, we note that each node i different from 1 and $n+2$ is connected twice with $i+1$ and $i-1$. Moreover, nodes 1 and $n+2$ are connected once with node 2 by the hyperedge $\{1, 2, 3\}$, and with $n+2$ by the hyperedge $\{n, n+1, n+2\}$. Therefore the second equation holds. Finally, each node i is connected only once with $i+2$ and $i-2$, obtaining the third equation. $\square$

In Table 5.2, we explicitly compute the neighborhood matrices F_4 and F_5 of the $4-$path and $5-$path hypergraphs, respectively. Again the sub-matrices with equal elements are in boldface.

The following result holds.

Theorem 14. *Consider a $n-$path. The MSP problem can be solved without surgical probes if and only if $n \geq 3$. In particular, if $n = 2$, a surgical probe is needed.*

Proof. The case $n = 2$ is obtained after noticing that $rank(F_2) = 3$, so one surgical probe is needed to recover **l**. Consider a generic F_n matrix, with $n \geq 3$. By Theorem 12 the statement is proved if $rank(F_n) = n + 2$. Following the same strategy as in the proof of Theorem 5.2.1, let us inspect the homogeneous linear system associated with F_{n-2}:

$$\begin{cases} x_2 + x_3 = 0 \\ x_1 + 2x_3 + x_4 = 0 \\ x_{i-2} + 2x_{i-1} + 2x_{i+1} + x_{i+2} = 0 \quad i = 3 \dots n-2 \\ x_{n-3} + 2x_{n-2} + x_n = 0 \\ x_{n-2} + x_{n-1} = 0 \end{cases} \tag{5.10}$$

It follows:

$$\begin{cases} x_3 = -x_2 \\ x_4 = -x_1 + 2x_2 \\ x_{i+2} = -x_{i-2} - 2x_{i-1} - 2x_{i+1} \quad i = 3, \dots n-2 \\ x_n = -x_{n-3} - 2x_{n-2} \\ x_{n-2} = -x_{n-1} \end{cases} \tag{5.11}$$

The previous equations show that any variable can be expressed by a linear combination of x_1 and x_2. In the equation defining x_i, we indicate α_i the coefficient of x_1 and β_i the coefficient of x_2.

Furthermore, the following initial conditions hold:

$$\begin{cases} \alpha_1 = 1 \\ \alpha_2 = 0 \\ \alpha_3 = 0 \\ \alpha_4 = -1 \end{cases} \quad \begin{cases} \beta_1 = 0 \\ \beta_2 = 1 \\ \beta_3 = -1 \\ \beta_4 = 2 \end{cases} \tag{5.12}$$

The next equations provide the recursive description of α_i and β_i for a generic index i:

$$\begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} = \begin{pmatrix} -\alpha_{i-4} - 2\alpha_{i-3} - 2\alpha_{i-1} \\ -\beta_{i-4} - 2\beta_{i-3} - 2\beta_{i-1} \end{pmatrix}. \tag{5.13}$$

Note that the two coefficients follow the same equation with different initial conditions. Let us consider α_n (we can act similarly when considering β_n); using basic tools of finite differences equations' theory we obtain that (5.13) has the following general solution:

$$\alpha_n = \sum_{k=1}^{4} c_k z_k^{n-1}, \; n = 1, 2, \dots \tag{5.14}$$

where c_k is a constant determined by the initial conditions and z_k is a root of the characteristic polynomial $z^4 + 2z^3 + 2z + 1 = 0$. Moreover, the solutions $\{z_k^n\}_{i=1}^4$ are linearly independent. In particular, we have

$$z_1 = \frac{1}{2}(-1 - \sqrt{2}\sqrt[4]{3} - \sqrt{3}), \qquad z_2 = \frac{1}{2}(-1 + \sqrt{2}\sqrt[4]{3} - \sqrt{3}),$$

$$z_3 = -\frac{1}{2} - \frac{i\sqrt[4]{3}}{\sqrt{2}} + \frac{\sqrt{3}}{2}, \qquad z_4 = -\frac{1}{2} + \frac{i\sqrt[4]{3}}{\sqrt{2}} + \frac{\sqrt{3}}{2}.$$

Note that $|z_3| = |z_4| = 1$, $|z_2| > 1$ and $|z_1| < 1$. So, (5.14) is the general expression of the α_i succession values and we use it in (5.11). In particular, from its last two equations, we obtain:

$$\begin{cases} 0 = x_{n-2} + x_{n-1} = x_1(\alpha_{n-2} + \alpha_{n-1}) + x_2(\beta_{n-2} + \beta_{n-1}) \\ 0 = x_{n-3} + 2x_{n-2} + x_n = x_1(\alpha_{n-3} + 4\alpha_{n-1} + \alpha_{n-4}) + x_2(\beta_{n-3} + 4\beta_{n-1} + \beta_{n-4}) \end{cases} \tag{5.15}$$

These equations have a unique solution $x_1 = x_2 = 0$ since $\alpha_{n-2}+\alpha_{n-1} = \sum_{i=1}^{4} c_i(1+z_i)z_i^{n-3}$, i.e. it is a linear combination of the solutions $\{z_i\}_{i=1\ldots4}$. Since they are linearly independent, then it holds $\alpha_{n-2}+\alpha_{n-1} \neq 0$, with $n \geq 5$. The same holds for $\beta_{n-1}+\beta_{n-2}$.

Therefore, from the first equation of (5.15) we obtain $x_2 = x_1\frac{(\alpha_{n-2}+\alpha_{n-1})}{(\beta_{n-2}+\beta_{n-1})}$. Substituting the value of x_2 in the second equation of (5.15), we finally obtain

$$x_1 \underbrace{\left(\alpha_n + 2\alpha_{n-2} + \alpha_{n-3} + \frac{\alpha_{n-1} + \alpha_{n-2}}{\beta_{n-1} + \beta_{n-2}}(\beta_n + 2\beta_{n-2} + \beta_{n-3})\right)}_{=c_n} = 0 \tag{5.16}$$

The proof ends if $c_n \neq 0$. Since $|z_2|$ is the only root greater than 1, then its exponential behaviour overwhelms the other terms. In particular, the computation of the values of c_n that we provide up to $n = 100$, shows that the condition is verified. As a consequence, the only solution of (5.16) is $x = 0$ obtaining that $rank(F_n) = n + 2$.

□

As previously observed, both for cluster and path hypergraphs the $2-$intersection graph is a line. Conversely, a hypergraph that has a line $2-$intersection graph can contain both clusters and paths at the same time. In such cases, it becomes quite hard to find a general expression of the related neighborhood matrices.

5.2.2 3-hypergraphs whose 2-intersection graph is a tree

Consider a $3-$hypergraph T_s whose $2-$intersection graph $L_3^2(T_s)$ is a perfectly height-balanced tree of height s and each node, but for the leaves, has degree 3. We indicate it as s-*tree*. Suppose that the edge e of T_s appears (as node) in $L_3^2(T_s)$ at level r. Then we say that e has *level* $p(e) = r$; if e is a leaf, then we call it *leaf hyperedge*, and it holds $p(e) = s$. The set of all leaf hyperedges is indicated with L.

Consider an internal node $e = \{x, y, z\}$ in $L_3^2(T_s)$; its children are $\{x, z, k_1\}$, $\{y, z, k_2\}$. Up to renaming of the nodes of T_s, we suppose that k_1 and k_2 appear only in the hyperedges that are nodes of the subtree of $L_3^2(T_s)$ having root e. So, at each level of $L_3^2(T_s)$, we have a sequence of nodes related to hyperedges of T_s of the form

$$\{x_1, z_1, k_1\}, \{y_1, z_1, k_2\}, \ldots, \{x_n, z_n, k_{2n-1}\}, \{y_n, z_n, k_{2n}\}.$$

An order can be set on them according to the $k_1, \ldots, k_{2n}$ numbering. Furthermore, at level s of the s-tree, the new nodes $k_1, \ldots k_{2n}$ of the leaf hyperedges are indicated as *leaf nodes*.

Observation 4. Since each internal node $e = \{x, y, z\}$ of $L_3^2(T_s)$ has degree 3 and T_s is a perfectly height-balanced tree, e uses all the three couples $\{x, y\}$, $\{y, z\}$, $\{x, z\}$ to be connected to its neighbourhoods nodes. On the other hand, if e is a leaf node, then it is connected with one only couple to its neighbours.

We assume w.l.g. that the root of $L_3^2(T_s)$ is the hyperedge $\{1, 2, 3\}$ of T_s. So, by definition, it follows that there exists a unique $s-$tree hypergraph for each $s \in \mathbb{N}$ (up to isomorphism). Figure 5.4 shows the s-tree T_2.

Lemma 14. *Let $s \geq 2$. The following statements hold:*

1. *given a $s-$tree hypergraph T_s, the corresponding $L_3^2(T_s)$ has $m = 3 \cdot 2^{s-2}$ leaves;*

2. *for each node k of T_s, if k is not a leaf node, then there exist exactly two paths in $L_3^2(T_s)$ that connect the first occurrence of k in a node to two corresponding leaves. Furthermore, all the nodes in these two paths contain the node k;*

3. *an $s-$tree hypergraph T_s has $6 \cdot 2^{s-2}$ nodes.*

Proof. 1. Let $s = 2$ and consider T_2. Without loss of generality, we have

$$T_2 = \{\{1, 2, 3\}, \{1, 2, 4\}, \{2, 3, 5\}, \{1, 3, 6\}\}$$

(see Figure 5.4). Note that it has $3 = 3 \cdot 2^{2-2}$ leaves. Consider a general T_{s+1} and suppose the thesis is true for T_s. Since every leaf of T_s generates two new leaves, T_{s+1} has $2(3 \cdot 2^{s-2}) = 3 \cdot 2^{s-1}$ leaves.

2. let z be a node of T_s and $e = \{x, y, z\}$ (one of its hyperedges), the first internal node of $L_3^2(T_s)$ in which z appears. By definition of s-tree its two children are $\{x, z, k_1\}$ and $\{y, z, k_2\}$. Note that only one child of each successive node will contain z in its relative edge until reaching the leaves.

3. We proceed by a simple induction. Basis: the T_2 hypergraph has $6 \cdot 2^{2-2} = 6$ nodes. Suppose that the thesis holds for T_s and let us compute the number of nodes of the successive s-tree T_{s+1}. When we add a new level, each leaf of the $s-$tree hypergraph generates two children and therefore two new nodes. Therefore it holds that the nodes of T_{s+1} double those of T_s, so $2(6 \cdot 2^{s-2}) = 6 \cdot 2^{s-1}$.

$\square$

Example 9. The T_2 s-tree of Figure 5.4 has $\{1, 2, 4\}$, $\{2, 3, 5\}$, $\{1, 3, 6\}$ leaf hyperedges, and 4, 5, and 6 are the leaf nodes.

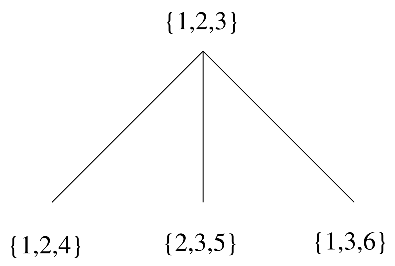


Figure 5.4: The figure shows $L_3^2(T_2)$ hypergraph. We have $L = \{\{1, 2, 4\}, \{2, 3, 5\}, \{1, 3, 6\}\}$. Moreover, nodes 4,5,6 are leaf nodes.

Moreover, the neighborhood matrix of T_2 is F_2, i.e., the following 6×6 square matrix

$$F_2 = \begin{pmatrix} 0 & 2 & 2 & \mathit{1} & \mathit{0} & \mathit{1} \\ 2 & 0 & 2 & \mathit{1} & \mathit{1} & \mathit{0} \\ 2 & 2 & 0 & \mathit{0} & \mathit{1} & \mathit{1} \\ \mathit{1} & \mathit{1} & \mathit{0} & 0 & 0 & 0 \\ \mathit{0} & \mathit{1} & \mathit{1} & 0 & 0 & 0 \\ \mathit{1} & \mathit{0} & \mathit{1} & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} A_2 & B^t \\ B & \overline{0} \end{pmatrix} \tag{5.17}$$

Note that the neighbourhood matrix F_2 of Example 5.2.2 can be decomposed into four matrices of dimension 3×3. An easy check reveals that the boldface entries refer to the matrix A_2 that contains the values of the internal nodes' links of $L_3^2(T_2)$. The italic entries refer to the matrices B and B^t that contain the connections between the internal nodes and the leaves of T_2. The fourth matrix is the zero matrix $\overline{0}$.

Let us consider the linear system associated to F_2. From the last three rows related to the matrix B we get

$$\begin{cases} x_1 = -x_2 \\ x_1 = x_3 \\ 2x_1 = 0. \end{cases} \tag{5.18}$$

Therefore we obtain $x_1 = x_2 = x_3 = 0$ and $rank(B) = 3$. Since $rank(B^t) = rank(B) = 3$ we obtain also $x_4 = x_5 = x_6 = 0$. Therefore F_2 has maximum rank and the MSP problem on T_2 can be solved without surgical probes.

The previous example shows a general property of the neighbourhood matrix of a T_s hypergraph. We underline that an immediate consequence of Lemma 14 is that the dimensions of F_s, for a generic t-tree hypergraph T_s, is always even.

Lemma 15. *Let T_s be a s-tree hypergraph with $s \geq 3$ and let $n = 6 \cdot 2^{s-2}$. Then F_s can be decomposed into four $\frac{n}{2} \times \frac{n}{2}$ square matrices:*

$$F_s = \begin{pmatrix} A_s & B^t \\ B & \overline{0} \end{pmatrix} \tag{5.19}$$

Moreover, the entries of A_s can be characterized as follows:

$$A_s(i,j) = \begin{cases} 2 & \textit{if } F_{s-1}(i,j) \neq 0 \\ 0 & otherwise. \end{cases}$$

The matrix B (and its transpose B^t) contains exactly two non-zero elements in each row and in each column. In particular

$$B_{ij} = 1 \textit{ if and only if there exists } k > \frac{n}{2} \textit{ such that } (i,j,k) \textit{ is an edge of } T_s.$$

Finally, $\overline{0}$ is the null matrix.

Proof. Let us focus on the matrix A_s: we note that Lemma 14 assures that A_s has the same dimension of F_{s-1}. By Observation 5.2.2, and since A_s contains the connections between the first $\frac{n}{2}$ (i.e. non-leaf) nodes, then it has the same non-zero elements of F_{s-1}. Furthermore, since $L_3^2(T_s)$ has one layer more than $L_3^2(T_{s-1})$, i.e., the last one, then each 1 entry of F_{s-1} changes into a 2 entry of A_s in the same position.

Concerning matrix B, its characterization follows from the definition of neighbourhood matrix. Note that B represents the last $\frac{n}{2}$ (i.e. leaf) nodes of T_s. By Observation 5.2.2, every row of B has exactly two 1 entries and no two rows can be equal.

Moreover, Lemma 14 states that every non-leaf node appears in exactly two leaf hyperedges, therefore every column contains exactly two 1 entries and again all the columns are different.

Finally, the leaf nodes are not connected so the bottom-right part of matrix F_s is null.

□

Lemma 16. *Let us consider the $2 \times \frac{n}{2}$ matrix π_B whose generic elements in positions $(1,i)$ and $(2,i)$, with $1 \leq i \leq \frac{n}{2}$, are the row indexes of the two only first and second non-zero entries in column i of B (as defined in Lemma 15). It holds that matrix π_B is a permutation forming one single cycle of maximal length $\frac{n}{2}$.*

The proof is a direct consequence of Point 2 in Lemma 14 and Lemma 15.

Theorem 15. *Let $s \geq 3$. The MSP problem on T_s can be solved with one surgical probe.*

Proof. Let F_s be the neighbourhood matrix associated with T_s and decomposed according to Lemma 15. The linear system associate to the matrix B, considering the property states in Lemma 16, turns out to be, for each $j = 1 \dots \frac{n}{2}$:

$$\begin{cases} x_j = -x_1 & \text{if } j \text{ is a leaf node of } T_{s-1} \\ x_j = x_1 & \text{otherwise.} \end{cases} \tag{5.20}$$

By Lemma 14, B has an even number of rows when $s \geq 3$, so the last equation of this system is the identity $0 = 0$, leading to $rank(B) = \frac{n}{2} - 1$. So, each variable $x_2, \dots, x_j$ is expressed in terms of x_1 and it can be substituted in the variables of A_s obtaining $\mathbf{v}\, x_1$, with $\mathbf{v}$ being the integer column vector of length $\frac{n}{2}$ of the x_1 coefficients in A_s. The variable column $\mathbf{v}\ x_1$ is then concatenated with the part of the linear system related to B^t obtaining:

$$\begin{pmatrix} \mathbf{v} & B^t \end{pmatrix} \mathbf{x} = 0. \tag{5.21}$$

Note that, since A_s has the same form of F_{s-1}, we have $\mathbf{v} \neq \mathbf{0}$ (the rows of the leaf nodes of T_{s-1} do not sum to 0 since all the associated variables are equal to $-x_1$). With a similar procedure, we obtain that $rank(F_s) = n - 1$. So, by Theorem 12 it follows that one only surgical probe is required to solve the MSP related problem. □

5.3 Convex Minimum Surgical Probing

In this section, we take a step back to graphs and consider, on them, a binary label vector ℓ and MSP with convexity constraints.

The probe vector $\mathcal{P}$ is determined by Equation (5.1) adapted to graph case, i.e.

$$\mathcal{P}_i = \sum_{j \in N[j]} A_{ij} \ell_j, \tag{5.22}$$

where A is the adjacency matrix of G. In particular, we consider close neighbourhood, i.e. $i \in N[i]$. We define the support of vector ℓ as the number of its non-zero elements. Obviously, we always assume that ℓ has support at least 1.

In the sequel, we will use the terms *convex*, shorthand, for g-convex, m-convex, or t-convex (see the definitions in Chapter 1). Moreover, if $\ell_v = 1$ we say that v is a $1-$vertex and a $0-$vertex otherwise.

So, we say that a label vector ℓ is *convex* if

$$\mathbf{1}_G^\ell := \{i \in V \mid \ell_i = 1\}$$

is a convex subset of V in G, according to one of the above definitions. We consider the following definition of the problem.

Definition 10 (Convex Minimum Surgical Probing (CMSP))**.** Given a connected graph G and a probe vector $\mathcal{P}$, determine the minimum number of surgical probes required to uniquely uncover a label vector ℓ that is convex on G.

If a graph is not connected, we consider its components as independent instances of CMSP. Our first observation shows that the labels of some vertices can be uncovered readily. It also holds for non-convex label vectors.

Observation 5. If $\mathcal{P}_v = 0$, then $\ell_u = 0$, for all $u \in N[v]$.

Path convexities imply that each 1-vertex necessarily has another 1-vertex in its neighbourhood (if the label vector has support 2 or more). This observation may identify additional 0-vertices. We formalize it in the following lemma.

Lemma 17. *Let $G = (V, E)$ be a connected graph with a convex label vector ℓ of support at least 2, and let $\mathcal{P}$ be the corresponding probe vector. For $v \in V$, we have that if $\mathcal{P}_v \leq 1$, then $\ell_v = 0$*

Proof. Let G, ℓ, and $\mathcal{P}$ be as in the lemma. Let $v \in V$ such that $\mathcal{P}_v \leq 1$. Towards a contradiction, suppose that $\ell_v = 1$.

Since ℓ has support at least 2, there is another vertex $u \in V$ such that $\ell_u = 1$. If v and u are adjacent, it follows that $\mathcal{P}_v \geq 2$. Otherwise, v and u are not adjacent, and since G is connected, there is a vertex $w \in I_g(u, v)$ (resp. $I_m(u, v)$ and $I_t(u, v)$) that is adjacent to v and $\ell_w = 1$. It follows that $\mathcal{P}_v \geq 2$. We reach the desired contradiction, and $\ell_v = 0$ holds. $\square$

We treat label vectors with support 1 as a special case since the notion of convexity is trivial in this case. We show that they can be recognized efficiently.

