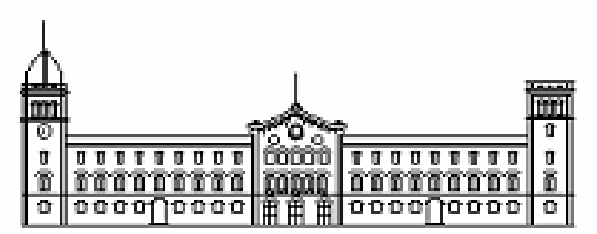


Treball final de grau

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques

Universitat de Barcelona

Cerca d'automorfismes i d'isomorfismes de grafs: algorismes i implementació

Juan José Bonet Ramis

Director: Antoni Benseny Ardiaca
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB
Barcelona, 23 de juny de 2013

Agraïments

M'agradaria agraïr al tutor del meu treball de final de grau de Matemàtiques, Antoni Benseny Ardiaca, tot el seu suport, la seva dedicació i paciència durant la realització d'aquest treball. Sense la seva col·laboració aquest projecte no hauria estat possible.

Índex

Índex de figures

Llista d'algorismes

Capítol 1

Introducció

1.1 Abstract

Graph Isomorphism problem (GIP) consists in constructing an efficient algorithm for testing whether two graphs are isomorphics and to find, if it exists, one isomorphism,i.e, to find a one-to-one, onto mapping from the set of vertexs of one graph to the set of vertexs of the other graph, so that the edges remain the same. GIP is one of the most interesting problems in Discrete Mathematics, it is essential for Graph Theory as well as for computational complexity theory. The problem has interest in the theoretical side, because it is not known if it is a P or NP-complete problem. On the one hand it is known that many classes of graphs admit polynomial time algorithms for isomorphism testing, such as rooted trees, planar graphs, circular graphs, graphs with bounded vertex degree, etc. On the other hand, the search for isomorphism-complete problems has produced a large list of problems, such as bipartite graph isomorphism, regular graph isomorphism, complement graph isomorphism, automorphism orbits search, automorphism generators search, etc. So, GIP is a good candidate for an intermediate computational status between P and NP problems. From the practical point of view, GIP has many applications for science and technology: pattern recognition, data mining, computer vision, chemistry and VLSI layout validation.

During the last decades many algorithms have been born to solve the problem, most of them based on Canonical labeling and direct backtracking, but Brendan McKay's *Nauty* algorithm has stood up among the others because its fine performing. In 2011, José Luis López Presa et al. tested *Nauty* with some special graphs, Miyazaki's graphs, and confirmed that it was very slow to find isomorphisms. So, they developed a new algorithm, *Conauto*, which attacks the problem in an original way, using special techniques to prune the automorphisms searching tree, to become one of the fastest algorithms today.

Tha aim of this work is, on the one hand, to study and analyze the theoretical background the algorithm *Conauto* relies on and, on the other hand, to try to perform any improvement in some aspects: it has been implemented a graphical interface to visualize the partitions on the graphs and it has been performed two different algorithms to calculate the whole automorphism group of the graph starting from a set of generators; a first one based on brute force (strength) and a second one based on the Schreier-Sims algorithm.

An algoritm that calculates all isomorphisms between two graphs, starting from one previous isomorphism and the whole automorphism group of one of the graphs, have also been implemented.

1.2 Motivació

Un dels problemes importants de la Matemàtica Discreta, i en particular de la Teoria de Grafs, és determinar si dos grafs són isomorfs o no i trobar un isomorfisme quan ho siguin. Això és, esbrinar si existeix una equivalència entre els vèrtexs i les arestes o arcs d'un graf i els de l'altre, que anomenem *isomorfisme*. Expressat d'una altra forma, determinar si es poden etiquetar els vèrtexs d'un graf usant les etiquetes dels vèrtexs de l'altre i obtenir una còpia exacta de l'altre graf. Una de les característiques més interessants del problema de trobar isomorfismes entre grafs és que, a hores d'ara, es desconeix si és tracta d'un problema de tipus P o NP-complet, això és, no se sap si es pot resoldre en temps polinomial o no.

El problema presenta aplicacions pràctiques molt interessants en diversos camps de la ciència i la tecnologia com el reconeixement de patrons, la mineria de dades, la distinció entre compostos químics i l'anàlisi de capes en circuits VLSI.
En els últims anys, el programa *Nauty* de Brendan McKay ha esdevingut un dels més eficients en la determinació d'isomorfismes entre grafs. El 2009 va aparéixer el programa *Conauto* de José Luis López Presa et al. que, amb successives modificacions posteriors, està competint en velocitat i eficiència amb la resta d'algorismes, tot i superant-los en alguns casos d'interès. Un dels punts claus del seu èxit és l'original atac que fa al problema, molt diferent de les aproximacions d'altres algorismes basades en l'etiquetatge canònic i el bactracking directe. El programa *Conauto* guanya eficiència en la cerca d'isomorfismes de grafs a partir de la cerca dels seus automorfismes.

La proposta de participació en la comprensió i extensió d'un programa d'aquestes característiques han estat claus en la motivació de la realització d'aquest treball.

1.3 Objectius i resum de resultats

En aquest treball es pretén realitzar una anàlisi comprensiva del programa *Conauto* i algunes extensions del mateix. L'objectiu principal del treball consisteix doncs a comprendre i fer una descripció acurada dels diversos algorismes que conformen el programa *Conauto* i la seva implementació i completar-lo en algun aspecte.

D'una banda, es volen estudiar els conceptes teòrics i els algorismes en els quals es basa el programa, presentant-los amb una formulació matemàtica i una descripció algorísmica consistent i aclaridora; d'altra banda, es pretén millorar una versió del *Conauto*, adaptada a *C++* pel director del treball, a fi d'incorporar-hi una part gràfica que permeti il·lustrar alguns aspectes d'interès.

Pel que fa a la comprensió dels conceptes i dels algorismes en les calsses *GIso*, en els capítols que segueixen se'n fa una nova descripció amb la finalitat d'aclarir alguns aspectes dels documents que acompanyen el programa, basant-se també en la nova implementació del programa, feta pel director. En el capítol 2, s'introdueixen els conceptes bàsics de grafs i es concreten els referits a automorfismes i isomorfismes i a la partició per graus. En el capítol 3, s'exposen els algorismes de generació de seqüències de particions d'un graf via refinament de particions per vèrtex i per conjunt. En el capítol 4, es detallen els algorismes emprats en la cerca d'automorfismes d'un graf, tot i argumentant les diferents tècniques emprades, basades en la compatibilitat de seqüències alternatives de particions. En el capítol 5, s'aborden finalment els algorismes de cerca d'isomorfismes entre dos grafs, emprant les particions per òrbites en els conjunts de vèrtexs donades pels automorfismes.

Pel que fa a la completació del programa, s'han implementat algorismes de visualització gràfica dels resultats, que han permès il·lustrar aquesta memòria. El programa ara pot visualitzar les cel·les de les particions i facilita l'observació gràfica de les òrbites dels vèrtexs i, en cas d'existir, de l'isomorfisme trobat entre dos grafs. Així, les adaptacions realitzades permeten reproduir gràficament algunes situacions que es produeixen durant la cerca d'automorfismes i de l'isomorfisme per poder-les analitzar millor i comprendre les accions realitzades.

El programa bàsic troba un conjunt de generadors del grup d'automorfismes del graf i fa un càlcul del nombre d'elements que té el grup sense generar-los. Una altra de les aportacions a considerar consisteix en la determinació de tots els elements del grup complet d'automorfismes mitjançant dos mètodes diferents. El primer i més senzill, utilitzant la força bruta; és a dir, calculant totes les possibles composicions entre els generadors disponibles del grup i emmagatzemant aquells elements que no hagin aparegut anteriorment. El segon mètode és més sofisticat i requereix considerablement menys temps de càlcul i es basa en la implementació de l'algorisme de Schreier-Sims. Una il·lustració dels resultats obtinguts consta al final del capítol 3.

Capítol 2

Conceptes generals

2.1 Conceptes bàsics de grafs

2.1.1 Definició. Un *graf dirigit* $G = (V, E)$ consisteix en un conjunt finit V de *vèrtexs* i un conjunt finit E d'*enllaços* tal que cada enllaç relaciona dos elements (vèrtexs) del conjunt V. L'enllaç $(u, v) \subseteq V \times V$ s'entén orientat d'u a v.

2.1.2 Definició. Donat un graf dirigit $G = (V, E)$, es considera la seva *matriu d'adjacència* M, donada per

$$M_{uv} = \begin{cases} 0 & si \quad (u,v) \quad \notin E \quad i \quad (v,u) \notin E \\ 1 & si \quad (u,v) \quad \notin E \quad i \quad (v,u) \in E \\ 2 & si \quad (u,v) \quad \in E \quad i \quad (v,u) \notin E \\ 3 & si \quad (u,v) \quad \in E \quad i \quad (v,u) \in E \end{cases}$$

2.1.3 Definició. Siguin $G = (V, E)$ un graf dirigit i M la seva matriu d'adjacència. Donat un vèrtex $u \in V$, el seu *grau* respecte al graf G, que notarem $Deg(u, V, G)$, ve donat per la terna (D_3, D_2, D_1) on $D_i = |\{v \in V \text{ tal que } \ M_{uv} = i\}|$, per a $i \in \{1, 2, 3\}$.