To close this section, we define the *core* $C_G = \bigcap_{i \in V} N_G[i]$, being G a graph, as the subset of vertices that are adjacent to all vertices. Obviously C_G is a clique in G.

Lemma 18. *Let $G = (V, E)$ be a k-connected graph with a convex label vector ℓ of support at least $k + 1$, and let $\mathcal{P}$ be the corresponding probe vector. For $v \in V$, if $\ell_v = 1$, then $\mathcal{P}_v \geq k + 1$.*

Algorithm 4 $Reduce(G, \mathcal{P}, \ell_i, i)$

```
$\mathcal{P}' \leftarrow \mathcal{P} \setminus \mathcal{P}_i$
$G' \leftarrow G \setminus i$
for $j \in N_G(i)$ do
    $\mathcal{P}'_j \leftarrow \mathcal{P}'_j - \ell_i$
end for
return $G', \mathcal{P}'$
```

5.3.1 Reduction Algorithm

In this section, we describe Algorithm 4 which removes vertices with known labels from graph G and *updates* a consistent probe vector $\mathcal{P}$.

Assume that ℓ_i is known, for $i \in V$. Let G' and $\mathcal{P}'$ be the result of running $Reduce(G, \mathcal{P}, \ell, i)$. We denote the result of removing ℓ_i from ℓ by ℓ^{-i}. The next observation follows readily.

Observation 6. Probe vector $\mathcal{P}$ is consistent with G and ℓ if and only if $\mathcal{P}'$ is consistent with G' and ℓ^{-i}.

One needs to be careful when applying Algorithm 4 as it does not preserve convexity, and G' may not be connected. For example, consider the cycle graph of order n, for $n \geq 7$, with a g-convex label vector ℓ of support 3. The three 1-vertices are consecutive along the cycle. If the reduction algorithm is applied to the central 1-vertex, the resulting graph is a path where only the two vertices at the ends (having degree one) are 1-vertices. Clearly, the labels are not g-convex (see Figure 5.5).

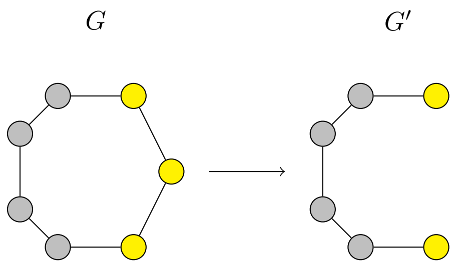


Figure 5.5: Left, a graph G with a g $-$ *convex* label in which the $1-$nodes are the yellow ones. G' is the result of *Reduce* in which the central $1-$node is removed. The label of G' is not g $-$ *convex* anymore.

On the positive side, Algorithm 4 does preserve g-convexity if 0-vertices are removed as shown by the next lemma.

Lemma 19. *Let $G = (V, E)$ be a graph and ℓ a g-convex label where $\ell_i = 0$ for $i \in V$. Then, ℓ^{-i} is a g-convex label on the components of $G' = G \setminus i$.*

Proof. Let G, G', and i be as in the lemma. Towards a contradiction, suppose that ℓ^{-i} is not g-convex on a connected component of G'. Hence, there are vertices $u, v \in V'$ such that $\ell_v = \ell_u = 1$, and a vertex $w \in I_g^{G'}(u, v)$ such that $\ell_w = 0$. Since ℓ is g-convex on G, we have that $w \notin I_g^G(u, v)$. It follows that vertices u, v are connected on a path via vertex i that is shorter than the path via vertex w. Consequently, $i \in I_g^G(u, v)$, and $\ell_i = 1$. We reach the desired contradiction, and the claim follows. □

Since adding a vertex to a graph and connecting it arbitrarily to existing vertices cannot add a chord to a path and only increase the number of vertices in $I_m(u, v)$, we have that the claim of Lemma 19 applies also to 1-vertices if ℓ is m-convex.

Observation 7. For any $v, u \in V'$, we have that $I_m^{G'}(u, v) \subseteq I_m^G(u, v)$.

Lemma 20. *Let $G = (V, E)$ be a graph, $i \in V$, and ℓ a m-convex label. Then, ℓ^{-i} is a m-convex label vector on the components of $G' = G \setminus i$.*

Proof. Let G, G', and i be as in the lemma. Suppose that ℓ^{-i} is not m-convex on a component of G'. Hence, there are vertices $u, v \in V'$ such that $\ell_v = \ell_u = 1$, and a vertex $w \in I_m^{G'}(u, v)$ such that $\ell_w = 0$. With Observation 7 it follows that $w \in I_m^G(u, v)$. We reach a contradiction since ℓ is m-convex on G. □

5.3.2 Label Vectors with Support 1

By definition, all label vectors with support equal to 1 are convex. They are a side case in our study, and we deal with them before the main sections.

Let G be a graph with convex labels ℓ and let $\mathcal{P}$ be a consistent probe vector. Our first observation implies that label vectors with support 1 can be recognized efficiently.

Observation 8. A convex label vector ℓ has support 1 if and only if $\mathcal{P}_i \leq 1$, for all $i \in V$.

The proof of the observation is trivial.

In the following, we assume that ℓ has support 1. Based on Observation 5, we identify 0-vertices $w \in V$ such that there is a neighbor $u \in N[w]$ where $\mathcal{P}_u = 0$. Note that the property can be checked easily. We apply Algorithm 4 $\mathit{Reduce}(G, \mathcal{P}, 0, w)$ in parallel, for all such vertices w, and let G' and $\mathcal{P}'$ be the resulting graph and probe vector. It's easy to verify that $\mathcal{P}'_i = 1$, for all $i \in V'$.

The core $C_{G'}$ of the reduced graph allows us to determine the number of required surgical probes to uncover ℓ.

Theorem 16. *Let G be a graph with a convex label vector ℓ of support 1, and let $\mathcal{P}$ be a consistent probe vector. Also, let $G', \mathcal{P}'$ be the output of $\mathit{Reduce}(G, \mathcal{P}, 0, w)$. Then, $|C_{G'}| - 1$ surgical probes are sufficient to uncover ℓ.*

Proof. Let $G', \mathcal{P}'$, and ℓ be be the output of $Reduce(G, \mathcal{P}, 0, w)$.

Assume that $\ell_v = 1$, for $v \in V'$. Vertex v is adjacent to all vertices $v \in V'$ since otherwise $\mathcal{P}'_u = 0$.

We conclude that vertices non-adjacent to each other vertex in G' cannot be the single 1-vertex. It follows that $v \in C_{G'}$, and that for all $i \in V' \setminus C_{G'}$, $\ell_i = 0$ holds.

For each $i \in C_{G'}$, let ℓ^i be the label vector where $\ell^i_k = 1$ if and only if $k = i$. Since $C_{G'}$ is a clique, each label vector ℓ^i, for $i \in C_{G'}$, is consistent with $\mathcal{P}'$. It follows that $|C_{G'}| - 1$ surgical probes are sufficient to distinguish between them. □

Interestingly, we show that short chords simplify CMSP dramatically. Namely, we prove that the converse of Lemma 17 is true if the label vector is t-convex.

Theorem 17. *Let $G = (V, E)$ be a connected graph with* t-convex *label vector ℓ of support at least 2, and let $\mathcal{P}$ be the corresponding probe vector. Then, the labels ℓ can be uncovered without surgical probes. Moreover, $\mathbf{1}^\ell_G = \{i \in V \mid \mathcal{P}_i \geq 2\}$*

Proof. Let G, ℓ, and $\mathcal{P}$ be as in the theorem. We show that $\ell_v = 1$ if and only if $\mathcal{P}_v \geq 2$, for $v \in V$. Sufficiency follows from Lemma 17.

To show necessity, assume that $\mathcal{P}_v \geq 2$, for $v \in V$. Either $\ell_v = 1$ or v has two neighbors $u, w \in N(v)$ such that $\ell_u = \ell_w = 1$. In the latter case, it's easy to see that $v \in I_t(u, w)$. Since ℓ is t-convex, $\ell_v = 1$, and the claim follows. □

5.3.3 Graphs with Small Maximum Cliques

In this section, we analyze CMSP for graphs where the size of a clique is bounded. To do that, we use the concept of H-free graphs.

As most of the results hold for m-convex and g-convex label vectors, we say *convex*, shorthand, for g-convex or m-convex.

K_3-free Graphs

Triangle-free or K_3-free graphs essentially do not have cliques (except two adjacent vertices). This allows us to show another situation where Lemma 17 holds in the opposite way.

Lemma 21. *Let $G = (V, E)$ be a connected and K_3-free graph with a convex label vector ℓ of support at least 2, and let $\mathcal{P}$ be a consistent probe vector. For $v \in V$, we have that $\ell_v = 1$ if and only if $\mathcal{P}_v \geq 2$.*

Proof. Let $G = (V, E)$, ℓ, and $\mathcal{P}$ be as in the hypothesis and let $v \in V$. Necessity follows from Lemma 17.

To show sufficiency, we assume that $\mathcal{P}_v \geq 2$. Towards a contradiction, suppose $\ell_v = 0$. It follows that there are two neighbors $x, y \in N(v)$ such that $\ell_x = \ell_y = 1$. Note that $(x, y) \notin E$, otherwise the vertices v, x, y induce K_3 in G, which contradicts that G is K_3-free.

It follows that the path (x, v, y) is a shortest and chordless path between x and y. Since ℓ is convex, we have that $\ell_v = 1$. We reach the desired contradiction, and the claim follows. □

Lemma 21 allows us to determine ℓ without using surgical probes if ℓ has support at least 2. Also, it is easy to check that K_2 requires one surgical probe to compute ℓ if (and only if) it has support 1. As it turns out, this is the only exception when considering K_3-free graphs.

Theorem 18. *Let G be a connected and K_3-free graph with a convex label vector ℓ, and let $\mathcal{P}$ be a consistent probe vector. Then, the labels ℓ can be computed without using surgical probes and $\mathbf{1}_G^{\ell} = \{v \in V \mid \mathcal{P}_v \geq 2\}$, except if $G = K_2$ and ℓ has support 1. In this case, one surgical probe is necessary and sufficient to uncover ℓ.*

Proof. Let $G = (V, E)$, ℓ, and $\mathcal{P}$ be as in the theorem. By Observation 8 we check if the support of ℓ is 1 or at least 2. In the latter case, the claim follows with Lemma 21.

To complete the proof, we assume that ℓ has support 1. Let $\bar{W} = \{v \in V \mid \mathcal{P}_v = 1\}$, and let $G' = G[\bar{W}]$ be the subgraph induced by $\bar{W}$.

Let $x \in V(G')$ such that $\ell_x = 1$. It follows that $N_G[x] = V(G')$, and that the core $C_{G'}$ contains x. Moreover, two neighbors $u, v \in N(x)$ cannot be adjacent since G is K_3-free.

We split the analysis into two cases according to $|N_{G'}(x)|$ without including the trivial case $|N_{G'}(x)| = 0$ which lead to $G = K_1$.

Case 1: $|N_{G'}(x)| \geq 2$. It follows that $C_{G'} = \{x\}$, and we identify x without any surgical probes. Moreover, $|V| \geq |V'| \geq 3$ and $G \neq K_2$.

Case 2: $|N_{G'}(x)| = 1$, and let $N_{G'}(x) = \{u\}$. Note that $G' = K_2$. It follows that $N_G(x) = \{u\}$. If $N_G(u) = \{x\}$, then $G = K_2$ and one surgical probe is sufficient and necessary to uncover ℓ.

Otherwise u has another neighbor $w \in N_G(u)$ (implying that $G \neq K_2$). Note that w and x are not adjacent, and hence, $\mathcal{P}_w = 0$. In this case, $\ell_u = 0$ due to Observation 5, and we identify x without using surgical probes. □

K_4-free Graphs

In this section, we study K_4-free graphs. The argument of Lemma 21 applies also to K_4-free graphs but in a weaker way. The proof of the next lemma is analogous to that of Lemma 21.

Lemma 22. *Let $G = (V, E)$ be a connected K_4-free graph with a convex label vector ℓ, and let $\mathcal{P}$ be a consistent probe vector. If $\mathcal{P}_v \geq 3$, then $\ell_v = 1$, for each $v \in V$.*

Proof. Let $G = (V, E), \ell$, and $\mathcal{P}$ be as in the lemma. Moreover, let $v \in V$ such that $\mathcal{P}_v \geq 3$. Towards a contradiction, suppose that $\ell_v = 0$. It follows that there are three neighbors $x, y, z \in N(v)$ such that $\ell_x = \ell_y = \ell_z = 1$.

Verify that one of the edges $(x, y), (x, z)$, or (y, z) is not contained in E, otherwise the vertices v, x, y, z induce K_4 in G, which contradicts that G is K_4-free.

Without loss of generality, let $(x, y) \notin E$. It follows that the path (x, v, y) is a shortest and chordless path between x and y. Since ℓ is convex, we have that $\ell_v = 1$. We reach the desired contradiction, and the claim follows. □

Lemmas 22 and 17 together allow us to determine the labels of the vertices whose probe is not equal to 2. However, we cannot fill the gap and compute all labels without using surgical probes. The following example shows that there are K_4-free graphs and probe vectors such that computing a convex label vector requires $\lfloor n/2 \rfloor$ surgical probes.

Example 10. For an even integer $h \geq 1$, consider the K_4-free graph $G_h = (V, E)$ such that $V = \bigcup_{i=1}^{h} v_i \cup x$, and $E = \bigcup_{i=1}^{h} (x, v_i) \cup \bigcup_{i=1}^{h/2} (v_{2i-1}, v_{2i})$. To complete the CMSP instance, we define probe vector $\mathcal{P}$ as follows: $\mathcal{P}_x = 1 + h/2$, and $\mathcal{P}_{v_i} = 2$ for $i = 1, \ldots, h$. The graph G_4 with probes is depicted in Figure 5.6.

The label of the central vertex x can be uncovered by Lemma 22 (if $h \geq 2$). The remaining vertices with unknown labels form adjacent pairs, e.g., v_1 and v_2. Observe that, for each pair, there are two convex label vectors $(0, 1)$ and $(1, 0)$ that are consistent with the probe vector. To distinguish between them, we need to use 1 surgical probe. It follows that $h/2$ surgical probes are required to uncover all labels.

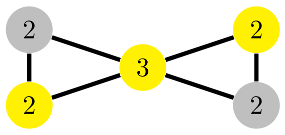


Figure 5.6: A K_4-free graph with g-convex labels indicated by the vertices' color (yellow vertices have label 1). Numeral values inside the nodes express the related probes. To uncover the labels 2 surgical probes are necessary.

The use of Lemma 22 is related to the cardinality (at least 3) of the label vector support. Otherwise, there is no vertex v such that $\mathcal{P}_v \geq 3$. We treat label vectors with support 2 as a special case, so in the following, we assume that label vectors have support 3 or more.

The main result of this section relates the number of necessary and sufficient surgical probes to the number of *true twins* in G. Recall that a pair of vertices u, v are true twins if $N[u] = N[v]$, i.e., u and v are adjacent and have the same open neighbourhood. For a graph $G = (V, E)$ and probe vector $\mathcal{P}$, we define the set $\text{TT}_{(G,\mathcal{P})} \subseteq E$ where $(u, v) \in \text{TT}_{(G,\mathcal{P})}$ if

(i) $N[u] = N[v]$,

(ii) $\mathcal{P}_v = \mathcal{P}_u = 2$,

(iii) there is $w \in N[v]$ such that $\mathcal{P}_w \geq 3$, and

(iv) for $x \in N[v] \setminus \{u, v, w\}$, we have that $\mathcal{P}_x = 1$.

We note that $\mathrm{TT}_{(G,\mathcal{P})}$ can be computed in polynomial time. First, we show that $|\mathrm{TT}_{(G,\mathcal{P})}|$ surgical probes are necessary. This lower bound holds for g-convex and m-convex label vectors.

Lemma 23. *Let G be a connected K_4-free graph with a convex label vector ℓ of support 3 or more, and let $\mathcal{P}$ be a consistent probe vector. Then, $|TT_{(G,\mathcal{P})}|$ surgical probes are necessary to uncover ℓ.*

Proof. Let $G = (V, E)$, $\mathcal{P}$, and ℓ be as in the lemma. Also, let $(u, v) \in \mathrm{TT}_{(G,\mathcal{P})}$, and let $w \in N[v]$ such that $\mathcal{P}_w \geq 3$.

By Lemma 22, $\ell_w = 1$. If there is a vertex $x \in N[v] \setminus \{u, v, w\}$, then $\mathcal{P}_x = 1$ by definition of $\mathrm{TT}_{(G,\mathcal{P})}$. By Lemma 17, it follows that $\ell_x = 0$.

Consequently, either $\ell_v = 1$ or $\ell_u = 1$, i.e., for u and v, *only* the two label vectors $(0, 1)$ and $(1, 0)$ are consistent with $\mathcal{P}$. Vertices that are adjacent to v are adjacent to u (and vice versa), and cannot help to distinguish between the two label vectors. It follows that one surgical probe is necessary to uncover ℓ_u and ℓ_v.

In total, $|\mathrm{TT}_{(G,\mathcal{P})}|$ surgical probes are necessary to uncover ℓ, and the claim is shown. □

We underline that the converse holds only for m-convex label vectors. To hit the result, we first define Algorithm 5 which uncovers 1-vertices based on Lemma 22 and removes them by calling Algorithm 4. Its correctness follows readily for m-convex label vectors.

Algorithm 5 REDUCEALL-$K_4(G, \mathcal{P})$

```
1: for all $v \in V$ such that $\mathcal{P}_v \geq 3$ do
2:     mark vertex $v$ {based on Lemma 22}
3: end for
4: for all marked vertices $v$ do
5:     $G, \mathcal{P} \leftarrow Reduce(G, \mathcal{P}, 1, v)$
6: end for
7:
8: return $G, \mathcal{P}$
```

We are ready to provide our main result of the section.

Theorem 19. *Let G be a connected K_4-free graph with a m-convex label vector ℓ of support 3 or more, and let $\mathcal{P}$ be a consistent probe vector. Then, $|TT_{(G,\mathcal{P})}|$ surgical probes are necessary and sufficient to uncover ℓ.*

Proof. Let $G = (V, E)$, $\mathcal{P}$, and ℓ be as in the lemma's hypotesis. Also, let $H, \mathcal{P}'$ be the output after applying Algorithm 5 to $G, \mathcal{P}$. As graph H may not be connected, let $H_1, H_2, \dots, H_t$ be the subgraphs induced by the connected components of H.

In the following, we analyze the structure of a component H_i, for $i \in 1, \dots, t$. We show that either $u, v \in V(H_i)$ and $(u, v) \in \mathrm{TT}_{(G,\mathcal{P})}$, or that the labels of vertices in $V(H_i)$ can be uncovered without using surgical probes.

Algorithm 5 uses Algorithm 4 to remove vertices from G and update $\mathcal{P}$ correctly. Note that, for $v \in V(H_i)$, we have $\mathcal{P}'_v \leq 2$. Let ℓ' be the ℓ restricted to $V(H_i)$. Due to Lemma 20, ℓ' is m-convex on $V(H_i)$.

We assume that there is a vertex $x \in V(H_i)$ such that $\ell_x = 1$. Otherwise, for all $v \in V(H_i), \mathcal{P}_v = 0$ and the labels of vertices can be uncovered without using surgical probes. With Lemma 17 it follows that $\mathcal{P}_x = 2$. Moreover, define $\bar{U} = \{v \in V(H_i) \mid \mathcal{P}_v = 2\}$.

Claim 1. If $|\bar{U}| = 1$, then the labels of the vertices in $V(H_i)$ can be uncovered without using a surgical probe.

Proof. If $|\bar{U}| = 1$, then $\bar{U} = \{x\}$, and vertex x is idetified without using surgical probes. With Lemma 17, we have $\ell_v = 0$, for $v \in V(H_i) \setminus \bar{U}$. □

In the following, we assume that $|\bar{U}| \geq 2$. Next, define $\bar{V} = \{v \in V \mid \mathcal{P}_v \geq 3\}$, and verify that $\bar{V}$ is non-empty since ℓ has support at least 3.