2.1.4 Definició. Siguin $G = (V, E)$ un graf dirigit i M la seva matriu d'adjacència. Donats un vèrtex $u \in V$ i un subconjunt $V_1 \subseteq V$, el *grau accessible* d'u respecte a V_1, $ADeg(u, V_1, G)$, ve donat per la terna (D_3, D_2, D_1) on $D_i = |\{v \in V_1 \text{ tal que } \ M_{uv} = i\}|$, per a $i \in \{1, 2, 3\}$.

2.1.5 Definició. Siguin $G = (V, E)$ un graf dirigit i M la seva matriu d'adjacència. Donats els subconjunts $V_1, V_2 \subseteq V$, V_1 té *grau accessible* d respecte a V_2, i es notarà per $ADeg(V_1, V_2, G) = d$, si $\forall u \in V_1$ se satisfà que $ADeg(u, V_2, G) = d$.

2.1.6 Definició. Es diu que la terna $(D_3, D_2, D_1) \prec (E_3, E_2, E_1)$ si la primera precedeix la segona en ordre lexicogàfic i que $(D_3, D_2, D_1) \succ (E_3, E_2, E_1)$ si és la segona la que precedeix la primera en ordre lexicogràfic.

2.2 Automorfismes, isomorfismes i invariants

2.2.1 Definició. Donats dos grafs $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ direm que són *isomorfs* si existeix una aplicació bijectiva $f : V_1 \to V_2$ tal que $(f(u), f(v)) \in E_2 \Leftrightarrow (u, v) \in E_1$. L'aplicació

f s'anomena *isomorfisme* entre els grafs. També s'escriu $G_1 \approx G_2$ per indicar que G_1 i G_2 són isomorfs.

2.2.2 Definició. Un isomorfisme f entre un graf G i ell mateix s'anomena *automorfisme* del graf. El conjunt $Aut(G)$ de tots els automorfismes d'un graf G és un subgrup del grup simètric de permutacions, anomenat *grup d'automorfismes* de G.

2.2.3 Definició. Sigui $\mathcal{G}$ una família de grafs. Un *invariant* en $\mathcal{G}$ és una funció ϕ amb domini $\mathcal{G}$ tal que

$$\phi(G_1) = \phi(G_2) \text{ si } G_1 \approx G_2.$$

2.2.4 Observació. Si $\phi(G_1) \neq \phi(G_2)$, es pot afirmar que G_1 i G_2 no són isomorfs; però, si $\phi(G_1) = \phi(G_2)$, no es pot afirmar res.

2.2.5 Definició. Sigui $\mathcal{G}$ una família de grafs. Un *invariant complet* en $\mathcal{G}$ és una funció ϕ amb domini $\mathcal{G}$ tal que

$$\phi(G_1) = \phi(G_2) \Leftrightarrow G_1 \approx G_2.$$

Un invariant complet també s'anomena *certificat*.

Alguns exemples d'invariants són el nombre de vèrtexs d'un graf, el nombre d'enllaços, el determinant de la matriu d'adjacència, el nombre de triangles en el graf, el nombre d'arbres d'expansió, etc. El qualsevol cas, cap d'aquests invariants és complet i no es coneix cap invariant complet computable en temps polinomial.

2.3 Partició per graus

2.3.1 Definició. Siguin $G = (V, E)$ un graf dirigit i M la seva matriu d'adjacència. Una *partició* del conjunt de vèrtexs V del graf G ve donada per la seqüència $\mathcal{S} = \{S_1, ..., S_n\}$ on $V = \cup_{i=1}^{n} S_i$, $S_i \cap S_j = \phi$ si $i \neq j$ i $S_i \neq \phi, \forall i, j \in \{1, ..., n\}$. Els conjunts S_i s'anomenen *cel·les* de la partició.

2.3.2 Definició. Siguin $G = (V, E)$ un graf i M la seva matriu d'adjacència. La *partició per graus* del graf G és una partició $\mathcal{S} = \{S_1, ..., S_n\}$ de V, que notarem $DegreePartition(G)$, tal que $\forall i, j \in \{1, ..., n\}$, $i < j \Rightarrow ADeg(S_i, V, G) \succ ADeg(S_j, V, G)$.

2.3.3 Observació. La partició de graus d'un graf també és un invariant.

Dos grafs isomorfs han de tenir la mateixa partició per graus. Per això, el primer pas del programa *Conauto* consisteix a calcular la partició per graus dels grafs.

En la figura 2.1 es mostra un graf de 8 vèrtexs i la seva partició per graus que conté dues cel·les.

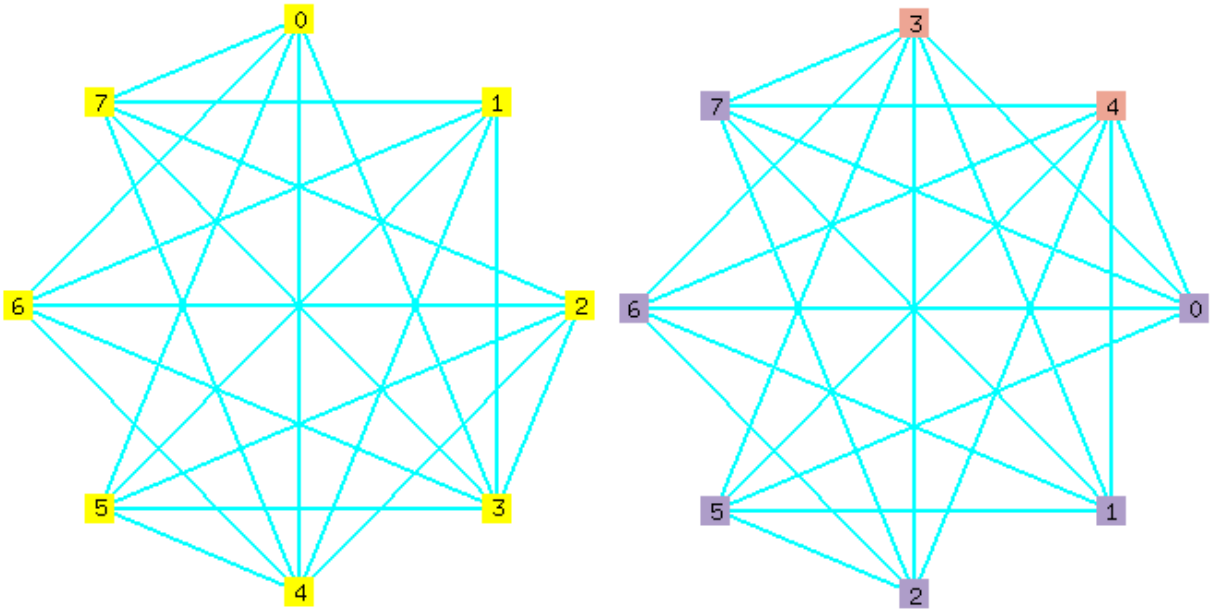


Figura 2.1: Graf regular de 8 vèrtex i la seva partició per graus.

Capítol 3

Refinament de particions

En el capítol anterior s'ha introduït el concepte de partició per graus d'un graf i s'ha apuntat que era un invariant. El pas següent en el programa *Conauto* és obtenir una seqüència de particions a partir d'anar trencant les cel·les de les particions anteriors fins arribar a obtenir una partició on totes les cel·les estiguin formades per un únic vèrtex (les anomenarem *unitàries*) o bé no tinguin adjacències. El pas d'una partició a la següent s'anomena *refinament* i pot ser per vèrtex o per conjunt.

3.1 Refinament per vèrtex

3.1.1 Definició. Siguin $G = (V, E)$ un graf i M la seva matriu d'adjacència. Siguin $v \in V$ un vèrtex i $W \subseteq V \backslash \{v\}$ un subconjunt de vèrtexs. La *partició per vèrtex* del conjunt W pel *vèrtex pivot* v, que es notarà per $partitionByVertex(W, v, G)$, és una partició $\mathcal{S} = \{S_1, ..., S_n\}$ de W tal que $\forall i, j \in \{1, ..., n\}$ si $i < j$, $ADeg(S_i, \{v\}, G) \succ ADeg(S_j, \{v\}, G)$.

3.1.2 Observació. La definició anterior s'aplicarà a conjunts W que seran cel·les d'una partició anterior.

Noti's que un vèrtex u d'una cel·la W pot tenir 4 tipus diferents d'ajacències amb un vèrtex v. Pot ser:

$$ADeg(u, v, G) = \begin{cases} 0 & si \quad (u,v) \notin E \quad i \quad (v,u) \notin E \\ 1 & si \quad (u,v) \notin E \quad i \quad (v,u) \in E \\ 2 & si \quad (u,v) \in E \quad i \quad (v,u) \notin E \\ 3 & si \quad (u,v) \in E \quad i \quad (v,u) \in E \end{cases}$$

Per tant, la partició per vèrtex d'una cel·la W pot trencar aquesta cel·la en fins a 4 cel·les noves, una per a cada un dels tipus d'adjacència que poden tenir els seus vèrtexs.