Claim 2. There is a vertex $z \in \bar{V}$ such that $(x, z) \in E$.

Proof. As $\bar{V}$ is not empty, let $y \in \bar{V}$ and note that $\ell_y = 1$. If $(x, y) \in E$, the claim follows immediately.

Assume that $(x, y) \notin E$. Since G is connected, there is a chordless path $(x, v_1, v_2, \dots, v_s, y)$ connecting x and y in G where $s \geq 1$. Since ℓ is m-convex, it follows that $\ell_{v_j} = 1$, for $j \in [s]$.

Then, $\mathcal{P}_{v_1} \geq 3$, and consequently, $v_1 \in \bar{V}$. Vertex $v_1 = z$ has the desired properties, and the claim follows. □

Since $\mathcal{P}_x = 2$ and $\ell_x = 1$, vertex z is the *only* neighbor of x in $\bar{V}$. It follows that $\mathcal{P}'_x = 1$.

Claim 3. For all $v \in V(H_i) \setminus \{x\}$, we have that $\ell_v = 0$.

Proof. Let $y \in V(H_i) \setminus \{x\}$. Towards a contradiction suppose that $\ell_y = 1$ which implies that $y \in \bar{U}$. As vertex x is adjacent to z and $\mathcal{P}_x = 2$, x cannot be adjacent to y.

Since H_i is connected there is a chordless path $(x, v_1, v_2, \dots, v_s, y)$ connecting x and y where $v_j \in V(H_i)$, for $j \in [s]$ and $s \geq 1$. Since ℓ is m-convex, we have $\ell_{v_j} = 1$, for

$j \in [s]$. It follows that x is adjacent to z and v_1 implying that $\mathcal{P}_x \geq 3$. We reach the desired contradiction, and the claim follows. $\square$

The previous claim shows that ℓ' has support 1 on H_i. It follows that, for all $v \in V(H_i) \setminus \{x\}$, we have that $\mathcal{P}'_v \leq 1$ and that $(v, x) \in E$ if and only if $\mathcal{P}'_v = 1$. We conclude that x is contained in the core $C_{\bar{W}}$ where $\bar{W} = \{v \in V(H_i) \mid \mathcal{P}'_v = 1\}$. Note that $\bar{U} \subseteq \bar{W}$ and that $x \in \bar{U} \cap C_{\bar{W}}$.

Next, we use that G is K_4-free to restrict the size of $C_{\bar{W}}$. Consider a vertex $v \in \bar{U}$. Either $\mathcal{P}'_v = 0$ and v is adjacent to two vertices in $\bar{V}$, or $\mathcal{P}'_v = 1$ and v is adjacent to x and to one vertex in $\bar{V}$ in G.

Claim 4. If vertex $v \in \bar{U} \cap \bar{W}$, then $(v, z) \in E$.

Proof. Let $y \in \bar{U} \cap \bar{W}$. Towards a contradiction suppose that $(y, z) \notin E$. Then, there is a vertex $u \in \bar{V}$ such that $(y, u) \in E$ as $\mathcal{P}_y = 2$ and $\mathcal{P}'_y = 1$.

Since $(x, y) \in E$, the path $P = (x, y, u)$ connects x and u in G. Recall that vertex z is the only neighbor of x in $\bar{V}$. It follows that $(x, u) \notin E$, and that P is a chordless path.

Since ℓ is m-convex, we have $\ell_y = 1$. But x is the only 1-vertex in $V(H_i)$, and we reach the desired contradiction. $\square$

The previous claim is essential to show the next.

Claim 5. If $|\bar{U}| \geq 3$, then $|C_{\bar{W}}| = 1$ and the labels of vertices in $V(H_i)$ can be uncovered without using surgical probes.

Proof. Assume that $|\bar{U}| \geq 3$, and let $u, v \in \bar{U} \setminus \{x\}$. Verify that if $(u, v) \in E$, then $\{x, u, v, z\}$ induce K_4 as a subgraph in G. Since G is K_4-free, $(u, v) \notin E$.

It follows that x is the only vertex that is adjacent to every vertex in $\bar{U}$. As $\bar{U} \subseteq \bar{W}$, we have $C_{\bar{W}} = \{x\}$, and vertex x can be identified without using surgical probes. For vertices $i \in V(H_i)$, we have $\ell_i = 0$. $\square$

In the following, we assume that $\bar{U} = \{x, u\}$. To finish the proof, we argue that $(u, x) \in \mathrm{TT}_{(G,\mathcal{P})}$. By definition, $\mathcal{P}_x = \mathcal{P}_u = 2$, and we showed that $z \in N(x)$ where $\mathcal{P}_z \geq 3$.

Next, we show that either $N[x] = N[u]$, or that ℓ' can be uncovered without using surgical probes.

Claim 6. If $N[x] \setminus N[u]$ is non-empty, the label of x can be uncovered without using surgical probes.

Proof. Assume there is a vertex $v \in N[x] \setminus N[u]$. Note that $v \in V(H_i)$ as z is the only neighbor of x in $\bar{V}$.

Since $(x, v) \in E$, we have $\mathcal{P}'_v = 1$, and that $v \in \bar{W}$. By definition $(v, u) \notin E$. It follows that $C_{\bar{W}} = \{x\}$, and we identify x without using surgical probes. $\square$

Claim 7. If $N[u] \setminus N[v]$ is non-empty, the label of x can be uncovered without using surgical probes.

Proof. Assume there is a vertex $v \in N[u] \setminus N[u]$. Since $(v, x) \notin E$, we have $\mathcal{P}_v = 0$. Due to Observation 5, we uncover $\ell_u = 0$. It follows that we identify x without using surgical probes. $\square$

Now, only the case where $N[x] = N[u]$ remains. Note that $N[x]$ may contain more vertices than x, u, z. Verify that if $v \in N[x] \setminus \{x, u, z\}$, then $\mathcal{P}_v = 1$. It follows that $(u, x) \in \mathrm{TT}_{(G,\mathcal{P})}$ and the theorem follows. $\square$

An important family of K_4-free graphs are outerplanar graphs. Recall that an outerplanar graph can be embedded in the plane such that edges do not intersect and all vertices lie on the outer face [43].

For maximal outerplanar graphs, each vertex appears exactly once on the outer face. Consider a pair of true twins $(u, v) \in \mathrm{TT}_{(G,\mathcal{P})}$. If $N[v] = \{u, v, w\}$, then w appears twice on the outer face. In case $|N[v]| \geq 4$, let $x \in N[v]$ such that $\mathcal{P}_x = 1$. It's easy to check that if there is a path connecting x and w, then $\{u, v, w, x\}$ induce a K_4-subgraph, and G cannot be outerplanar. Without such a path, w appears twice on the outer face. In both cases, it follows that G cannot be maximal outerplanar. We conclude that if G is maximal outerplanar, then $\mathrm{TT}_{(G,\mathcal{P})}$ is empty and that ℓ can be uncovered without using surgical probes if its support is at least 3.

Finally, we observe that Lemma 22 generalizes to K_h-free graphs, for a fixed integer h. The proof is analogue to that of Lemma 22.

Theorem 20. *Let $G = (V, E)$ be a connected K_h-free graph with a convex label vector ℓ, and let $\mathcal{P}$ be a consistent probe vector. If $\mathcal{P}_v \geq h$, then $\ell_v = 1$, for $v \in V$.*

5.3.4 Grid Graphs

Let G be a grid graph of dimension $n_1 \times n_2$ with g-convex label vector ℓ. We associate the vertices with coordinates (i, j), for $i = 1, \ldots, n_1$ and $j = 1, \ldots n_2$. By the definition of g-convexity, it follows that the 1-vertices form a sub-grid, i.e., a rectangle. We assume that the 1-vertices extend in both dimensions implying that $|\mathbf{1}_G^\ell| \geq 4$.

The convex sub-grid induced by $\mathbf{1}_G^\ell$ has four vertices $C = \{(x_k, y_h) \mid k, h = 1, 2\}$, for $1 \leq x_1 < x_2 \leq n_1$ and $1 \leq y_1 < y_2 \leq n_2$, that are located at its corners (see Figure 5.7). It follows that $\mathcal{P}_{(i,j)} = 3$ if and only if $(i, j) \in C$, and that $\mathbf{1}_G^\ell = \{(x, y) \in V \mid x_1 \leq x \leq x_2 \text{ and } y_1 \leq y \leq y_2\}$.

Theorem 21. *Let G be a grid graph with g-convex label vector ℓ, and let $\mathcal{P}$ be a consistent probe vector. The labels ℓ can be uncovered without using surgical probes.*

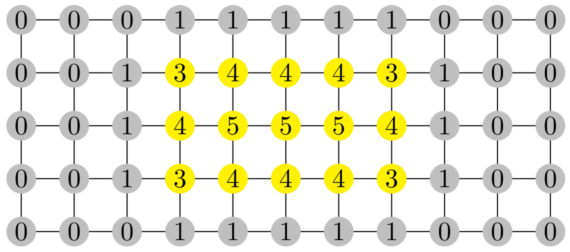


Figure 5.7: Grid graph with a convex set of 1-vertices in yellow and corresponding probe vector.

5.3.5 King's Graphs

The King's graph, named after the moves of the homonymous chess piece, is a grid graph with additional diagonal edges (see Figure 5.8). We assume that label vectors are g-convex in this section as m-convex label vectors are less interesting for applications[1].

Note that a King's graph is not K_4-free but K_5-free. It follows that if $\mathcal{P}_v \geq 5$, then $\ell_v = 1$ based on the results of the previous section. Our first goal is to show that if $\mathcal{P}_v \geq 4$, then $\ell_v = 1$. To prove our results, we use the geometric topology of King's graph.

Let $G = (V, E)$ be a King's graph of dimension $n_1 \times n_2$ with g-convex label vector ℓ. We associate the vertices with coordinates, i.e., $V = \{(i,j) \mid i \in [n_1],\ j \in [n_2]\}$. The closed neighborhood of vertex (i,j) is $N[i,j] = \{(h,k) \mid 1 \leq h \leq n_1,\ 1 \leq k \leq n_2,$ for $h \in \{i, i \pm 1\},\ k \in \{j, j \pm 1\}\}$. We call $D_{(i,j)} = \{(h,k) \mid 1 \leq h \leq n_1,\ 1 \leq k \leq n_2,$ for $h \in \{i \pm 1\},\ k \in \{j \pm 1\}\}$ the *diagonal-neighbors*, and $C_{(i,j)} = N[i,j] \setminus D_{(i,j)} \cup (i,j)$ the *cross-neighbors*. Moreover, we say two cross- (diagonal-) neighbors are *opposite* if they have one (no) common coordinate, i.e., a straight line connecting them crosses (i,j).

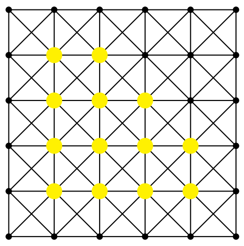


Figure 5.8: A King's graph and g-convex subset indicated by the yellow vertices.

Next, we consider the shape of g-convex subsets of V in G. Compared to the grid

[1]Consider, for example, two 1-vertices in the same row with distance two or more. Then, almost all vertices in all columns between them must also be 1-vertices.

graph (see Section 5.3.4), convex sets are not only rectangles. The diagonal edges allow for diagonal borders of the convex regions. Our first observation shows that diagonal borders have step size at most one, resembling the Eucledian meaning.

Observation 9. Let $(i, j), (i + 2, j + 1) \in V$ such that $\ell_{(i,j)} = \ell_{(i+2,j+1)} = 1$. If ℓ is g-convex, then $\ell_{(i+1,j+1)} = 1$.

The observation follows readily since there is a shortest path between (i, j) and $(i + 2, j + 1)$ through $(i + 1, j + 1)$. We remark that g-convex label vectors on the King's graph do not correspond to hv-convexity on binary matrices. This important case will be studied in Chapter 6.

Observation 9 helps to prove the next lemma.

Lemma 24. *Let $G = (V, E)$ be a King's graph with a g-convex label vector ℓ, and let $\mathcal{P}$ be a consistent probe vector. If $\mathcal{P}_v \geq 4$, then $\ell_v = 1$, for $v \in V$.*

Proof. Let $G = (V, E)$, ℓ, and $\mathcal{P}$ be as in the lemma. Moreover, let $v \in V$ such that $\mathcal{P}_v \geq 4$.

Towards a contradiction, suppose that $\ell_v = 0$. It follows that there are four 1-vertices $X = \{x_1, x_2, x_3, x_4\} \subseteq N(v)$. Verify that X contains at most two non-opposite diagonal- and cross- neighbors, respectively. Otherwise $\ell_v = 1$ is implied since ℓ is g-convex.

We conclude that X contains exactly two non-opposite diagonal- and cross-neighbors, respectively. Assume that $v = (i, j)$. Without loss of generality let $x_1 = (i, j - 1)$ and $x_2 = (i - 1, j)$ be two non-opposite cross-neighbors of (i, j). There are four configurations for the two non-opposite diagonal-neighbors x_3, x_4, e.g., $x_3 = (i - 1, j - 1)$ and $x_4 = (i - 1, j + 1)$. It is easy to check that for each configuration, Observation 9 leads to the conclusion that $\ell_v = 1$. We reach the desired contradiction, and the lemma follows. $\square$

We note that Lemma 24 describe the best situation in the sense that there are vertices v where $\mathcal{P}_v = 3$ and $\ell_v = 0$ hold.

Observation 9 implies an even stronger result for vertices that are not on the *border*. The *border* of G is $B_G = \{(i, j) \in V \mid i = 1, n_1 \text{ or } j = 1, n_2\}$. Note that non-border vertices, i.e., $V \setminus B_G$, are exactly the vertices with 8 neighbors. The next lemma shows that the converse of Observation 5 is true for non-border vertices in a King's graph.

Lemma 25. *Let $G = (V, E)$ be a King's graph with a g-convex label vector ℓ, and let $\mathcal{P}$ be a consistent probe vector. For $v \in V \setminus B_G$, we have $\ell_v = 0$ if and only if there is $u \in N(v)$ such that $\mathcal{P}_u = 0$.*

Proof. Let $G = (V, E)$, ℓ, and $\mathcal{P}$ be as in the lemma. Necessity follows with Observation 5.

To show sufficiency, let $v \in V \setminus B_G$ such that $\ell_v = 0$ and $v = (i, j)$. Lemma 24 assures we have that $\mathcal{P}_v \leq 3$. Hence, let $X = \{x_1, x_2, x_3\} \subseteq N(v)$ contain three neighbors of v where $\ell_{x_1} = \ell_{x_2} = \ell_{x_3} = 1$. Then, since ℓ is g-convex and $\ell_v = 0$, there are two possible configurations (up to symmetry): either X contains two non-opposite diagonal neighbors or cross-neighbors, respectively.

In the first case, assume that, without loss of generality, $x_1 = (i-1, j+1)$, $x_2 = (i-1, j)$, and $x_3 = (i-1, j-1)$. Consider the vertex $u = (i, j+1)$, and towards a contradiction suppose that $\mathcal{P}_u \geq 1$. It follows that there is a vertex $w \in N(u)$ such that $\ell_w = 1$. Verify that any such vertex w implies, together with Observation 9, that $\ell_v = 1$ since ℓ is g-convex. Consequently, we have $\mathcal{P}_u = 0$, and the claim follows.

In the second case, assume that, without loss of generality, $x_1 = (i, j-1)$, $x_2 = (i-1, j-1)$, and $x_3 = (i-1, j)$. Consider the vertex $u = (i+1, j+1)$, and suppose that $\mathcal{P}_u \geq 1$. Analogous to the first case, we derive a contradiction. Consequently, $\mathcal{P}_u = 0$, and the claim follows.

□

Lemma 25 allows us to uncover the labels of vertices in $V \setminus B_G$, while Lemmas 17 and 24 detect the label of vertices $v \in B_G$ where $\mathcal{P}_v \leq 1$ or $\mathcal{P}_v \geq 4$. We complete the analysis by showing a few special cases, showing that a g-convex label vector can be uncovered without using surgical probes on a King's graph.

In many applications, $\mathbf{1}_G^\ell$ presents a set whose exact location and boundaries must be determined. Here, it is reasonable to assume that the object is contained entirely in a (large enough) grid, i.e., does not intersect with the borders. We say that the label vector ℓ is *zero-border* on G if $\ell_v = 0$, for all $v \in B_G$.

Theorem 22. *Let G be a King's graph with g-convex label vector ℓ, and let $\mathcal{P}$ be a consistent probe vector. If ℓ is zero-border, ℓ can be uncovered without using surgical probes.*

Theorem 22 follows readily from Lemma 25.

Chapter 6

Reconstruction of convex polyominoes and related problems

In this chapter we consider the problem of reconstructing convex polyominoes from two different notions of projections and convexity.

In the first section we consider the reconstruction of full convex polyominoes starting from horizontal and vertical projections. In particular, the computational complexity of this problem is not yet known and only partial results are present in the literature.

In the second section, we study a variant of the MSP problem (introduced in Chapter 5) that considers binary matrices and projections collected using two types of scanning windows over it. We solve the problem by providing algorithms that reconstruct a binary matrix in polynomial time. We refer to Chapter 2 for the definitions.

We highlight that part of the results presented in this chapter are published in [59]

6.1 Reconstruction of Convex Polyominoes

In this section we study the reconstruction of (full) convex polyominoes, considered as connected sets of cells on a square surface, using their horizontal and vertical projections. We initially start by describing the algorithm used for the reconstruction of $hv-$convex polyominoes. Then, we show a possible approach to the reconstruction of full convex polyominoes relying on this algorithm. Since the generalization is far from trivial, we present some attempts found in the literature that lead to the definition of subclasses of convex polyominoes that can be reconstructed in polynomial time. Finally, we provide new results regarding the SAT formulas appearing in the reconstruction phase of the algorithm. Although the complexity problem remains open, the properties presented in this section may help in its determination.

6.1.1 The reconstruction of hv-convex polyominoes from horizontal and vertical projections

In [30] the authors defined the algorithm $HVRec$ that reconstructs (in polynomial time w.r.t. its projections' dimensions) a hv-convex polyomino compatible with an input couple of horizontal and vertical projections H and V, if it exists. In [60], it has been proved that imposing hv-convexity does not guarantee the uniqueness of the reconstruction of a polyomino from a couple of projections. This is due to the presence of specific configurations of points called *switching components* (see [21] for a survey on the topic and [61, 62, 33] for the characterization and related results). The cells of these configurations can be divided into two disjoint subsets having the same projections (proving the non-unicity of the set in which they belong). In the following, each of these sets will be labelled with a boolean variable or its negation, to underlie their complementarity. The strategy of the reconstruction concerns the iterative detection of two subsets of cells: the *kernel* whose cells belong to all the solutions compatible with H and V, if any, and the *shell* whose cells are outside all solutions. Finally, the cells not yet assigned are proved to belong to the switching components of the final solutions and their belonging to a $hv-$covnex solution is evaluated according to a defined $2-$SAT formula.

In more details, $HVRec$ gets as input two vectors H and V and performs the two following steps:

Step 1 ***(kernel and shell computation)*****:** according to each possible position of the $hv-$convex feet, it detects the cells that are common to all the $hv-$convex polyominoes having H and V as horizontal and vertical projections, say the *kernel*. At the same time, it also detects the cells that are external to the polyomino and that constitute the *shell* of the polyomino. Both the detection tasks are iteratively performed by using a sequence of *filling operations* that take advantage of the convexity constraints and from the knowledge of the vectors H and V. In particular, some filling operations preserve the convexity, i.e. cells comprised between $kernel$ cells belong to $kernel$ and, similarly, cells between a $shell$ cell and the border belong to the $shell$. Other filling operations take into account the horizontal and vertical projections to decide to which set points belong, see [30] for a complete description of them.

So, $HVRec$ converges to the final kernel by approximating it both from inside and from outside. The process ends when the filling operations either fail, meaning that no solution exists for the chosen feet positions or leave the kernel and the shell unchanged. In the latter case, the cells that remain unidentified, so not belonging to the kernel or to the shell, are grouped in an ambiguous set $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_s\}$, where each X_i is a *switching component*. If $\mathcal{X} = \emptyset$, then the polyomino has been successfully reconstructed. Otherwise, Step 2 is required;

Step 2 ***(2-SAT formula definition and valuation)*****:** each switching component in $\mathcal{X}$ is detected and its cells alternatively labelled with a new boolean variable x or $\overline{x}$. Recall that $\overline{x}$ stands for not x. Finally, a $2 - SAT$ formula involving all the variables is defined. It imposes on the cells of $\mathcal{X}$ both the hv-convexity of the kernel and the shell, preserving the coherence with the vectors H and V. A valuation of the 2-SAT formula (obtained in polynomial time) leads to a solution of the reconstruction problem in the sense that a cell of $\mathcal{X}$ belongs to the polyomino if and only it is labelled with a variable whose valuation is true. The computed polyomino is then provided as output.

To further grab the concept and the importance of *switching components*, consider the following definition (defined in [33]).

Definition 11. Consider the points belonging to $\mathcal{X}$ after Step 1. Given the four areas NW, NE, SW, SE (see Figure 6.1) we have that:

- the vertical correspondent of point $p = (i, j)$ belonging to $SE \cup SW$ is the point $\bar{p} = (i, j + v_j)$;
- the vertical correspondent of a point $p = (i, j)$ belonging to $NE \cup NW$ is the point $\bar{p} = (i, j - v_j)$;

Similar definitions hold for horizontal correspondents.

The importance of the previous definition relies on the fact that if p and p' are correspondents, then $p \in kernel$ if and only if $p' \notin kernel$ is not. Moreover, correspondences between points define (necessary close) paths, the previously mentioned *switching components*. An example of corresponding points is shown in Figure 6.1

We formally define a switching component as follows:

Definition 12. A *Switching Component* $\mathcal{X}_i$ is a closed path of corresponding points $p_r, r \in \mathbb{Z}_{2l}$ such that p_{2k} and p_{2k+1} are vertical correspondents, while p_{2k-1} and p_{2k} are horizontal correspondents. $\mathcal{X}_i[0]$ denote the set of vertices with even indices and $\mathcal{X}_i[1]$ the set of vertices with odd indices.

For example, Figure 6.2 shows a polyomino with a single switching component. Since each point in $\mathcal{X}_i[0]$ and $\mathcal{X}_i[1]$ has a unique horizontal and vertical correspondent in the other set, they have the same horizontal and vertical projections.

Therefore, any solution must fall in one of the following cases:

1. either $\mathcal{X}_i[0] \subseteq S$ and $\mathcal{X}_i[1] \cap S = \emptyset$;
2. or $\mathcal{X}_i[1] \subseteq S$ and $\mathcal{X}_i[0] \cap S = \emptyset$.

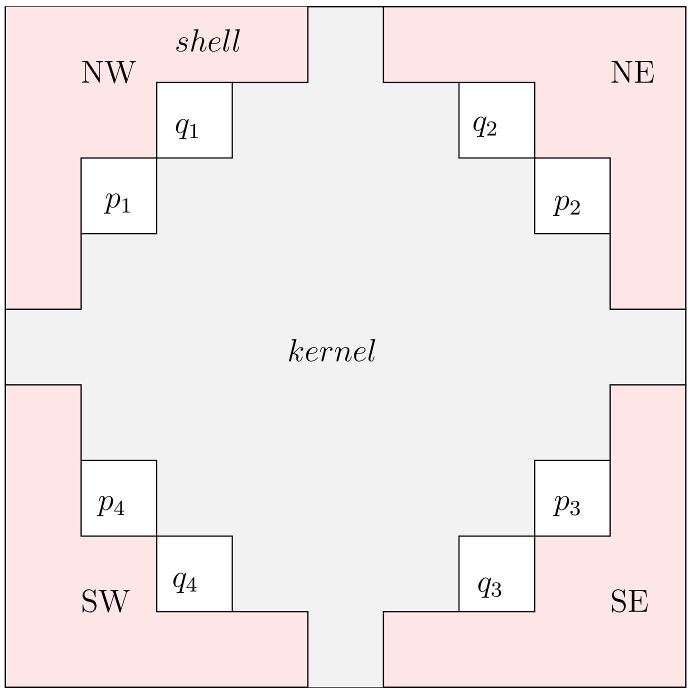


Figure 6.1: Example of a polyomino after running Step 1 of $HVRec$. The *kernel* and the *shell* are highlighted. The couples of points $(p_1, p_4), (p_2, p_3), (q_1, q_4), (q_2, q_3)$ are vertical correspondents. Similarly, the couple of points $(p_1, p_2), (p_3, p_4), (q_1, q_2), (q_3, q_4)$ are horizontal correspondents. Note that the paths $p_1p_2p_3p_4$ and $q_1q_2q_3q_4$ define two different switching components. The four zones outside the *kernel* are highlighted.

As previously underlied, the two possible states of the cells belonging to a switching component can be encoded by a boolean variable. The search for an assignment of the Boolean variables is done by expressing the constraints issued from $hv-$convexity on the Boolean variables associated with each switching component.

Example 11. Suppose that Figure 6.5, (a) depicts a part of the NW path of a polyomino reconstructed after performing Step 1 of $HVRec$. Assume that the cells labelled $x_1, \ldots, x_7$ belong to the set $\mathcal{X}$. Step 2 requires defining a sequence of 2-SAT formulas coding the hv-constraint on them both related to the kernel and to the shell. More precisely, the h-convexity of the kernel imposes the formulas $x_2 \to x_3$, and $x_6 \to x_7$, while the v-convexity imposes $x_6 \to x_5$. Note that the h-convexity of the shell imposes their equivalent clauses. These clauses (by abuse of notation we indicate the implication $(x \to y)$ as a clause due to its logical equivalence to the clause $(\neg x \wedge y)$) guarantee a valuation that preserves the hv-convex of the final solution, as desired.

To conclude this section, we note that for several years it has been observed that the second step of $HVRec$ always provides a solution. This observation leads to the empirical conjecture that states the $2-SAT$ formulas expressing the $hv-$convexity are always feasible.

However, in [63] an example is provided in which this is not the case. The authors called it the *bad guy*. We show this counter-example in Figure 6.2.

6.1.2 An approach to reconstruct convex polyominoes from horizontal and vertical projections

The authors of [32, 33], underline that there is no natural generalization of ***HVRec*** when hv-convexity constraint changes into full convexity. In particular it is proved that the 2-SAT formula imposing the hv-convexity defined in Step 2 may change into a k-SAT formula, with $k \geq 2$.

Relying on ***HVRec***, we approach a reconstruction strategy, say $CRec$ (Convex Reconstruction), by defining a slightly modified version of $HVRec$. In particular, it is modified as follows:

StepConv 1: apply Step 1 and include in the kernel the cells of its convex hull after each iteration of the filling operations. The computation of the shell is left unchanged;

StepConv 2: each switching component in $\mathcal{X}$ is detected as in Step 2. However, in general, a SAT formula ϕ_{Conv} is defined to impose convexity on $\mathcal{X} \bigcup kernel$. In particular, φ_{Conv} includes both those clauses φ_{hv} of Step 2 and, for each point $x \in \mathcal{X}$, the clauses $\varphi_x = (x \to x_1) \wedge \cdots \wedge (x \to x_n)$, with $x_1, \ldots, x_n$ being the points belonging to the convex hull H_x computed after the inclusion of x in the kernel.

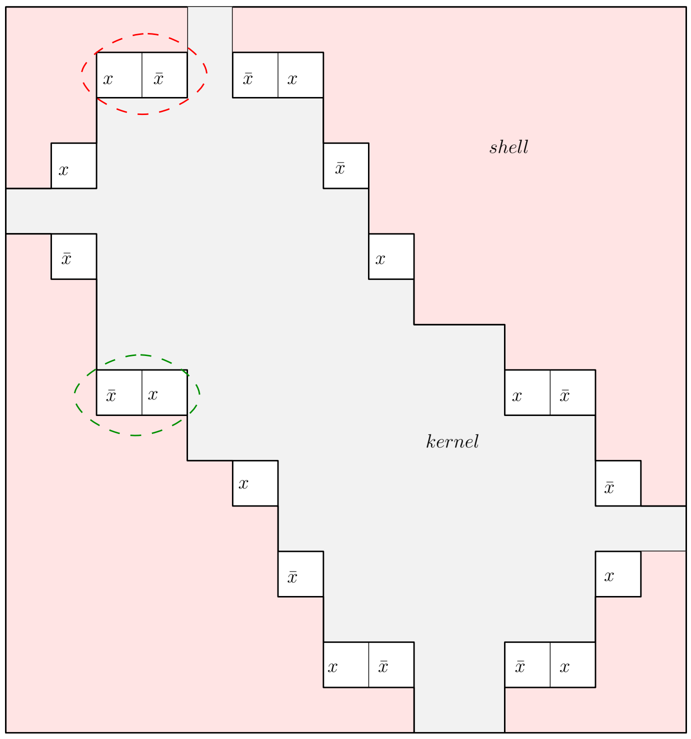


Figure 6.2: The *bad-guy*. In this example the $hv-$convexity constraints lead to inconsistent $2-$clasuses, providing a counter-example to the conjecture that the $2 - SAT$ formulas arising in $HVRec$ always have a solution. The red and green circles highlight where we obtain incompatible constraints. In fact, we have both $x \to \bar{x}$ and $\bar{x} \to x$.

Some properties of the clauses that can arise in StepConv 2 are hereafter provided, leaving open the computational complexity of their valuation. As an example, a similar way forward is used in [64] where the reconstruction of a sub-class of hv-convex polyominoes is performed by means 3-SAT Horn clauses whose valuation requires polynomial time.

So, let us consider the points of $\mathcal{X}$ that lie above the WN-path of the convex kernel identified in StepConv 1 (for the points related to the three remaining NE, ES and SW convex paths we proceed analogously). We recall that the membership of these points to one of the convex polyominoes consistent with the input horizontal and vertical projections H and V, if any, has to be determined.

The clauses of the formula we are going to define in StepConv 2 consider how the inclusion of each point in $\mathcal{X}$ reflects on the others, in order to preserve the convexity of the structure.

We provide a detailed implementation of $CRec$ in *Matlab* in the Appendix.

6.1.3 Previous results

Previous works have studied the problem in some subclasses. In particular, consider the following definitions.

Definition 13. A switching component is *regular* if the turning angle at each vertex has a constant orientation (i.e. it always turns clockwise or always counterclockwise). All the other switching components are called *irregular*.

For example, Figure 6.3 shows an example with both regular and irregular switching components.

Definition 14. Two switching components $\mathcal{X}_i$ and $\mathcal{X}_j$ are *adjacent* if they contain respectively a point p_i and a point p_j which are $4-$adjacent.

In [33] the following result is shown.

Lemma 26. *Consider an instance of $HVRec$ or $CRec$. Then the Boolean variables of any pair of adjacent regular switching components are equal.*

The proof relies on the study of the relative position that switching components must have, employing a directed graph that describes all the possible configurations that arise starting from one of the four regions NW, NE, SW, SE.

Interestingly, the previous lemma allows us to prove the following general results.

Theorem 23. *Consider an instance of $HVRec$ or $CRec$. If all the switch components are regular, then any pair of them is either equivalent or independent.*

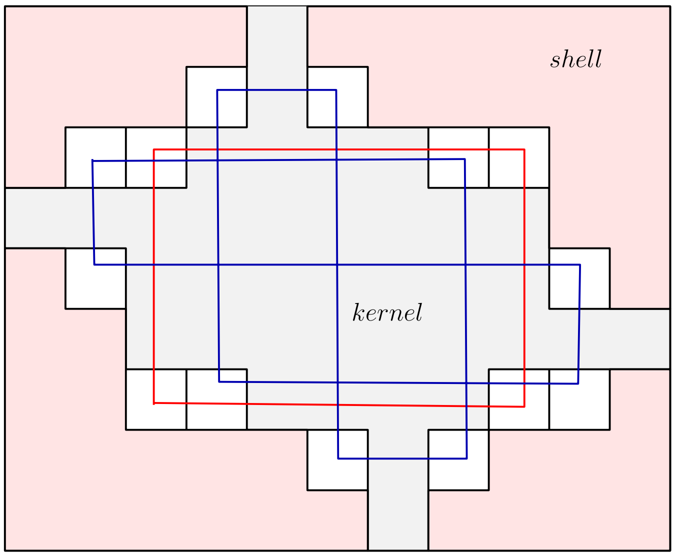


Figure 6.3: A configuration in which are present both regular and irregular switching components. In particular, the red one is a regular switching component, while the blue one is irregular.

Theorem 23 states that we can have either equivalent or independent regular switching components. Therefore, the notion of *extended switching component* can be introduced.

Definition 15. An extended switching component $\tilde{\mathcal{X}}_i$ is the union of all the switching components whose Boolean variables can be evaluated together, i.e., showing only two possible admissible patterns for all of them together.

The main difference between the previous notion and the standard switching components is that, for the former, it is possible to define an order.

Formally, we introduce an order relation between points in $\mathcal{X}$. For a pair of points $p(x, y)$ and $p'(x', y')$ we denote $p < p'$ if $x < x'$ or if $x = x'$ and $y < y'$ for the points of West areas. On the other hand, we say that $p < p'$ if $x > x'$ or if $x = x'$ and $y < y'$ for the East areas. This order relation is defined on arbitrary set of points and induced an order relation $A < B$ between two sets if, for any pair of $a \in A$ and $b \in B$, we have $a < b$. Interestingly we have the following result holds.

Proposition 8. If all switching components are regular, the extended switching components can be ordered.

This result confirms that extended switching components have a simple structure along the boundary of the *kernel*, since they are independent and well-ordered.

Recent results [63] used the previous tools. In particular, it has been proved recently that a subclass of the convex polyominoes can be reconstructed in polynomial time.

In more detail, consider the following definition.

Definition 16. Consider a set $S \subseteq \mathbb{Z}^2$. S is *thin* if there exists a point $p = (X, Y) \in \mathbb{Z}^2$ such that the feet of S are strictly located in diagonally opposite quadrants of (X, Y), i.e. it must holds:

- $x(South(S)) < X < North(S))$ and $y(West(S)) < Y < y(East(S))$ or
- $x(South(S)) > X > x(North(S))$ and $y(West(S)) > Y > y(East(S))$.

If S is not thin, it is *fat*

Figure 6.4 shows examples of *thin* and *fat* $hv-$convex configurations.

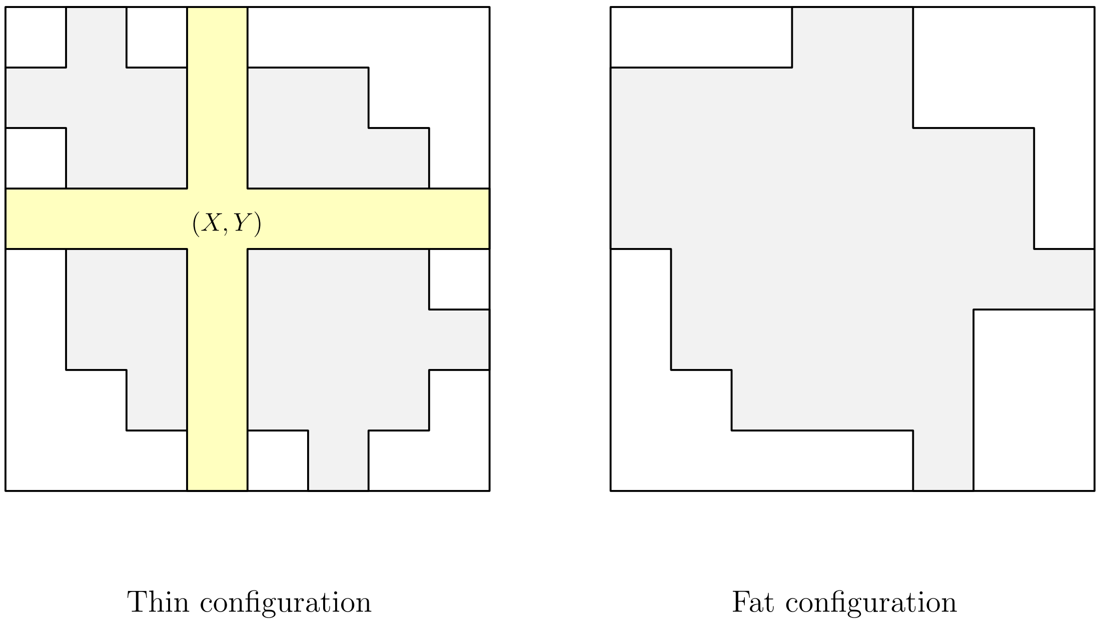


Figure 6.4: Examples of thin and fat $hv-$convex polyominoes. This property depends on the relative position of the set's feet.

In [63] the author proves the following theorem.

Theorem 24. *The class of fat convex polyominoes can be reconstructed in polynomial time.*

6.1.4 Convexity preserved by k-SAT clauses

Coming to the original results, we code the boundary of the considered polyomino using its *boundary word* [38]. For all the notions used here, we refer to Chapter 2.

First of all, we show that a switching component's cell inclusion in the kernel can be performed if the point is minimal w.r.t. the Christoffel word where it lies on.

In particular, from [40], Theorem 1 and its corollaries, it follows

Proposition 9. Let w be the Christoffel word of slope $\frac{a}{b}$ and denote by $w(i : j)$ the substring of w between indices i and j. Then, it exists only one index $1 < i < a + b$ in which $w(i, i + 1) = 01$, such that $w_1 = w(1 : i - 1)\, 1$ and $w_2 = 0\, w(i + 2 : a + b)$ (we consider the substring $w(i + 2 : a + b) = w(a + b)$ if $i = a + b - 1$) are both Christoffel words. Furthermore, it holds that $i = min(w)$.

Relying on Proposition 9, we obtain the following result stating that the inclusion in a NW Christoffel word w of a point different from $min(w)$ does not preserve the convexity of w and so of the whole NW path.

Proposition 10. Let w be the Christoffel word of slope $\frac{a}{b}$ and $min(w) < i < a + b$. The Christoffel path of slope $\frac{|w(1:i-1)|_1+1}{|w(1:i-1)|_0}$ includes the point $c(\widetilde{w})$. On the other hand, if $1 < i < min(w)$, then the Christoffel path of slope $\frac{|w(i+1:a+b)|_1}{|w(i+1:a+b)|_0+1}$ includes the point $c(\widetilde{w})$.

Proof. Recall that the two indexes $c(\widetilde{w})$ and $min(w)$ are equal in the upper and lower Christoffel paths, respectively. Let us proceed by contradiction assuming that the Christoffel path of slope $\frac{|w(1:i-1)|_1+1}{|w(1:i-1)|_0}$ does not include the point $c(\widetilde{w})$, when $min(w) < i < a+b$. We consider the integer points $O = (0, 0)$, $B = (b, a)$ and $C = (|w(1 : i - 1)|_0, |w(1 : i - 1)|_1 + 1)$. By the proof of Theorem 3.3 in [65], it holds that the triangle OCB contains at least one integer point. Let D be the point of OBC closest to the line segment of slope $\frac{a}{b}$. It follows that the triangle ODB contains no integer points and furthermore, by assumption, D is different from $c(\widetilde{w})$, reaching a contradiction. A similar argument holds if $0 < i < min(w)$. □

We can rephrase this proposition in a more algorithmic fashion.