3.1.3 Definició. Siguin dues particions $\mathcal{S} = \{S_1, ..., S_n\}$ de V_1 i $\mathcal{T} = (T_1, ..., T_r)$ de V_2 amb $V_1 \cap V_2 = \emptyset$. La concatenació de les particions $\mathcal{S}$ i $\mathcal{T}$ve donada per $\mathcal{S} \circ \mathcal{T} = (S_1, ..., S_n, T_1, ..., T_r)$.

3.1.4 Definició. Siguin $G = (V, E)$ un graf i M la seva matriu d'adjacència. Donats una partició $\mathcal{S} = \{S_1, ..., S_n\}$ de V i un vèrtex de la partició $v \in S_i$ per a algun $i \in \{1, ..., n\}$, el *refinament de la partició* $\mathcal{S}$ pel *vèrtex pivot* v, que es notarà per $vertexRefinement(\mathcal{S}, v, G)$,

és una nova partició $\mathcal{T} = \mathcal{T}_1 \circ ... \circ \mathcal{T}_n$ tal que $\forall i \in \{1, ..., n\}$, $\mathcal{T}_i$ és una partició buida si la cel·la S_i no té adjacències en V i és igual a $partitionByVertex(S_i, v, G)$ en cas contrari.

3.1.5 Exemple. La primera figura mostra un graf regular de 8 vèrtexs amb una partició formada per una única cel·la que conté tots els vèrtexs (pintats del mateix color fucsia). La segona figura mostra el resultat d'aplicar un refinament per vèrtex a la partició utilitzant com a pivot el vèrtex 0: el vèrtex 0 té adjacències de tipus 3 amb els vèrtexs 1, 4 i 6 i no té cap altre vèrtex adjacent; per tant, trenca la cel·la en dues cel·les noves $\{1, 4, 6\}$ de color malva i $\{2, 3, 5, 7\}$ de color blau. El vèrtex 0 queda descartat com es veurà més endavant.

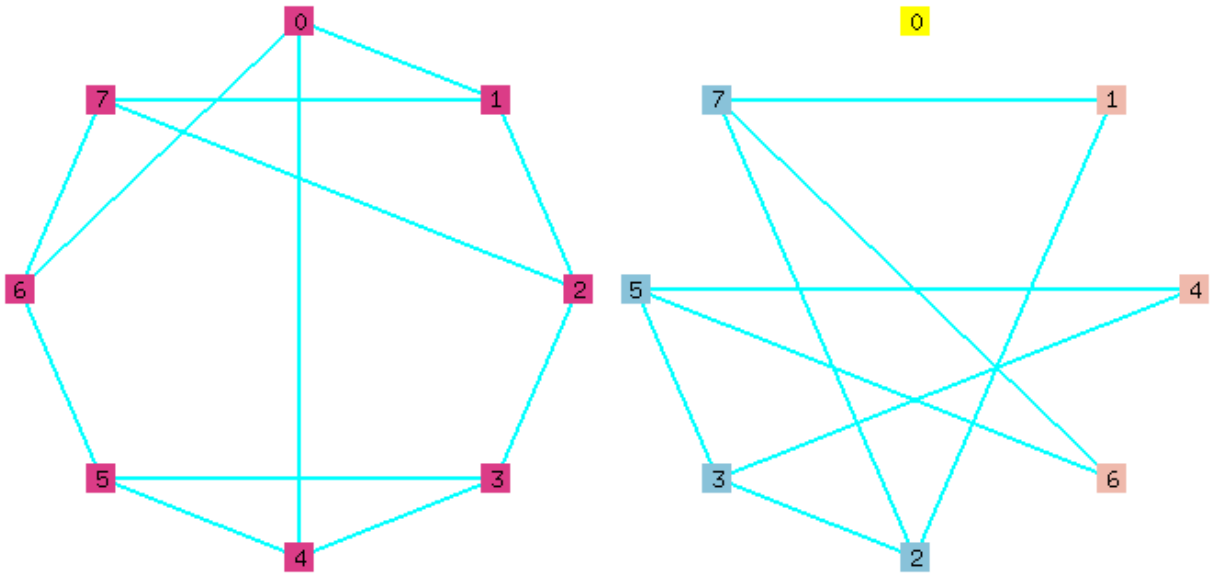


Figura 3.1: Graf regular de 8 vèrtexs amb una cel·la. Partició per vèrtex amb vèrtex 0 com a pivot.