Corollary 8. *Let w be a Christoffel word in the WN-path of the kernel obtained after StepConv 1. If the point $c(\widetilde{w})$ belongs to the shell, then all the points of $\mathcal{X}$ that lie on w also belong to the shell.*

Corollary 9. *Let w be a Christoffel word in the WN-path of the kernel obtained after StepConv 1. If we include in the final convex solution a point of $\mathcal{X}$ lying above its WN-path, then also the point $c(\widetilde{w})$ has to be included in order to preserve the convexity.*

Example 12. Let us assume that StepConv 1 provided the WN-path depicted Figure 6.5 and that the cells $x_1, \ldots x_7$ belong to $\mathcal{X}$. The $WN-$path is the Christoffel word of slope

$m = \frac{6}{11}$, $w = 0010010010010$ **01** 01, where the boldface entries indicate its minimal point. If we require to add the point x_3, then the computation of the related convex hull (straight line in Figure 6.5, (b)) imposes the inclusion of the points x_4 and x_5. So the two new clauses $x_3 \to x_4$ and $x_3 \to x_5$ have to be added to the clauses already defined in Example 11. Note that x_5 is $c(\widetilde{w})$ as expected by Proposition 10. Note that the inclusion of the points x_3, x_4 and x_5 in the kernel produces the Christoffel words $w_1 = 00100101$ and $w_2 = (001)^3$ (that is not primitive) whose slopes preserve the decreasing order.

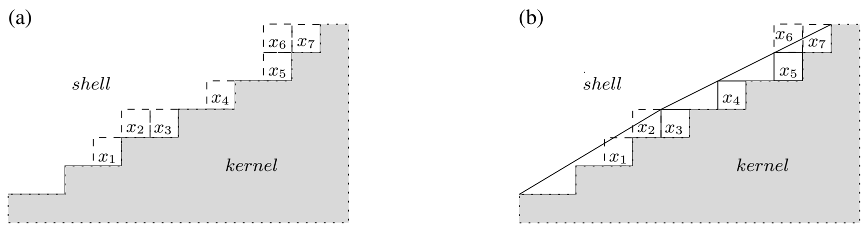


Figure 6.5: The WN-path of the kernel computed by StepConv 1 and the points belonging to $\mathcal{X}$. In (a) the situation is depicted, while in (b) it is shown the convex hull computed after the inclusion in the kernel of x_3. To preserve convexity, it is also required the inclusion of x_4 and x_5.

Furthermore, we underline that, if the word w of Example 12 is followed by a Christoffel word of slope m, with $\frac{1}{2} < m < \frac{6}{11}$, then the slope of the WN-path is not preserved after the splitting, and some more points may need to be included. This situation may also arise if we include in the WN-path of the kernel in Figure 6.5 the point $c(\widetilde{w}) = x_5$ only. The following example shows the situation for a different WN-path.

Example 13. Let w_1 and w_2 be two Christoffel words of a WN-path of a convex kernel, with $\rho(w_1) = \frac{3}{5} > \rho(w_2) = \frac{11}{20}$ as in Fig. 6.6. Including in the kernel the point $x = c(\widetilde{w}_1)$ changes w_1 into two new Christoffel words $u_1 v_1$, with $\rho(u_1) = \frac{2}{3}$ and $\rho(v_1) = \frac{1}{2}$. Now, the sequence of slopes $\rho(u_1), \rho(v_1)$ and $\rho(w_2)$ is not decreasing. Furthermore, $(v_1\ w_2) = 001\ 0010010010010010100100100100101$ is not a Christoffel word, so the corresponding path is not $WN-$convex. To get back convexity, we need to include a second point in the polyomino, i.e., the point $y = c(\widetilde{w}_2)$ that belongs to the convex hull H_x depicted in Fig.6.6, on the right, obtaining the word $w_3 =$ 0010010010010010 **10** 0100100100100101, where the included point is in boldface (see Fig. 6.6 for a visual representation).

The $WN-$convexity is so imposed by the clause $x \to y$ that, in terms of Christoffel words, produces $v_1\ w_2$ changes into w_3.

In [32], the authors consider all possible cases that arise when a single point is included in a WN-path, and determine when further points need to be included to preserve

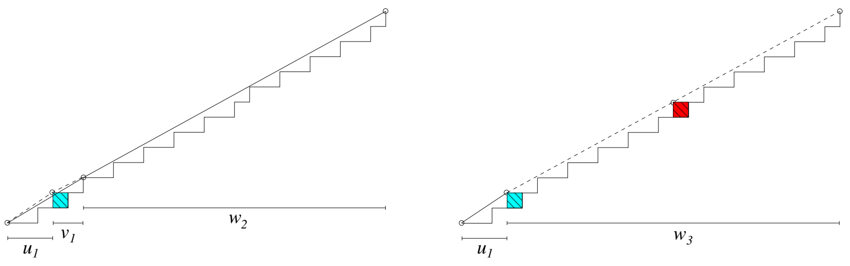


Figure 6.6: The inclusion of one single point (on the left) prevents the WN-convexity of the path. A second point (on the right) has to be added to keep it back.

global convexity. In the sequel, we characterize these situations through logical implications.

We underline that, if the convex hull H_x related to a point $x \in \mathcal{X}$ includes points of the shell, since their values can be considered 0, then the same value is transferred to x by the implication $x \rightarrow 0$.

Example 14. Consider the Christoffel word $w = 00101001010010101$ of slope $\rho = \frac{7}{10}$ that is in the $NW-$path of a kernel. Let $x_1, \dots x_6$ be points of $\mathcal{X}$ (see Figure 6.7 (a)). Let us include x_2 to the kernel, i.e. $x_2 = 1$. Then, we must also include x_1, x_3 and x_5 obtaining $\varphi_{x_2} = (x_2 \rightarrow x_1) \wedge (x_2 \rightarrow x_3) \wedge (x_2 \rightarrow x_5)$ (see Figure 6.7 (b)). On the other hand, if x_6 is included, then x_3 and x_5 must be also included in the kernel obtaining $\varphi_{x_6} = (x_6 \rightarrow x_3) \wedge (x_6 \rightarrow x_5)$ (see Figure 6.7 (c)). Finally, if we include both x_2 and x_6 and we compute again the convex hull, then we realize that that also the points x_1, x_3, x_4, and x_5 have to be included, with x_4 being a new one (see Figure 6.7 (d)). So, a new clause has to be added to $\varphi_{x_2} \wedge \varphi_{x_6}$, i.e. $(x_2 \wedge x_6) \rightarrow x_4$. This provides an example of a situation where a $3 - SAT$ formula is required.

Relying on the above example, it may happen, in general, that the inclusion of a subset of k points of $\mathcal{X}$ leads to a $(k+1)$-SAT formula as defined in StepConv 2. In the sequel, we define some properties that allow us to simplify the SAT clauses leading to a normal form involving only 3-SAT clauses. Up to now, no polynomial time valuation is known for this class of formulas.

6.1.5 Properties of the k-SAT formulas to impose global convexity

As previously shown, in [63] the author proved that in the case of fat convex polyominoes, adding points of $\mathcal{X}$ to the convex kernel can be performed in polynomial time. This result is related to some specific switching components of the elements of the class.

However, the complexity for the whole set is still unknown. Therefore, the general versions of the algorithm to reconstruct convex polyominoes from projections may ben-

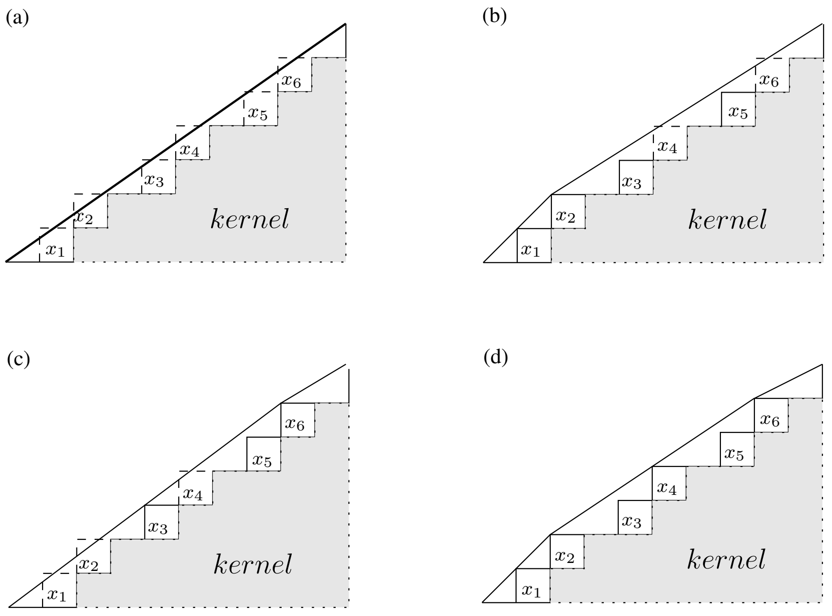


Figure 6.7: An example of $3 - SAT$ in the convex reconstruction problem.

efit from the following results about the SAT formula φ_{Conv} that has to be defined in StepConv 2 to express the global convexity. Given two points x and y in $\mathcal{X}$, we define the partial order $x <_o y$ if and only if x precedes y while moving clockwise along the WN-path of the kernel.

Proposition 11. Let $c = (x_1 \wedge x_2) \rightarrow x$ be a clause included φ_{Conv}. Then $x_1 <_o x <_o x_2$.

Proof. Let us assume, w.l.o.g., that $x <_o x_1 <_o x_2$. This implies that either the segment $[x_1; x_2]$ is one of the sides of the convex hull computed after the kernel inclusion of x_1 and x_2 or it lies behind it. In the first case, it holds $x_1 \rightarrow x$, while in the second case either $x_1 \rightarrow x$ or $x_2 \rightarrow x$ according to which among x_1 or x_2 is a vertex of the convex hull. So, c is equivalent to one of the two above clauses and it has not to be included in φ_{Conv}. □

Proposition 12. Let $c = (x_1 \wedge x_2 \wedge x_3) \rightarrow x$ be a clause included in φ_{Conv}. There exists a 3-SAT clause that is equivalent to c.

Proof. Let us assume w.l.o.g. that $x_1 < x < x_2 < x_3$, where $x_i < x_j$ means that the point x_i precedes x_j while moving clockwise along the WN-path of the kernel. Then the convex hull computed after the kernel inclusion of the three points x_1, x_2, and x_3 includes also x. Two cases arise:

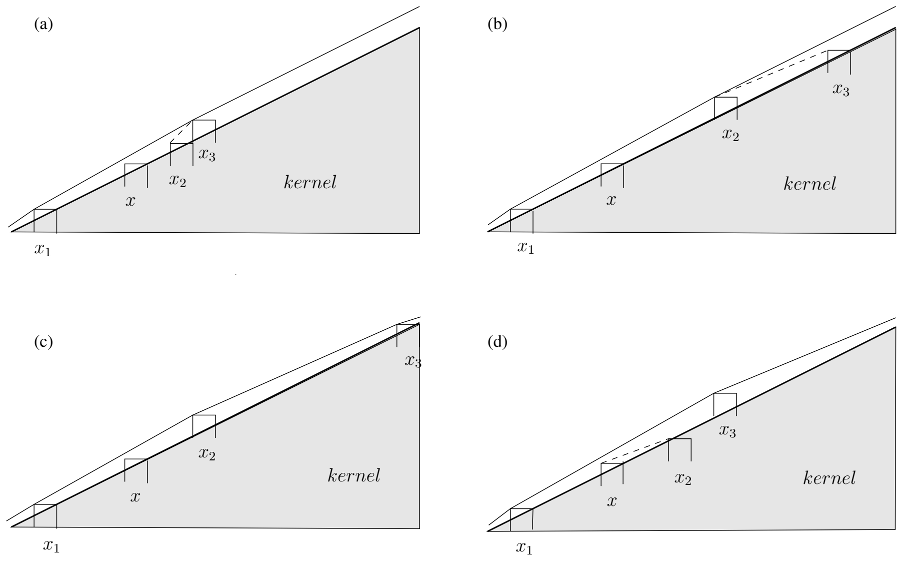


Figure 6.8: The different configurations of points related to the clause c in the proof of Proposition 6.1.5. Internal segments are dashed while a full line indicates the convex hull.

$i)$ the line segment $[x_2; x_3]$ lies inside the convex hull. In this case either the segment $[x_1; x_3]$ or $[x_1; x_2]$ is a side of the convex hull. In the first case the point x_2 lies below it, see Figure 6.8, (a). Then $(x_1 \wedge x_3) \rightarrow x_2$, and c turns out to be equivalent to $c_1 = (x_1 \wedge x_3) \rightarrow x$. Analogously, in the second case x_3 lies inside the convex hull, see Figure 6.8, (b). Then $x_2 \rightarrow x_3$, and c turns out to be equivalent to $c_1 = (x_1 \wedge x_2) \rightarrow x$.

$ii)$ the line segment $[x_2; x_3]$ is a side of the convex hull. Reasoning similar to $i)$ holds. Two cases arise: either $[x_1; x_2]$ is also a side of the convex hull or it is internal to the convex hull. In the first case c is equivalent to $c_1 = (x_1 \wedge x_2) \rightarrow x$, see Figure 6.8, (c). In the latter case, the clauses $x_2 \rightarrow x_1$ and $x_2 \rightarrow x$ are equivalent to c, see Figure 6.8, (d).

□

A similar reasoning on a generic SAT clause leads to

Corollary 10. *Let $c = (x_1 \wedge \cdots \wedge x_k) \rightarrow x$ be a clause in φ_{Conv}, with $k \geq 2$. There exists a set of clauses in 3-SAT that are equivalent to c.*

Proposition 13. If a clause c of φ_{Conv} includes both x and $\bar{x}$, then c can be reduced to 2−SAT.

Proof. The proof directly follows from the definitions of the logical operators. Two cases arise: if $c = (x_1 \wedge \bar{x}_1) \to x$, then the implication is a tautology. Otherwise, if $c = (x_1 \wedge x) \to \bar{x}_1$, then only three valuations of x_1 and x are admissible to have $c = 1$: either $x_1 = 1$ and $x = 0$ (so being $\bar{x}_1 = 0$) or $x_1 = 0$ and x is either 0 or 1. These valuations are equivalent to the 2-SAT clause $x_1 \to \bar{x}$. $\square$

Experimental evidence allows us to conjecture the following statement:

Conjecture 2. Consider two consecutive Christoffel words w_1 and w_2 such that adding $c(\widetilde{w}_1)$ or $c(\widetilde{w}_2)$ separately does not cause the addition of anything else in the convex hull. Then, if $(x_i \wedge x_j) \to x_k$ is a clause of φ_{Conv}, then x_i and x_j cannot be both minimum of the two consecutive Christoffel words.

From the above conjecture it follows that, in case StepConv 1 detects a set $\mathcal{X}$ whose elements are minimum of the Christoffel words of the kernel border, then the reconstruction procedure can be performed in polynomial time.

6.2 *MSP* for polyominoes

In this section, we introduce a new perspective on the reconstruction problem faced so far. In particular, we consider a general setting in which projections are computed using 8 and $4-$neighbours of each element of the matrix. We will see later in this section that these types of problems are strongly related to their graph counterparts.

Consider a binary matrix A of dimension $n \times m$ and a position (i, j) inside it. We define $N^8(i, j)$ as the set of the positions in the 8-neighbourhood of (i, j), i.e.,

$$N^8(i,j) = \{(h,k) \,|\, 1 \leq h \leq n,\ 1 \leq k \leq m,\ \text{for } h \in \{i, i\pm1\},\ k \in \{j, j\pm1\}\} \cup \{(i,j)\}.$$

Analogously, the set $N^4(i, j)$ is the 4-neighbourhood of (i, j) and contains the vertical neighbours $(i \pm 1, j)$ and the horizontal neighbours $(i, j \pm 1)$ entries, including (i, j) itself.

Analogously to what did in Chapter 5, we indicate *probe* p^8_{ij} (resp. p^4_{ij}) the sum of the entries of the elements of A in the positions $N^8(i, j)$ (resp. $N^4(i, j)$).

When varying (i, j), we can arrange the probes of the $m \times n$ matrix A in two integer projections matrices of the same dimensions P^8 or P^4, respectively, according to the N^8 or N^4 considered set. The entries of the two probe matrices P^8 or P^4 range from 0 to 9, in the first case, or from 0 to 5, in the latter, i.e., the size of the considered neighbourhood sets.

As explained in the previous section, when we are interested in reconstructing hv-polyominoes using their projections, an issue is the uniqueness of the obtained binary matrix solution. To remove this ambiguity, we repeat what was done in Chapter 5, using

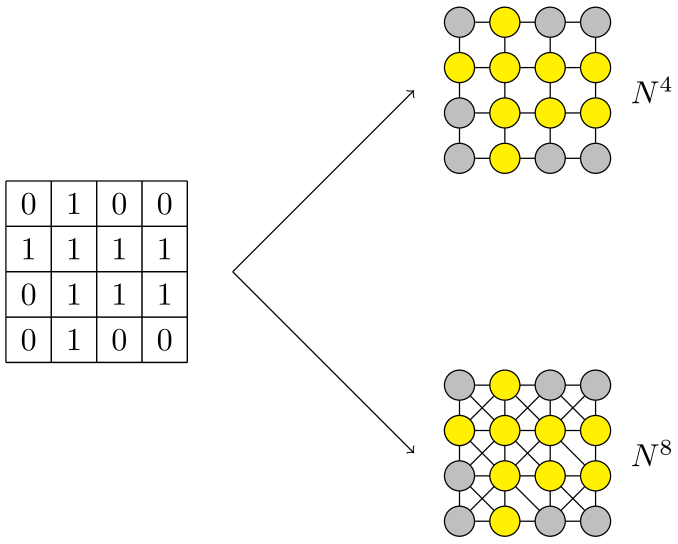


Figure 6.9: Equivalence between the binary matrix on the left with the Grid Graph (up) and the King's Graph (down). Grey nodes represent 0-entries and yellow nodes are $1-$entries.

the *surgical probes*. We recall that a surgical probe at position (i, j) returns the value of a_{ij}. Therefore, considering a binary matrix A, the following problem arises.

Definition 17 (hv – Minimum Surgical Probing)**.** Given a projection matrix P, the Minimum Surgical Problem (MSP_{hv}) asks to find the minimum number of surgical probes needed to uniquely determine a $hv-$convex matrix A consistent with P, if it exists.

This problem was presented also in Chapter 5 for graphs, but here we are focused on its application for polyominoes. Here we only consider the cases in which $P = P^4$ or $P = P^8$. However, a much more general subset of positions may be considered.

Note that using the notions introduced in Chapter 5, A can be modelled as a graph G, depending on the type of probe considered. In fact, suppose to associate to each entry (i, j) of A a node $v_{ij} \in V(G)$. Then, $N^8(i, j)$ corresponds to the closed neighbour set of node v_{ij} if G is a King Graph. Similarly, $N^4(i, j)$ corresponds to the closed neighbour set of node v_{ij} if G is a Grid Graph. Thus, this problem can be equivalently considered as a MSP problem on graphs in which each node possesses a binary label $\ell \in \{0, 1\}^n$ and we have to find a $hv-$convex subset of $1-$nodes. Figure 6.9 shows an example.

6.2.1 P^8 scans to reconstruct hv-polyominoes

Let A be a $hv-$convex matrix of dimension $n \times m$; in this section we aim at finding the minimal number of surgical probes to get a faithful reconstruction of A using the projection matrix P^8.

We define the element a_{ij} to be *internal* if $i \notin \{1, n\}$ and $j \notin \{1, m\}$; it is a *border element* otherwise.

The following lemma allows determining the entries' values of the internal elements of A from its projection matrix P.

Lemma 27. *Let P^8 be an instance of MSP$_{hv}$ problem. Let (i, j) be an internal position of the related solution, then $a_{ij} = 0$ if and only if there is an entry $(i', j') \in N^8(i, j)$ such that $p^8_{i',j'} = 0$.*

Proof. $(\Leftarrow)$ Assume that there is $(i', j') \in N^8(i, j)$ such that $p_{i',j'} = 0$. It follows that $a_{ij} = 0$.

$(\Rightarrow)$ Let us proceed by contradiction assuming $a_{ij} = 0$, and $p^8_{i',j'} \geq 1$ for all $(i', j') \in N(i, j)$. It follows that from $p^8_{i-1,j-1} \geq 1$, $p^8_{i,j-1} \geq 1, p^8_{i+1,j-1} \geq 1$ and hv-convexity constraints we have either $a_{i,j-1} = 1$ or $a_{i,j-2} = 1$.