Per realitzar el refinament per vèrtex s'utilitza l'algorisme 1.

algorisme 1 Refinament per vèrtex d'una partició.

```
Partition GIso::vertexRefinement(S^l)
 1: S^{l+1} ← newPartition(|V^l|)
 2: for all S_i^l ∈ S^l do
 3:    if ADeg(S_i^l, V, G) = 0 then
 4:       for all v ∈ S_i^l do
 5:          add v to discarded vertices
 6:       end for
 7:    else
 8:       for all adjacency type of S_i with pivot vertex do
 9:          S_j^{l+1} ← createNewCell(S_i^l)
10:          S_j^{l+1} ← putVerticesOnNewCell(S_i^l)
11:       end for
12:    end if
13: end for
14: return S^{l+1}
```

L'algorisme crea una nova partició refinada $\mathcal{S}^{l+1}$ i examina cada cel·la $S_j^l, j \in \{1, ..., r_l\}$, de la partició anterior $\mathcal{S}^l$. Si la cel·la no té adjacències afegeix els seus vèrtexs al vector de vèrtexs descartats. En cas contrari, examina les adjacències dels vèrtexs de la cel·la amb el vèrtex pivot, que poden ser de tipus 0, 1, 2 o 3, i crea una nova cel·la per a cadascuna de les adjacències anteriors. A continuació, afegeix els vèrtexs a cadascuna de les noves cel·les en funció de la seva adjacència amb el pivot. En el cas que no hi ha hagi cap vèrtex amb alguna de les adjacències, no es crearan les cel·les corresponents. L'algorisme retorna la nova partició refinada $\mathcal{S}^{l+1}$.

3.2 Refinament per conjunt

3.2.1 Definició. Siguin $G = (V, E)$ un graf i M la seva matriu d'adjacència. Siguin $V_1, V_2 \subseteq V$. La *partició per conjunt* de V_1 per V_2, que notarem $partitionBySet(V_1, V_2, G)$ és una partició $\mathcal{S} = \{S_1, ..., S_n\}$ de V_1 tal que $\forall i, j \in \{1, ..., n\}$ si $i < j$, $ADeg(S_i, V_2, G) \succ ADeg(S_j, V_2, G)$.

3.2.2 Definició. Siguin $G = (V, E)$ un graf i M la seva matriu d'adjacència. Donades una partició $\mathcal{S} = \{S_1, ..., S_n\}$ de V i $P = S_i$ per algun $i \in \{1, ..., n\}$ una cel·la de la partició, el *refinament de la partició* $\mathcal{S}$ *pel conjunt pivot* P, que es notarà per $setRefinement(\mathcal{S}, P, G)$, és una nova partició $\mathcal{T} = \mathcal{T}_1 \circ ... \circ \mathcal{T}_n$ tal que $\forall i \in \{1, ..., n\}$, $\mathcal{T}_i$ és una partició buida si la cel·la S_i no té adjacències en V i és igual a $partitionBySet(S_i, P, G)$ en cas contrari.

3.2.3 Exemple. La imatge de l'esquerra de la figura 3.2 mostra un graf amb 14 vèrtexs amb la partició formada per un vèrtex descartat $\{0\}$ i dues cel·les: la primera de color fucsia formada pels vèrtexs $\mathcal{S}_1 = \{2, 3, 4\}$ i la segona de color verd formada pels vèrtexs $S_2 = \{1, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$. La imatge de la dreta mostra el resultat d'aplicar un refinament per conjunt utilitzant com a conjunt pivot la primera cel·la: els graus accessibles dels vèrtexs de S_2 respecte a S_1 són: $ADeg(\{1\}, S_1, G) = (D_3, D_2, D_1) = (3, 0, 0)$, $ADeg(\{5\}, S_1, G) = (2, 0, 0)$, $ADeg(\{7\}, S_1, G) = (1, 0, 0)$ i $ADeg(\{u\}, S_1, G) = (3, 0, 0)$ si $u \in \{6, 8, 9, 10, 11, 12, 13\}$; per tant la cel·la S_2 es trenca en 4 cel·les noves $S'_1 = \{1\}$, $S'_2 = \{5\}$, $S'_3 = \{7\}$, $S'_4 = \{6, 8, 9, 10, 11, 12, 13\}$ que en la imatge es poden apreciar en colors diferents.