A similar reasoning applied to $p^8_{i-1,j+1}, p^8_{i,j+1}$ and $p^8_{i+1,j+1}$ imply that either $a_{i,j+1} = 1$ or $a_{i,j+2} = 1$. Finally, the convexity implies $a_{ij} = 1$, a contradiction.

□

Figure 6.10 shows an example of the partial reconstruction of hv-matrix using the previous lemma. Note that we adopt a graph representation of the matrix A in such a way the projections matrix contains exactly the sum of the neighbours' labels. The orange nodes are the positions whose values are not yet determined while yellow nodes represent 1-entries and grey nodes 0-entries.

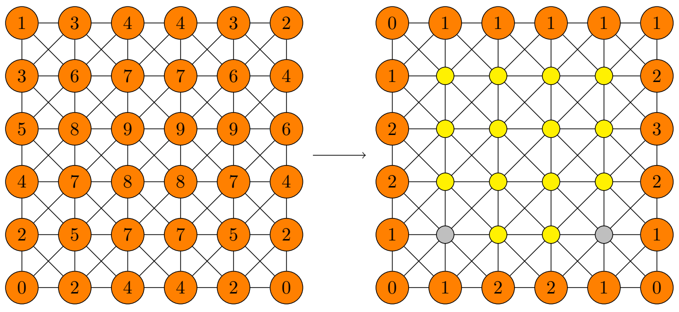


Figure 6.10: An example of application of Lemma 27. The 1-elements of the discrete set A in $\mathbb{Z}^2$ are in yellow, the 0-elements are in grey, while in orange are the unknown elements. The updated projections on each of them are computed on the right part.

After the application of Lemma 27, it remains to retrieve the labels of the border nodes. It is here that some ambiguities may occur, requiring surgical probing.

To accomplish the reconstruction task, the idea is to use the updated projections of the border entries after subtracting the values of the already computed internal nodes.

Algorithm 8 is the main routine to detect the positions where a surgical probe has to be performed, if needed, to reach the final faithful reconstruction of a $n \times m$ matrix A, with $n, m \geq 5$ from a projection matrix P^8. As subroutines, it uses Algorithms 6 and 7 hereafter described. Using Lemma 27, Algorithm 7 proceeds in reconstructing the border nodes values. We initially set each border element $a_{ij} = -1$ meaning that its value is yet unknown. We denote with $d(i,j) = |N^8(i,j)| - 1$ without (i,j) itself (i.e. $d(i,j)$ is the degree of node v_{ij}). $N(i,j)_k$ is the number of k-nodes (i.e. the number of nodes with label k), with $k \in \{-1, 0, 1\}$, in the neighbour of the node (i,j).

In order to complete the reconstruction, different strategies are adopted according to the length of the sides. In particular, Algorithm 6 performs the border reconstruction in case it has dimension 6 since some special cases appear. We treat the consecutive border elements of A as a 1×6 matrix A' having scan P'.

Algorithm 6 Rec-6-line

```
Require: an unknown $1 \times 6$ matrix $A'$ and its probe $P'$
Ensure: some entries of $A'$ compatible with $P'$
  if $p'_{1,3} = 1$ and $p'_{1,4} = 1$ then
    $a'_{1,i} = 0, i \in \{1, 2, 5, 6\}$
  else if $p'_{1,3} = 1$ and $p'_{1,4} = 2$ then
    $a'_{1,i} = 0, i \in \{1, 2, 3\}$
    $a'_{1,i} = 1, i \in \{4, 5\}$
  else if $p'_{1,3} = 2$ and $p'_{1,4} = 1$ then
    $a'_{1,i} = 0, i \in \{4, 5, 6\}$
    $a'_{1,i} = 1, i \in \{2, 3\}$
  else if $p'_{1,3} = 2$ and $p'_{1,4} = 2$ then
    $a'_{1,i} = 0, i \in \{1, 2, 5, 6\}$
    $a'_{1,i} = 1, i \in \{3, 4\}$
  end if
  Return: $A'$
```

Algorithm 7 is the core of the reconstruction of A: first, it computes the values of its internal elements, then it moves to those in the border. If n or m have dimension 6, then it runs *Rec-6-line* on them; otherwise, it starts computing the unknown entries of A recursively, based on the updated probes of P^8, until no further changes are possible. If some ambiguities remain, then surgical probes are required.

Algorithm 7 P^8-Rec

```
Require: an unknown $n \times m$ binary matrix $A$ and its scan matrix $P^8$
Ensure: some values of the elements of $A$ compatible with $P^8$
 1: Apply Lemma 27 to a $n \times m$ matrix $A$ and set to $-1$ its non detected (border) entries

 2: Compute $\overline{P}$ as $\overline{p}_{ij} = p^8_{ij} - |N(i,j)_1|$, with $1 \leq i \leq n$ and $1 \leq j \leq m$
 3: if $n = 6$ or $m = 6$ then
 4:     Apply *Rec-6-line* to the borders of $A$ of length 6 and update $\overline{P}$ accordingly
 5: end if
 6: check = TRUE
 7: while check do
 8:     check = FALSE
 9:     Update $\overline{p}_{ij} = \overline{p}_{ij} - |N(i,j)_1|$, with $1 \leq i \leq n$ and $1 \leq j \leq m$
10:     for $1 \leq i \leq n$ and $1 \leq j \leq m$ do
11:         if $\overline{p}_{ij} = |N(i,j)_{-1}|$ then
12:             check = TRUE
13:             update $a_{k,t} = 1$ for each $(k,t) \in N(i,j)_{-1}$
14:         else if $\overline{p}_{ij} = 0$ then
15:             check = TRUE
16:             update $a_{k,t} = 0$ for each $(k,t) \in N(i,j)_{-1}$
17:         else if $\overline{p}_{ij} = 2$ and $i \notin \{1,2,n-1,n\}$ and $j \notin \{1,2,m-1,m\}$ then
18:             check = TRUE
19:             $a_{ij} = 1$
20:         end if
21:     end for
22:     if check == FALSE then
23:         break
24:     end if
25: end while
```

Figure 6.11 shows the results of one iteration of Algorithm 7, starting from the probe of Figure 6.10, on the left.

Lemma 28. *Algorithm 7,* P^8-Rec, *computes the values of all the entries of a $n \times m$ binary matrix A from its scan P^8, if it exists, except for at most the four corners and their neighbours.*

Proof. At first, Algorithm 7 applies Lemma 27 to retrieve all the internal entries of A.

Algorithm 8 Probe-Detect

```
Require: an unknown binary matrix $A$ and its scan $P^8$
Ensure: the entries' values of $A$, and the number of the needed surgical probes
  check = TRUE
  while check do
    Run $P^8$-Rec on inputs $A$ and $\overline{P}^8$
    if for each $1 \leq i \leq n$ and $1 \leq j \leq m$, it holds $a_{ij} \neq -1$ then
      check = FALSE
    else
      Do a surgical probe in $(i, j)$ such that $a_{ij} = -1$ and $d(i, j) = 5$ (i.e., $(i, j)$ is not a corner element)
    end if
  end while
```

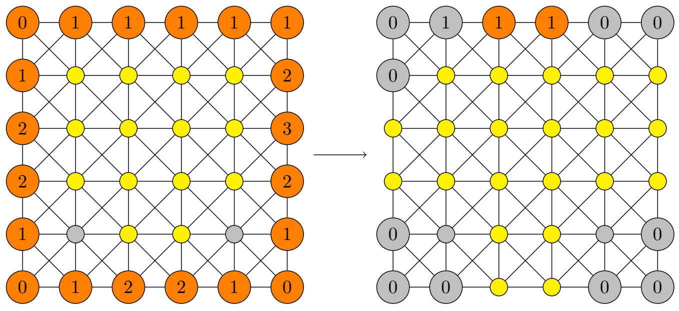


Figure 6.11: Starting from the situation in Figure 6.10, we show the results after one application of Algorithm 7. The successive iteration of the algorithm returns $a_{1,3} = 1, a_{1,4} = 0$.

Let us first assume w.l.o.g. that $n = 6$. Algorithm 6, runs *Rec-6-line* on the border of A of length 6. Maintaining the notation A' and P' for a border and its scans, as in *Rec-6-line*, the following cases hold:

- if $p'_{1,3} = p'_{1,4} = 1$, then the only 1-entry must lie between $a'_{1,3}$ and $a'_{1,4}$. Therefore all the other nodes still are unknown and so have value -1;
- if $p'_{1,3} = 1$ and $p'_{1,4} = 2$, then, by convexity, we know that $a'_{1,3} = 1$, therefore $a'_{1,4} = a'_{1,5} = 1$. The entry $a'_{1,6}$ remains unknown;
- if $p'_{1,3} = 2$ and $p'_{1,4} = 1$ a symmetric result holds;
- if $p'_{1,3} = p'_{1,4} = 2$, by convexity we infer that $a'_{1,3} = a'_{1,4} = 1$ and $a'_{1,5} = 0$.

A simple check confirms that the thesis holds in this case.

Now, let us consider $n > 6$. By the hv-convexity of A it holds that along the border of length n it lies a sequence of consecutive values 1. If those values are 3 or more, then there exists an entry of $\overline{P}^8$ (as defined in Algorithm 7) on the border having value 3, so P^8-*Rec* inserts the three values 1.

On the other hand, if there are less than three consecutive elements 1, then there are three consecutive values 0, so P^8-*Rec* inserts three entries 0.

In both cases, the remaining values of the border are obtained in successive updates of $\overline{P}^8$.

Finally, the cases with n less than 6 are treated in Section 6.2.2 by exhaustive computation. We will underline that the only unknown values of A lie according to the thesis of the lemma.

□

Theorem 25. *If P^8 is a scan whose dimensions are greater than 5, then P^8-rec reconstructs the related hv-polyomino A without surgical probes.*

The proof follows directly from the reconstruction provided in Lemma 28 when the dimensions of the input scan P^8 are greater than five.

On the other hand, the exhaustive computation presented in Section 6.2.2 on smaller cases shows that at most four different configurations may share the same projections, and, to determine them, only two surgical probes are needed.

The following example shows the full reconstruction for a 5×5 hv-convex matrix that requires one surgical probe. Follow the steps of the reconstruction in Fig. 6.12 where the 1-entries are colored in yellow, 0 entries are in grey, and -1 entries are in orange.

Example 15. Consider the scan matrix F^8 in Fig.6.12, (a). By applying P^8-*Rec* we obtain the matrix in Fig.6.12, (b), where the border entries are those of the updated $\overline{P}^8$ matrix.

Note that P^8-*Rec* also detects the values of $a_{1,3} = a_{3,5} = a_{5,3} = a_{3,1} = 1$. Then it performs a surgical probe in one border position with degree equal to 5 (i.e., the red circle in Fig.6.12, (b) and it avoids corner nodes). Further runs of P^8-*Rec* lead to the final solution.

6.2.2 Small cases for P^8

In this section, we present the results of the exhaustive computation of the hv-matrix sharing the same scan, and the required minimal surgical probes needed to determine them. The interested reader can use the *Matlab* code provided in the Appendix to reply the tests. We start from those of dimension 5×5: Figure 6.13 shows, row by row, all the different couples sharing the same scans, up to rotations and symmetries. So, one surgical probe is needed to determine the elements of the couples.

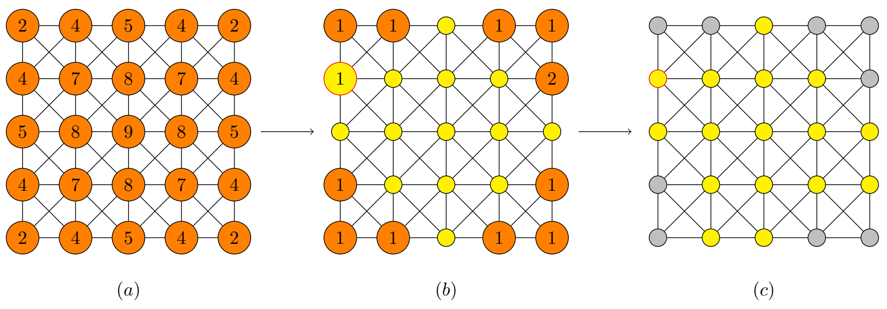


Figure 6.12: Left, example of projections of a matrix A. Middle, reconstruction obtained applying Algorithm 7. A surgical probe is highlighted in red. Right, final reconstruction of the labels.

The exhaustive generation of the smaller cases provides at most four different configurations that may share the same scans, as shown in Fig. 6.14, last row, for the cases of dimension 2×3. In those cases, the characterization of each matrix can be performed after two surgical probes at most.

The following theorem holds:

Theorem 26. *If a scan P^8 has one dimension lower than 6, then P^8-rec uniquely reconstructs the related hv-polyomino A from P^8 with at most two surgical probes.*

6.2.3 P^4 scans to reconstruct hv-polyominoes

We highlight that in this section we keep the notation and the notion introduced in the previous section. Again, we indicate with A the $hv-$convex matrix to reconstruct.

In particular, here we aim to find the minimal number of surgical probes to get a faithful reconstruction of A using the projection matrix P^4.

Note that Lemma 27 can be adapted to the case of interest and the proof is similar.

Lemma 29. *Let P^4 be an instance of MSP$_{hv}$ problem. Let (i, j) be an internal position of the related solution, then $a_{ij} = 0$ if and only if there is an entry $(i', j') \in N^4(i, j)$ such that $p^4_{i',j'} = 0$.*

Lemma 29 allows us to uniquely detect the internal elements of the matrix A. Figure 6.15 shows an example of the use of Lemma 29 to detect internal elements.

Moving to the border entries, we realize that their detection requires small changes in Algorithm 7 P^8-*Rec* that is updated to Algorithm 9, P^4-*Rec* where the notation introduced in the previous section is maintained. Algorithm 6 *Rec-6-line* and Algorithm 8 *Probe-Detect* are preserved, up to updating P^8-*Rec* with P^4-*Rec*.

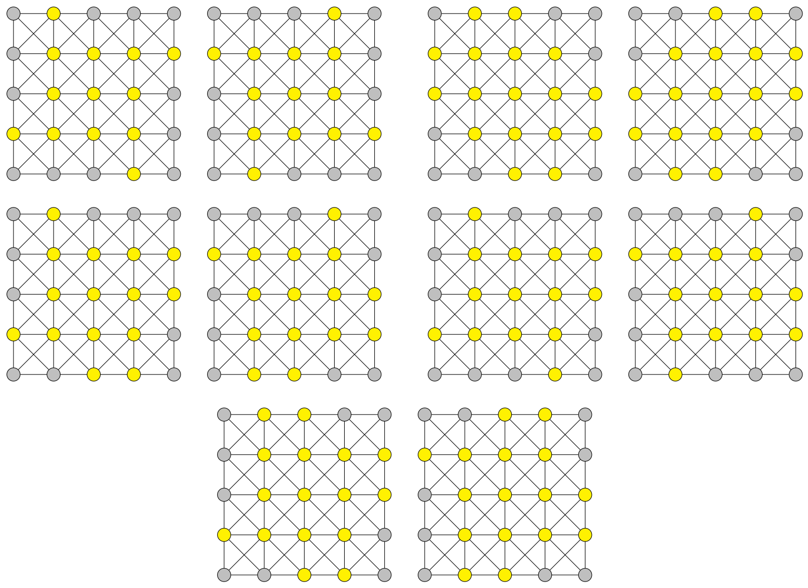


Figure 6.13: In each row represents polyominoes sharing the same projections. Note that, to uniquely determine one of them, at most one surgical probe is needed.

Algorithm 9 P^4-Rec

```
Require: an unknown $n \times m$ binary matrix $A$ and its scan matrix $P^4$
Ensure: some values of the elements of $A$ compatible with $P^4$
  1 : 16 as in Algorithm 7
  17: else if $\bar{p}_{ij} = 2$ and $i \notin \{1, n\}$ and $j \notin \{1, m\}$ then
  18 : 25 as in Algorithm 7
```

The main difference between P^8-Rec and P^4-Rec emerges in the border entries whose value is 2. This is due to the fact that the scans of the positions close to a corner do not include each other anymore.

Theorem 27. *If P^4 is a scan whose dimensions are greater than 4, then P^4-rec reconstructs the related hv-polyomino A without surgical probes.*

Proof. The result easily follows considering that Algorithm 9 works for all the cases with $n, m \geq 6$, since there are no differences with Algorithm 7 in this case. If $n = 5$ or $m = 5$ we can reconstruct the border without surgical probes: w.l.g. let us reconstruct the first row of A. The knowledge of $p_{2,3}$ and all the internal elements of A allows us to get $a_{1,3}$. Updating the scan and using P^4-*Rec*, we have, for $i \in \{2, 4\}$, that $p_{1,i} = 2$ implies $a_{1,i} = 1$, $a_{1,i} = 0$ otherwise. The values $a_{1,i}$ with $i \in \{1, 5\}$ can be obtained by difference, and the thesis follows.

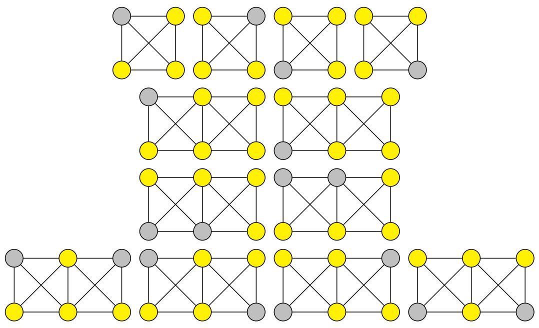


Figure 6.14: Each row group matrices share the same projections according to a square window. At most 2 surgical probes are needed to determine

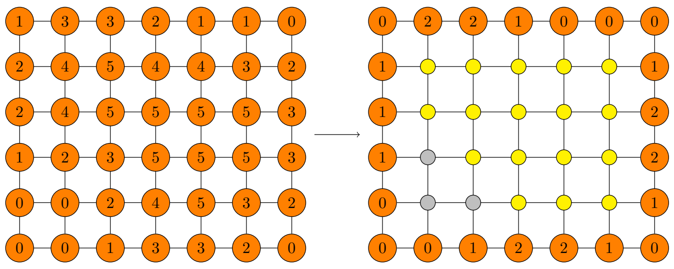


Figure 6.15: An example of application of Lemma 29. 1-nodes are highlighted in yellow, $0-$nodes in grey and unknown nodes in orange. Note that updated projections are computed on the right.

□

Example 16. Let us consider the probe P^4 shown in Fig. 6.16, left. P^4-Rec computes the internal entries of the related matrix A and some elements of each border, according to the proof of Theorem 27. In Fig. 6.16, center, there are the updated scans. Finally, in Fig. 6.16, right, the whole matrix is reconstructed.

6.2.4 Small cases for P^4

In this section, we exhaustively study the hv-polyominoes whose dimensions are smaller than five and that share the same P^4 scans in order to solve the related MSP_{hv} problem. The interested reader can use the *Matlab* code provided in the Appendix to reply the tests. Again, the computation reveals that some matrices require surgical probes. As an example, Fig. 6.17 shows two couples of them.

Furthermore, no more than two different hv-polyominoes share the same P^4 scans, so one surgical probe is required to determine them. The following theorem holds.

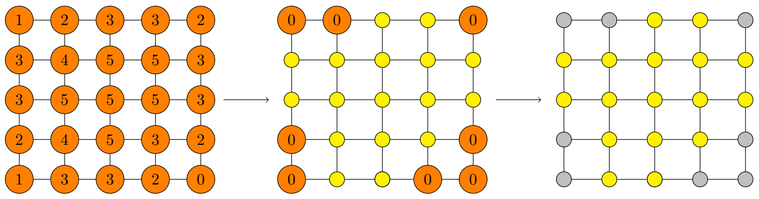


Figure 6.16: The steps performed by P^4-*Rec* leading to the reconstruction of a hv-polyomino from the P^4 scan on the left. Since the dimensions of P^4 are greater than 4, then no surgical probes are needed.