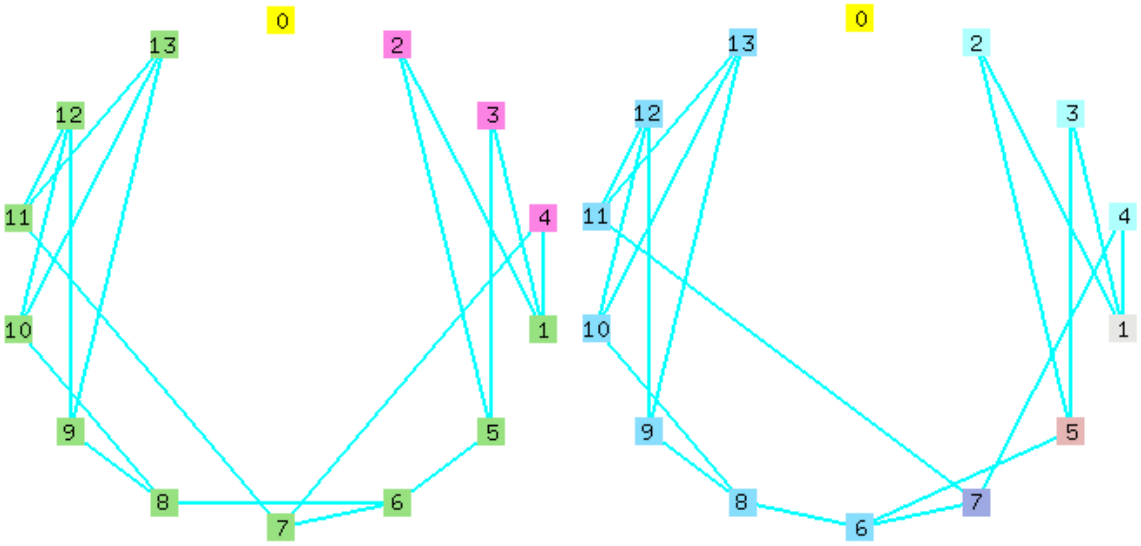


Figura 3.2: Partició per conjunt d'un graf de 14 vèrtexs.

L'algorisme 2, *setRefinement*, és l'encarregat de fer la partició per conjunt:

algorisme 2 Refinament per conjunt d'una partició.

bool GIso::setRefinement(Partition $\mathcal{S}^l$)

1: $\mathcal{S}^{l+1} \leftarrow newPartition(|V^l|)$
2: $success \leftarrow FALSE$
3: **for all** $S_i \in \mathcal{S}^l$ **do**
4: **if** $ADeg(S_i, V^l, G) = 0)$ **then**
5: **for all** $v \in S_i^l$ **do**
6: add v to discarded vertices
7: **end for**
8: **else**
9: **for all** $u \in S_i$ **do**
10: $AD[u] \leftarrow ADeg(u, S^l_{\mathcal{P}^l}, G)$
11: **end for**
12: $S'_i \leftarrow sortByADeg(S_i, AD)$
13: $j \leftarrow 0$
14: **for all** $u \in S'_i$ **do**
15: $v \leftarrow$ {precedent vertex to u in S'_i}
16: **if** (AD[u] != AD[v]) **then**
17: $success \leftarrow TRUE$
18: $j \leftarrow j + 1$
19: $T_j \leftarrow createNewCell(\mathcal{S}^{l+1})$
20: **end if**
21: $T_j \leftarrow T_j \cup \{u\}$
22: **end for**
23: **end if**
24: **end for**
25: **return** success

L'algorisme crea una nova partició refinada $\mathcal{S}^{l+1}$ i examina cada cel·la de la partició anterior $\mathcal{S}^l$. Si la cel·la no té adjacències, afegeix els seus vèrtexs al vector de vèrtexs descartats. En cas contrari, calcula el grau accessible de cada vèrtex de la cel·la amb la cel·la pivot $S^l_{\mathcal{P}^l}$ i ordena els vèrtexs de la cel·la segons aquest grau. A continuació, afegeix els vèrtexs a la nova partició creant una cel·la nova T_j per cada vèrtex que tingui un grau accessible diferent al del vèrtex anterior (els vèrtexs estan ordenats pel grau). Si s'ha creat més d'una cel·la nova a partir d'una vella, l'algorisme retorna TRUE per indicar que el refinament ha tingut èxit. En cas contrari, retorna FALSE.

Un cop explicats els dos tipus de refinament que s'utilitzen en l'algorisme *Conauto*, cal veure com es decideix quin tipus de refinament s'ha de fer servir en cada partició i quina és la millor cel·la pivot per portar-lo a terme. L'algorisme 3, *bestCell*, és l'encarregat de determinar, donada una partició, quina és la millor cel·la pivot.

algorisme 3 Determinació de la cel·la pivot òptima.

```
Cell GIso::bestCell( Partition S )
 1: bC ← S_1
 2: for all S_i ∈ S & i > 1 do
 3:    if (Valid(S_i)) then
 4:       if (!Valid(bC) || |bC| > |S − i| || (|bC| = |S_i| & ADeg(S_i, V, G) > ADeg(bC, V, G)))
          then
 5:          bC ← S_i
 6:       end if
 7:    end if
 8: end for
 9: if (!Valid(bC) & |bC| > 1) then
10:    return bestIndividualizedCell( S )
11: end if
12: return bC
```

L'algorisme tracta d'escollir la cel·la vàlida amb menor nombre de vèrtexs que tingui el major nombre d'adjacències (major grau accessible). En el cas que no hi hagi cap cel·la vàlida, es crida l'algorisme 4, *bestIndividualizedCell*.