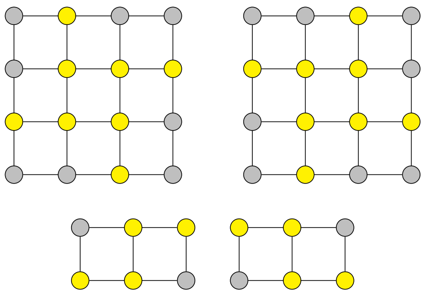


Figure 6.17: Two couples of hv-polyominoes sharing the same P^4 scan. In both cases, only a single surgical probe is needed to determine them.

Theorem 28. *If a scan P^4 has one dimension lower than 5, then P^4-*rec *uniquely reconstructs the related hv-polyomino A from P^4 with at most one surgical probe.*

Bibliography

[1] Claude Berge. Graphs and hypergraphs. 1973.

[2] Michael R Garey, David S. Johnson, and R Endre Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

[3] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[4] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010.

[5] Claude Berge. *Hypergraphs: combinatorics of finite sets*, volume 45. Elsevier, 1984.

[6] Gabriel Deza and Shmuel Onn. Optimization over degree sequences of graphs. *Discrete Applied Mathematics*, 296:2–8, June 2021.

[7] Fethi Jarray. Solving problems of discrete tomography: application in workforce scheduling. *4OR*, 3:337–340, 2005.

[8] Fethi Jarray, Marie-Christine Costa, and Christophe Picouleau. Complexity results for the horizontal bar packing problem. *Information processing letters*, 108(6):356–359, 2008.

[9] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS computational biology*, 5(5):e1000385, 2009.

[10] Emad Ramadan, Arijit Tarafdar, and Alex Pothen. A hypergraph model for the yeast protein complex network. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 189. IEEE, 2004.

[11] Max Falkenberg, Alessandro Galeazzi, Maddalena Torricelli, Niccolò Di Marco, Francesca Larosa, Madalina Sas, Amin Mekacher, Warren Pearce, Fabiana Zollo, Walter Quattrociocchi, et al. Growing polarization around climate change on social media. *Nature Climate Change*, pages 1–8, 2022.

[12] Matteo Cinelli, Gianmarco De Francisci Morales, Alessandro Galeazzi, Walter Quattrociocchi, and Michele Starnini. The echo chamber effect on social media. *Proceedings of the National Academy of Sciences*, 118(9):e2023301118, 2021.

[13] Alexandru T Balaban. Applications of graph theory in chemistry. *Journal of chemical information and computer sciences*, 25(3):334–343, 1985.

[14] Michael D Koenig and Stefano Battiston. From graph theory to models of economic networks. a tutorial. *Networks, topology and dynamics*, 613:23–63, 2009.

[15] P. Erdős and T. Gallai. Graphs with prescribed degrees of vertices. *Math. Lapok*, 11:264–274, 1960.

[16] A. K. Dewdney. Degree sequences in complexes and hypergraphs. *Proceedings of the American Mathematical Society*, 53(2):535–540, 1975.

[17] Sarah Behrens, Catherine Erbes, Michael Ferrara, Stephen G. Hartke, Benjamin Reiniger, Hannah Spinoza, and Charles Tomlinson. New results on degree sequences of uniform hypergraphs. *The Electronic Journal of Combinatorics*, 20(4), November 2013.

[18] Srecko Brlek and Andrea Frosini. A tomographical interpretation of a sufficient condition on h-graphical sequences. In *Discrete Geometry for Computer Imagery*, pages 95–104. Springer International Publishing, 2016.

[19] Andrea Frosini, Christophe Picouleau, and Simone Rinaldi. On the degree sequences of uniform hypergraphs. In *Discrete Geometry for Computer Imagery*, pages 300–310. Springer Berlin Heidelberg, 2013.

[20] Andrea Frosini, Christophe Picouleau, and Simone Rinaldi. New sufficient conditions on the degree sequences of uniform hypergraphs. *Theoretical Computer Science*, 868:97–111, May 2021.

[21] G.T. Herman and A. Kuba, editors. *Discrete tomography Foundations algorithms and applications*. Birkhäuser, 1999.

[22] Gabor T. Herman and Attila Kuba, editors. *Discrete Tomography*. Birkhäuser Boston, 1999.

[23] William Kocay and Pak Ching Li. On 3-hypergraphs with equal degree sequences. *Ars Combinatoria*, 82:145–158, 2007.

[24] Svatopluk Poljak, Vojtěch Rödl, and Daniel TURZiK. Complexity of representation of graphs by set systems. *Discrete Applied Mathematics*, 3(4):301–312, 1981.

[25] Amotz Bar-Noy, Toni Böhnlein, Zvi Lotker, David Peleg, and Dror Rawitz. The generalized microscopic image reconstruction problem. *Discrete Applied Mathematics*, 321:402–416, 2022.

[26] J. M. Carazo, C. O. Sorzano, E. Rietzel, R. Schröder, and R. Marabini. *Discrete Tomography in Electron Microscopy*, pages 405–416. Birkhäuser Boston, Boston, MA, 1999.

[27] K.J. Batenburg, S. Bals, J. Sijbers, C. Köbel, P.A. Midgley, J.C. Hernandez, U. Kaiser, E.R. Encina, E.A. Coronado, and G. Van Tendeloo. 3d imaging of nanomaterials by discrete tomography. *Ultramicroscopy*, 109(6):730–740, 2009.

[28] Sandra Van Aert, Kees J. Batenburg, Marta D. Rossell, Rolf Erni, and Gustaaf Van Tendeloo. Three-dimensional atomic imaging of crystalline nanoparticles. *Nature*, 470(7334):374–377, February 2011.

[29] R.J. Gardner, P. Gritzmann, and D. Prangenberg. On the computational complexity of reconstructing lattice sets from their x-rays. *Discrete Mathematics*, 202(1):45–71, 1999.

[30] Elena Barcucci, Alberto Del Lungo, Maurice Nivat, and Renzo Pinzani. Reconstructing convex polyominoes from horizontal and vertical projections. *Theoretical Computer Science*, 155(2):321–347, 1996.

[31] R Gardner and Peter Gritzmann. Discrete tomography: determination of finite sets by x-rays. *Transactions of the American Mathematical Society*, 349(6):2271–2295, 1997.

[32] Paolo Dulio, Andrea Frosini, Simone Rinaldi, Lama Tarsissi, and Laurent Vuillon. Further steps on the reconstruction of convex polyominoes from orthogonal projections. *Journal of Combinatorial Optimization*, 44(4):2423–2442, May 2021.

[33] Yan Gerard. Regular switching components. *Theoretical Computer Science*, 777:338–355, July 2019.

[34] Andrea Frosini and Maurice Nivat. Binary matrices under the microscope: A tomographical problem. *Theoretical Computer Science*, 370(1):201–217, 2007.

[35] Ignacio M Pelayo. *Geodesic convexity in graphs*. Springer, 2013.

[36] S. Brlek, J.O. Lachaud, X. Provençal, and C. Reutenauer. Lyndon + christoffel = digitally convex. *Pattern Recognition*, 42(10):2239–2246, 2009. Selected papers from the 14th IAPR International Conference on Discrete Geometry for Computer Imagery 2008.

[37] S. Brlek, J.O. Lachaud, X. Provençal, and C. Reutenauer. Lyndon christoffel digitally convex. *Pattern Recognition*, 42(10):2239–2246, October 2009.

[38] Herbert Freeman. On the encoding of arbitrary geometric configurations. *IEEE Transactions on Electronic Computers*, EC-10(2):260–268, June 1961.

[39] Monsieur Lothaire. *Combinatorics on words*, volume 17. Cambridge university press, 1997.

[40] J.-P. Borel and F. Laubie. Quelques mots sur la droite projective réelle. *Journal de théorie des nombres de Bordeaux*, 5(1):23–51, 1993.

[41] Niccolò Di Marco, Andrea Frosini, and William Lawrence Kocay. A study on the existence of null labelling for 3-hypergraphs. In Paola Flocchini and Lucia Moura, editors, *Combinatorial Algorithms*, pages 282–294, Cham, 2021. Springer International Publishing.

[42] Andrea Frosini, William L. Kocay, Giulia Palma, and Lama Tarsissi. On null 3-hypergraphs. *Discrete Applied Mathematics*, 303:76–85, November 2021.

[43] F. Harary. *Graph Theory*. Addison Wesley Publishing Company, 1972.

[44] Niccolò Di Marco, Andrea Frosini, William Lawrence Kocay, Elisa Pergola, and Lama Tarsissi. Structure and complexity of 2-intersection graphs of 3-hypergraphs. *Algorithmica*, 85(3):745–761, June 2022.

[45] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979.

[46] Anne Berry and Geneviève Simonet. Computing a clique tree with the algorithm maximal label search. *Algorithms*, 10(1):20, January 2017.

[47] Petr Hliněný and Jan Kratochvíl. Computational complexity of the krausz dimension of graphs. In Rolf H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science*, pages 214–228, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[48] Alan A. Bertossi. The edge hamiltonian path problem is NP-complete. *Information Processing Letters*, 13(4-5):157–159, January 1981.

[49] Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, November 1981.

[50] M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.

[51] M. R. Garey, D. S. Johnson, and R. Endre Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, December 1976.

[52] Maurice Nivat. Sous-ensembles homogénes de $\mathbb{Z}^2$ et pavages du plan. *Comptes Rendus Mathematique*, 335(1):83–86, 2002.

[53] D. Battaglino, A. Frosini, and S. Rinaldi. A decomposition theorem for homogeneous sets with respect to diamond probes. *Computer Vision and Image Understanding*, 117(4):319–325, 2013. Special issue on Discrete Geometry for Computer Imagery.

[54] Amotz Bar-Noy, Toni Böhnlein, Zvi Lotker, David Peleg, and Dror Rawitz. Weighted microscopic image reconstruction. In Tomáš Bureš, Riccardo Dondi, Johann Gamper, Giovanna Guerrini, Tomasz Jurdziński, Claus Pahl, Florian Sikora, and Prudence W.H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science*, pages 373–386, Cham, 2021. Springer International Publishing.

[55] Toni Böhnlein, Niccolò Di Marco, and Andrea Frosini. Minimum surgical probing with convexity constraints. In *Lecture Notes in Computer Science*, pages 136–147. Springer Nature Switzerland, 2023.

[56] Niccolò Di Marco and Andrea Frosini. The generalized microscopic image reconstruction problem for hypergraphs. In *Lecture Notes in Computer Science*, pages 317–331. Springer International Publishing, 2023.

[57] Claude Berge. Hypergraphs. page 256, 1989.

[58] Robert Janczewski, Pawel Obszarski, and Krzysztof Turowski. Weighted 2-sections and hypergraph reconstruction. *Theoretical Computer Science*, 915:11–25, 2022.

[59] Niccolò Di Marco and Andrea Frosini. Properties of SAT formulas characterizing convex sets with given projections. In *Lecture Notes in Computer Science*, pages 153–166. Springer International Publishing, 2022.

[60] Alberto Del Lungo, Maurice Nivat, and Renzo Pinzani. The number of convex polyominoes reconstructible from their orthogonal projections. *Discrete Mathematics*, 157(1):65–78, 1996.

[61] Paolo Dulio and Andrea Frosini. On some geometric aspects of the class of hv-convex switching components. In Joakim Lindblad, Filip Malmberg, and Nataša Sladoje, editors, *Discrete Geometry and Mathematical Morphology*, pages 299–311, Cham, 2021. Springer International Publishing.

[62] Paolo Dulio and Andrea Frosini. Characterization of hv-convex sequences. *Journal of Mathematical Imaging and Vision*, 64(7):771–785, May 2022.

[63] Yan Gerard. About the reconstruction of convex lattice sets from one or two x-rays. *arXiv preprint arXiv:2211.08091*, 2022.

[64] Andrea Frosini and Laurent Vuillon. Tomographic reconstruction of 2-convex polyominoes using dual horn clauses. *Theoretical Computer Science*, 777:329–337, July 2019.

[65] Jean Berstel. *Combinatorics on words: Christoffel words and repetitions in words*, volume 27. American Mathematical Soc., 2009.

Chapter 7

Appendix

Algorithms for null label detection

In this section we provide *Matlab* code that deals with the null label problem. In particular, the following functions are provided:

- *dual hyp*, computes the dual of a $3-$hypergraph;
- *find_null_label*, compute (if exists) the null label of a $3-$hypergraph;
- *find_null_label_rand*, search randomly the null label of a $3-$hypergraph. It is useful when dealing with bigger $3-$hypergraphs;

Finally, we provide an example in which $3-$hypergraphs are generated and the existence of a null label is tested in each of them. The interested reader can check that Conjecture 1 is satisfied for $n \leq 7$.

```
function [Ad] = dual_hyp(A)
% This function construct the dual of the 3-hypergraph
   described by the edge list A. The dual of a
   hypergraph A is the hypergraph B whose edges are the
   complementary of the edges of A.

n = length(unique(A));
arc = nchoosek((1:n),3);

Ad = setdiff(arc,sort(A,2),'rows');

end
```

```
function [null_label] = find_null_label(A,n)
% This function find (if exists) a null label for the 3-
   hypergraph described by the edge list A. It generates
    all the unique permutations and then applied them in
    order to find it.

null_label = 0;
ed = size(A,1); % number of edges
label = ones(1,ed);
label(fix((ed/2))+1:end) = 0; % half elements are -1

% generate unique permutations
k = nnz(label);
C = nchoosek(1:ed,k);
m = size(C,1);
L = zeros(m,ed);

for ix = 1:m
    L(ix,C(ix,:)) = 1;
end

L(L == 0) = -1;

for i = 1:m

    label = L(i,:);
    B = incidence_matrix_hyp(A,n);

    if isequal(B'*label', zeros(n,1))

        null_label = label;
        break

    end

end
```

```
function [null_label] = find_null_label_rand(A,n,iter)
% This function try to find a null label for the 3-
   hypergraph described by the edge list A. It generates
    random permutations and then applied them in order
   to find it. The users must give a maximum number of
   iterations.

null_label = 0;
ed = size(A,1); % number of edges

% generate unique permutations
k = ed/2;

count = 0;
value = 0;

while(value == 0 && count <= iter)

    comb = randperm(ed,k);
    n_label = -ones(1,ed);
    n_label(comb) = 1;
    count = count + 1; % number of null label verified

    if isequal(A'*n_label', zeros(n,1))

        null_label = label;
        break

    end

end

% Generate 3-hypergraphs and check if they are null

n = input("Choose the number of nodes: ");
m = input("Choose the number of edges: ");
A = nchoosek(1:n,3);
```

```
n = length(unique(A));
c_arc = size(A,1);
all1 = nchoosek(1:c_arc,m);

for k = 1:size(all1,1) % all the hypergraph

    B = A(all1(k,:),:);

    H = incidence_matrix_hyp(B,n);
    deg = sum(H);

    if  ~isequal(zeros(1,size(H,2)),mod(deg,2))

        nl = find_null_label(B,n);

        if nl == 0

            disp('B has no null label!')
            B

        else

            disp('B has a null label!')
            B
            nl

        end

    end

end
```

Reconstruction of Convex Polyominoes

In this section, we perform a *Matlab* implementation of the algorithm for the reconstruction of (full) convex polyominoes. The algorithm, called *reconstruct_polyomino* starts with the horizontal and vertical projections of a (convex) polyomino P. Following the strategy presented in [30], our reconstruction procedure performs in two steps: first, it applies the filling operations but, differently from the standard approaches, after each iteration the convex hull is computed and points belonging to it are added into the kernel. After the first part, the output is a matrix with elements in $\{-1, 0, 1\}$. $0-$elements are not belonging to P, while $1-$elements are belonging for sure to P. The remaining elements are ambiguous and belong to the switching components.

To compute the Convex Hull, the procedure described in [36] is followed. In particular, the Algorithm relies on the fundamental Property 1, presented in Chapter 2.

The function *getborder* has been obtained from MathWorks (`https://it.mathworks.com/matlabcentral/fileexchange/12303-getborder`). All the functions used to perform *reconstruct_polyomino* are presented with a description provided before the code.

```
function wd = word_rotation(wd,i)
% apply the rotation of 90 degrees to the characters of
   w.

if nargin == 1

    i = 1;

end

for j = 1:i

    e = wd == 'e';
    n = wd == 'n';
    w = wd == 'w';
    s = wd == 's';

    wd(e) = 's';
    wd(n) = 'e';
    wd(w) = 'n';
    wd(s) = 'w';

end

end
```

```
function wd = word_rotation(wd,i)
% Apply the rotation of 90 degrees to the characters of
   w.

if nargin == 1

    i = 1;

end

for j = 1:i

    e = wd == 'e';
    n = wd == 'n';
    w = wd == 'w';
    s = wd == 's';

    wd(e) = 's';
    wd(n) = 'e';
    wd(w) = 'n';
    wd(s) = 'w';

end

end

function [w,P] = NWConvexHull(w)
% The word w represents the NW border of a polyomino.
% This function computes the vertices belonging the the
   convex hull of w
% and then modifies w in order to obtain a convex border
   .
%
% INPUT
% w : word that identifies the border;
%
% OUTPUT
% w : final (convex) border;
% P : stack containing the position and direction of
   each vertex belonging
% to the convex hull of w.

w = convertStringsToChars(w);
P = [];
n = length(w);

for i = 1:n

    u.pos = i;

    switch w(i)

        case 'e'

            u.a = 0; u.b = 1;

        case 'n'

            u.a = 1; u.b = 0;

        otherwise

            error('The path is not the border of a
               polyomino')

    end

    l = 1;

    while(l && ~isempty(P))

        v = P(1);

        if v.a*u.b <= v.b*u.a
```

```
            u.pos = v.pos;
            u.a = u.a + v.a;
            u.b = u.b + v.b;
            P(1) = [];

        else

            l = 0;

        end

    end

    P = [u P];

end

% fill the interior

position = zeros(1,length(P)+1); % position of the
   vertices

for k = length(P):-1:1

    position(length(P)-k+1) = P(k).pos;

end

position(length(P)+1) = length(w) + 1;
position = position - 1; % position in the string

for j = length(P):-1:1

    use = christoffel(P(length(P)-j+1).b,P(length(P)-j
       +1).a);
    w = replaceBetween(w,position(j)+1,position(j+1),use
        );

end

end

function P = ConvexHull(P)
% Compute the convex hull of the polyomino P trough
   christoffel words.

P = logical(P);
lb = getborder(P,'inside');
[n,m] = size(lb);

% Find the four feet

W = find(lb(:,1),1,'last');

N = find(lb(1,:),1,'first');

E = find(lb(:,m),1,'first');

S = find(lb(n,:),1,'last');

% compute the borders

NW = lb(1:W,1:N);
w1 = NWBorderToWord(NW);
w1 = NWConvexHull(w1);

NE = lb(1:E,N:end);
NE = rot90(NE);
w2 = NWBorderToWord(NE);
w2 = NWConvexHull(w2);
w2 = word_rotation(w2);

ES = lb(E:end,S:end);
ES = rot90(ES,2);
w3 = NWBorderToWord(ES);
w3 = NWConvexHull(w3);
w3 = word_rotation(w3,2);

SW = lb(W:end,1:S);
SW = rot90(SW,3);
w4 = NWBorderToWord(SW);
w4 = NWConvexHull(w4);
w4 = word_rotation(w4,3);

w = strcat(w1,w2,w3,w4);

lb = WordToBorder(w,W,n,m);
P(lb) = 1;

end

function [rejected] = isNW_Convex(w)
% Verify if the border definied by the word w is NW-
   convex.
% The alphabet of w must be {'e','n'}.

w = convertStringsToChars(w);
n = length(w);

index = 1;
rejected = 0;
while ~rejected && index <= n

    [l1,n1] = FirstLyndonFactor(w(index:n));
    rejected = ~(IsChristoffelPrimitive(l1));
    index = index + n1*length(l1);

end

rejected = ~rejected;
```

```
end

function w = NWBorderToWord(b)
% Starting from the border b, this function converts the
    border (given as a
% logical matrix) to its relative word, using alphabet
   {'e','n'}.

w = '';
W = find(b(:,1),1,'last');
N = find(b(1,:),1,'first');

initial = [W,1]; % initial position of the border
final = [1,N]; % final position
aux = initial;

while ~isequal(aux,final)

    if aux(1) ~= 1

        if b(aux(1)-1,aux(2)) % nord

            aux(1) = aux(1) - 1;
            w = strcat(w,'n');

        elseif b(aux(1),aux(2)+1) % east

            aux(2) = aux(2) + 1;
            w = strcat(w,'e');

        else

            error('Something is wrong in the border')

        end
    else % first row

        % only est is available

        if b(aux(1),aux(2)+1)

            aux(2) = aux(2) + 1;
            w = strcat(w,'e');

        else

            disp('Error on the board of P')
            return

        end

    end

end

end

function value = IsConvex(P)
% Verify if the polyomino P is convex trough a string
   algorithm.

value = 1;
[n,m] = size(P);
lb = getborder(logical(P),'inside');

W = find(lb(:,1),1,'last');

N = find(lb(1,:),1,'first');

E = find(lb(:,m),1,'first');

S = find(lb(n,:),1,'last');
```

```
% compute the NW border

NW = lb(1:W,1:N);
w1 = NWBorderToWord(NW);

if ~isNW_Convex(w1)

    value = 0;

end

NE = lb(1:E,N:end);
NE = rot90(NE);
w2 = NWBorderToWord(NE);

if ~isNW_Convex(w2)

    value = 0;

end
ES = lb(E:end,S:end);
ES = rot90(ES,2);
w3 = NWBorderToWord(ES);

if ~isNW_Convex(w3)

    value = 0;

end

SW = lb(W:end,1:S);
SW = rot90(SW,3);
w4 = NWBorderToWord(SW);

if ~isNW_Convex(w4)
```

```
    value = 0;

end

end

function value = check_condition(r,h,v,i)
% Given a single row (or column) of a polyomino and its
   projections, the
% function checks if the projections conditions are
   satisfied.

value = 1;

[n,m] = size(r);

if n > 1 && m == 1

    if sum(r == 1) > v(i)

        value = 0;

    end

elseif n == 1 && m > 1

    if sum(r == 1) > h(i)

        value = 0;

    end

end

function [P,alpha,beta,flag] = filling_operations(P,i,
   row_or_col,h,v)
% Compute the filling operations for reconstructing a hv
   -convex polyomino
```

```
% starting from P and its vertical and horizontal
   projections.

flag = 1;

alpha = find(P == 1);
beta = find(P == 1 | P == -1);

if row_or_col == 0 % row

    r = P(i,:);

elseif row_or_col == 1 % col

    r = flip(P(:,i));

else

    error("Wrong row_or_col")

end

% kernel connecting

n = length(r);

aux_alpha = find(r == 1);
r(min(aux_alpha):max(aux_alpha)) = 1;
aux_alpha = find(r == 1);

if ~check_condition(r,h,v,i)

    flag = 0;
    return

end
```

```
% shell connecting

aux = find(r == 0);

if ~isempty(aux_alpha)

    for t = 1:length(aux)

        if aux(t) < min(aux_alpha)

            r(1:aux(t)) = 0;

        elseif aux(t) > max(aux_alpha)

            r(aux(t):end) = 0;

        end

    end

end

if ~check_condition(r,h,v,i)

    flag = 0;
    return

end

% coherence kernel

aux_alpha = find(r == 1);
aux_beta = find(r == 1 | r == -1); % beta
c = aux_beta(2:end)-aux_beta(1:end-1);
c = c(:);

% test if beta is connected
```

```
if isequal(c,ones(length(c),1)) && isempty(aux_alpha)

    if row_or_col == 0 % row

        r(max(min(aux_beta),max(aux_beta)-h(i)+1):min(
           min(aux_beta)+h(i)-1,max(aux_beta))) = 1;

    elseif row_or_col == 1 % col

        r(max(min(aux_beta),max(aux_beta)-v(i)+1):min(
           max(aux_beta),min(aux_beta)+v(i)-1)) = 1;

    end

elseif isequal(c,ones(length(c),1)) && ~isempty(
   aux_alpha)

    if row_or_col == 0 % row

        r(min(aux_alpha):min(min(aux_beta)+h(i)-1,max(
           aux_beta))) = 1;
        r(max(min(aux_beta),max(aux_beta)-h(i)+1):max(
           aux_alpha)) = 1;

    elseif row_or_col == 1 % col

        r(min(aux_alpha):min(min(aux_beta)+v(i)-1,max(
           aux_beta))) = 1;
        r(max(min(aux_beta),max(aux_beta)-v(i)+1):max(
           aux_alpha)) = 1;

    end

end

if ~check_condition(r,h,v,i)
```

```
    flag = 0;
    return

end

aux_alpha = find(r == 1);
aux = find(r == 0);

% coherence shell

if ~isempty(aux_alpha)

    if row_or_col == 0 % row

        aux = union(aux,1:max(aux_alpha)-h(i));
        aux = union(aux,min(aux_alpha)+h(i):n);

    elseif row_or_col == 1 % col

        aux = union(aux,1:max(aux_alpha)-v(i));
        aux = union(aux,min(aux_alpha)+v(i):n);

    end

end

if ~check_condition(r,h,v,i)

    flag = 0;
    return

end

r(aux) = 0;

% final assignation
```

```
if row_or_col == 0 % row

    P(i,:) = r;

elseif row_or_col == 1 % col

    P(:,i) = flip(r);

end

if check_condition(P,h,v) == 0

    flag = 0;

end

alpha = find(P == 1);
beta = find(P == 1 | P == -1);

function [P,exist] = inside_rec(Ptry,h,v)
% Function that works inside the
   reconstrunction_polyomino algorithm.
%
% INPUT
% Ptry : logical matrix representing the polyomino;
% h,v : desired projections.
%
% OUTPUT
% P : obtained polyomino after the (non-convex)
   reconstruction algorithm;
% exist : variable equal to 0 if the desired polyomino
   does not exist and
% is equal to 1 otherwise.

exist = 1;
[n,m] = size(Ptry);
```

```
alpha_old = find(Ptry == 1);
beta_old = find(Ptry == 1 | Ptry == -1);
alpha_new = 0;
beta_new = 0;
P = Ptry;

while(~isequal(alpha_old, alpha_new) || ~isequal(
   beta_old, beta_new))

    alpha_old = alpha_new;
    beta_old = beta_new;
    count_r = 1;
    count_c = 1;

    for tt = 1:n+m

        if tt <= n

            [Ptry,~,~,flag] = filling_operations(Ptry,
               count_r,0,h,v);

            if flag == 0

                break

            end

            count_r = count_r + 1;

        else

            [Ptry,~,~,flag] = filling_operations(Ptry,
               count_c,1,h,v);

            if flag == 0

                break
            end

            count_c = count_c + 1;

        end

    end

    if flag == 0

        break

    end

    alpha_new = find(Ptry == 1);
    beta_new = find(Ptry == 1 | Ptry == -1);

end

if flag == 0

    exist = 0;
    return

else

    for i = 1:n % rows

        r = Ptry(i,:);
        a = find(r == 1);
        b = find(r == -1 | r == 1);

        if length(a) > h(i) || length(b) < h(i)

            exist = 0;
            return
        end

    end

    for i = 1:m % rows

        r = Ptry(:,i);
        a = find(r == 1);
        b = find(r == -1 | r == 1);

        if length(a) > v(i) || length(b) < v(i)

            exist = 0;
            return

        end

    end

    P = Ptry;

end

end

function lb = WordToBorder(w,W,n,m)
% Conver a word in the alphabet {e,n,s,w} as the border
   of a polyomino.

lb = zeros(n,m);
lb(W,1) = 1;
aux = [W,1]; % actual position

for i = 1:length(w)

   switch w(i)
```

```
        case 'e'

            lb(aux(1),aux(2)+1) = 1;
            aux(2) = aux(2) + 1;

        case 'n'

            lb(aux(1)-1,aux(2)) = 1;
            aux(1) = aux(1) - 1;

        case 'w'

            lb(aux(1),aux(2)-1) = 1;
            aux(2) = aux(2)-1;

        case 's'

            lb(aux(1)+1,aux(2)) = 1;
            aux(1) = aux(1) + 1;

    end

end
end

function [w] = christoffel(x,y)

% Take as input a couple of numbers x and y and give as
   output the Christoffel Word of slope y/x.

check = 0;
w1=[];
j=0;

if x<y
```

```
    check=1;
    x1=y;
    y1=x;

else

    x1=x;
    y1=y;

end

for i=1:x1
    if (i*y1-j*x1)<x1

        w1=[w1 'e'];

    else

        w1=[w1 'e' 'n'];
        j=j+1;

    end

end

if check==1

    w1 = strrep(w1,'e','a');
    w1 = strrep(w1,'n','e');
    w1 = strrep(w1,'a','n');
    w1=fliplr(w1);

end

w=w1;

end
```

```
function P = ConvHull_Feet(P,i,j,k,t)
% Compute the convex hull starting from only the feet of
    the polyomino P.
%
% INPUT
% P : logical matrix with only feet = 1.
% i : initial position for the foot in the first column;
% j : initial position for the foot in the first row;
% k : initial position for the foot in the last column;
% t : initial position for the foot in the last row.
%
% OUTPUT
% P : convex hull of the polyomino P.

[n,m] = size(P);

% construct the borders
NW = christoffel(j-1,i-1);
lb = WordToBorder(NW,i,i,j);
P(1:i,1:j) = lb|P(1:i,1:j);

NE = christoffel(k-1,m-find(P(1,:),1,'last'));
lb = WordToBorder(NE,m-find(P(1,:),1,'last')+1,...
    m-find(P(1,:),1,'last')+1,k);
P(1:k,find(P(1,:),1,'last'):m) = rot90(lb,3)|P(1:k,find(
   P(1,:),1,'last'):m);


ES = christoffel(m-find(P(n,:),1,'last'),n-find(P(:,m)
   ,1,'last'));
lb = WordToBorder(ES,n-find(P(:,m),1,'last')+1,...
    n-find(P(:,m),1,'last')+1,m-find(P(n,:),1,'last')+1)
       ;
P(find(P(:,m),1,'last'):n,find(P(n,:),1,'last'):m) =
   rot90(lb,2)|...
    P(find(P(:,m),1,'last'):n,find(P(n,:),1,'last'):m);
```

```
SW = christoffel(n-find(P(:,1),1,'last'),t-1);
lb = WordToBorder(SW,t,t,n-find(P(:,1),1,'last')+1);
P(find(P(:,1),1,'last'):n,1:t) = rot90(lb,1)|P(find(P
   (:,1),1,'last'):n,1:t);

% fill the interior

for i = 1:n

    P(i,find(P(i,:),1,'first'):find(P(i,:),1,'last')) =
       1; % fill rows
    P(find(P(:,i),1,'first'):find(P(:,i),1,'last'),i) =
       1; % fill columns


end

end

function P = reconstruct_polyomino(h,v,conv,i,j,k,t)
% Function that reconstruct a polyomino based on the
   horizontal and
% vertical projections h and v.
% i,j,k,t are the (optional) position of the feet.
% If conv = 1 the function search for a (full) convex
   polyomino.

m = length(v); % number of columns
n = length(h); % number of rows

v = v(:)';
h = h(:);

P = -ones(n,m);

% check initial conditions
```

```
if ~isequal(v<=n,ones(1,m)) || ~isequal(h<=m,ones(n,1))

    error("Dimension does not match")

end

if sum(h) ~= sum(v)

    error("The sum of the projections isn't equal")

end

% place feet

Ptry = P;
control = 0;

if nargin <= 3

    for i = 1:(n-v(1)+1) % first column

        for j = 1:(m-h(1)+1) % first row

            for k = 1:(n-v(m)+1) % last column

                for t = 1:(m-h(n)+1) % last row

                    % place feet and projections

                    % first column

                    for u = i:i+v(1)-1

                        Ptry(u,1:h(u)) = 1;

                    end
```

```
% first row

for u = j:j+h(1)-1

    Ptry(1:v(u),u) = 1;

end

% last column

for u = k:k+v(m)-1

    Ptry(u,m:-1:(m-h(u)+1)) = 1;

end

% last row

for u = t:t+h(n)-1

    Ptry((n-v(u)+1):n,u) = 1;

end

if conv % apply convex hull

    Ptry = ConvHull_Feet(Ptry,i,j,k,
       t);

end

[Ptry,flag] = inside_rec(Ptry,h,v);

if conv && flag % apply again the
   reconstruction

    P = Ptry;
```

```
        while(~IsConvex(Ptry ==1))

            use = ConvexHull(Ptry == 1);
            Ptry(use) = 1;

            [Ptry,flag] = inside_rec(
               Ptry,h,v);
            P = Ptry;

            if ~flag

                control = 1;
                break

            end

        end

    elseif flag % already convex

        P = Ptry;

    end

    if ~flag

        control = 1;
        break

    end

end

if control

    break
```

```
                end

            end

            if control

                break

            end

        end

        if control

            break

        end

    end

else % chosen feet

    Ptry(i:i+v(1)-1,1) = 1;
    Ptry(1,j:j+h(1)-1) = 1;
    Ptry(k:k+v(m)-1,m) = 1;
    Ptry(n,t:t+h(n)-1) = 1;

    % first column

    for u = i:i+v(1)-1

        Ptry(u,1:h(u)) = 1;

    end

    % first row
```

```
for u = j:j+h(1)-1

    Ptry(1:v(u),u) = 1;

end

% last column

for u = k:k+v(m)-1

    Ptry(u,m:-1:(m-h(u)+1)) = 1;

end

% last row

for u = t:t+h(n)-1

    Ptry((n-v(u)+1):n,u) = 1;

end

% from here P is ready

[Ptry,flag] = inside_rec(Ptry,h,v);

if ~flag

    disp("The polyomino does not exists with those
       feet")
    return

elseif conv % apply again the reconstruction

    P = Ptry;
```

```
        while(~IsConvex(Ptry ==1))

            use = ConvexHull(Ptry == 1);
            Ptry(use) = 1;

            [Ptry,flag] = inside_rec(Ptry,h,v);
            P = Ptry;

            if ~flag

                disp("The polyomino does not exists with
                    those feet")
                return

            end

        end

    else % already convex

        P = Ptry;

    end

end
```

MSP for polyominoes

In this section we provide the *Matlab* code we used to test the small cases for the MSP problem on polyominoes. In particular, we used the following functions:

- *get_projections*, gets the projections of a matrix according to the type of neighbours defined by the user;
- *is_hv_convex*, test if a binary matrix A is $hv-$convex;
- *small_cases*, find the matrices sharing the same projections according to the type of neighbour defined by the user.

Finally, we provide a script that uses all the previous function and ask to the user to provide the dimensions of the matrix to test.

```
function P = get_projections(A,type)
% This function gets the projections of matrix A
   according to the neighbours defined in type. If type
   == 'S', then it is its 8-neighbourhood, if type == "D
   " then it is its 4-neighbourhood.

[n,m] = size(A);
P = zeros(n,m);

for i = 1:n

    for j = 1:m

        if strcmp(type,'S')

            P(i,j) = sum(A(max(1,i-1):min(n,i+1),max(1,j
               -1):min(m,j+1)),'all');

        elseif strcmp(type,'D')

            if (i > 1 && i < n && j > 1 && j < m)

                P(i,j) = A(i-1,j) + A(i,j) + A(i+1,j) +
                   ...
                    A(i,j-1) + A(i,j+1);

            elseif i == 1 && j == 1

                P(i,j) = A(1,1) + A(1,2) + A(2,1);

            elseif i == 1 && j == m

                P(i,j) = A(1,m) + A(1,m-1) + A(2,m);

            elseif i == n && j == 1
```

```
                P(i,j) = A(n,1) + A(n-1,1) + A(n,2);
            elseif i == n && j == m
                P(i,j) = A(n,m) + A(n-1,m) + A(n,m-1);
            elseif i == 1 % first row
                P(i,j) = sum(A(1,j-1:j+1),'all') + A(2,j
                   );
            elseif i == n % last row
                P(i,j) = sum(A(n,j-1:j+1),'all') + A(n
                   -1,j);
            elseif j == 1 % first column
                P(i,j) = sum(A(i-1:i+1,1)) + A(i,2);
            elseif j == m % last column
                P(i,j) = sum(A(i-1:i+1,m),'all') + A(i,m
                   -1);
            end
        else
            error("Wrong type. It must be 'S' or 'D'")
        end
    end
end
```

```
end

function flag = is_hv_convex(A)
% Find if a binary matrix is hv-convex.

[n,m] = size(A);

flag = 1;

for k = 1:n

    use = find(A(k,:));
    i = min(use);
    j = max(use);

    if ~isequal(unique(A(k,i:j)),1)

        flag = 0;
        return

    end

end

for k = 1:m

    use = find(A(:,k));
    i = min(use);
    j = max(use);

    if ~isequal(unique(A(i:j,k)),1)

        flag = 0;
        return

    end
```

```
end

function [T,M] = small_cases(n,m,type)
% This function finds matrices with the same projections
    (i.e. having switching components). It generates all
    the matrices of dimension n x m and computes their
   projections according to the window. 'D' is the 4-
   neighbourhood and 'S' the 8-neighbourhood.
The intern of the matrix is always full otherwise is not
possible to have switching components.
Return a tensor with the projections of all possible
   polyominoes.

%% Compute all the possible feet's lengths
ln = nchoosek(1:n,2);
ln = [ln;ln(:,2:-1:1);[(1:n)',(1:n)']];

lm = nchoosek(1:m,2);
lm = [lm;lm(:,2:-1:1);[(1:m)',(1:m)']];

l = [];

for q = 1:size(ln,1)

    use = repmat(ln(q,:),size(lm,1),1);

    l = [l;[use,lm]];

end

%% Create the matrix

A = zeros(n,m);

% Fill the intern
A(2:n-1,2:m-1) = 1;
```

```
% Try with all the possible feet

T = [];
M = [];
count = 1;

%% Generate all the possible matrices

for w = 1:size(l,1) % length of the feet

    for i = 1:(n-l(w,1)+1) % first column

        for j = 1:(m-l(w,3)+1) % first row

            for k = 1:(n-l(w,2)+1) % last column

                for t = 1:(m-l(w,4)+1) % last row

                    B = A;

                    B(i:i+l(w,1)-1,1) = 1; % first
                       column
                    B(1,j:j+l(w,3)-1) = 1; % first row
                    B(k:k+l(w,2)-1,m) = 1; % last column
                    B(n,t:t+l(w,4)-1) = 1; % last row

                    if is_hv_convex(B)

                        % Get projections

                        P = get_projections(B,type);

                        % Save the projections in a
                           tensor

                        M(:,:,count) = B;
                        T(:,:,count) = P;
                        count = count + 1;

                    end

                end

            end

        end

    end

end

% This script print all the $hv-$convex polyominoes of
   dimension n x m having the same projections according
    to the type of neighbours given by the user.

close all
clear all

% test small cases
n = input('Insert the number of rows: ');
m = input('Insert the number of columns: ');
type = input('Insert S for 4-Neighbourhood or D for 8-
   neighbourhood: ', 's');

while type ~= 'S' && type ~= 'D'

     type = input('Insert S for 4-Neighbourhood or D for
         8-neighbourhood: ', 's');

end

[T,A] = small_cases(n,m,type);
```

```
% Create a vector to save the equal matrix projections
use = zeros(1,size(T,3));
aux = 1; % determine equal projections

for i = 1:size(T,3)

    value = 0;

    for j = 1:size(T,3)

        if isequal(T(:,:,i),T(:,:,j))

            if i ~= j && ~isequal(A(:,:,i),A(:,:,j)) &&
               use(j) == 0

                use(i) = aux;
                use(j) = aux;
                value = 1;

            end

        end

    end

    if value

        aux = aux + 1;

    end

end

u = unique(use);

for r = 1:length(u)-1
```

```
    M = A(:,:,use == u(r+1));
    F = reshape(M,size(M,1)*size(M,2),[])';
    F = unique(F,'rows');
    M = reshape(F',[size(M,1),size(M,2),size(F,1)]);
    l = size(M,3);

    for s = 1:l

            subplot(1,l,s)
            imagesc(M(:,:,s))

    end

    pause

end
```