3.2.4 Definició. Siguin $G = (V, E)$ un graf i $\mathcal{S} = \{S_1, ..., S_n\}$ una partició de V. $\mathcal{S}$ és *equitativa* si $\forall i, j \in \{1, ..., n\}$ i $\forall u, v \in S_i$ es té que $ADeg(u, S_j, G) = ADeg(v, S_j, G)$.

3.2.5 Observació. En una partició equitativa no es pot decidir quina cel·la pivot s'hauria d'utilitzar, perquè hi ha simetria. Aquest és un possible nivell on sorgiran automorfismes i, per tant, el refinament s'etiqueta com a *UNKNOWN* i serà un possible nivell de *BACKTRACKING*.

3.2.6 Definició. Siguin $G = (V, E)$ un graf, $\mathcal{S} = \{S_1, ..., S_n\}$ una partició equitativa d'un conjunt $V_1 \subseteq V$ i $\mathcal{T} = \{T_1, ..., T_m\}$ una partició equitativa de $V_2 \subseteq V_1$. $\mathcal{T}$ és una *subpartició* de $\mathcal{S}$ si, i només si, $\forall i \in \{1, ..., n\}, \nexists j, k \in \{1, ..., m\}$ tals que $T_j, T_k \subseteq S_i$, $ADeg(T_j, V_2, G) > 0$ i $ADeg(T_k, V_2, G) > 0$. És a dir, cada cel·la de $\mathcal{T}$ amb adjacències està inclosa en una cel·la diferent de $\mathcal{S}$.

algorisme 4 Determinació de la cel·la pivot òptima si totes són no vàlides.

Cell GIso::bestIndicidualizedCell(Partition $\mathcal{S}$)

1: $bC \leftarrow S_1$
2: $C \leftarrow compute\_valid\_cells(\mathcal{S})$;
3: $n \leftarrow 0$
4: $bC \leftarrow S_1 \in C$
5: **for all** $S_i \in C$ **do**
6: $\mathcal{T} \leftarrow$ multirefinement($\mathcal{S}$);
7: – Sigui $\mathcal{T} = \{T_1, ..., T_{r'}\}, W = \cup_{i=1}^{r'} T_i$
8: **if** ($|W| = |\mathcal{T}|$) **then**
9: **return** bC
10: **end if**
11: **if** ($\mathcal{T}$ is subpartition of $|\mathcal{S}|$) **then**
12: **return** bC
13: **end if**
14: **if** (r' + $|V|$ - $|W|$ < n) **then**
15: $bC \leftarrow S_i$
16: $n \leftarrow r' + |V| - |W|$
17: **else if** ((r' + $|V|$ - $|W|$ = n) && ($|S_i| < |bC|$)) **then**
18: $bC \leftarrow S_i$
19: **end if**
20: **end for**
21: **return** bC

L'algorisme 4 mostra la funció *bestIndividualizedCell* que tracta de trobar la millor cel·la pivot per a realitzar la partició següent. Primer crida la funció *compute_valid_cells* que s'encarrega de generar el conjunt de cel·les vàlides com a candidates a ser la cel·la pivot; aquest conjunt és el format per totes les cel·les que tenen diferent nombre de vèrtexs o diferent grau accessible. És a dir, s'agafa una única cel·la d'entre totes les que que coincideixen en nombre de vèrtexs i grau accessible. Així el conjunt serà: $C = \{S_i \in \mathcal{S} : \nexists j < i, |S_i| = |S_j| \text{ i } ADeg(S_j, V, G) = ADeg(S_i, V, G)\}$. A continuació es tracten cadascuna d'aquestes cel·les com a pivot i es prova de generar una seqüència de particions fins arribar a la partició equitativa següent: aquesta tasca la realitza la funció *multirefinement*. A partir de la seqüència obtinguda, poden donar-se diverses situacions:

1. S'ha arribat a l'última partició i es retorna la cel·la que estem analitzant.
2. S'ha arribat a una partició que és subpartició de l'original, es retorna la cel·la tractada.
3. En cas contrari, es proven la resta de cel·les i es retorna aquella que minimitzi la suma de cel·les i vèrtexs descartats en la partició equitativa final trobada. Aquest criteri garanteix que es trenquin al màxim totes les cel·les.

L'algorisme 5 realitza el multirefinament de la partició equitativa $\mathcal{S}$. Primer fa un refinament per vèrtex i obté una nova partició $\mathcal{T}$ i cerca la millor cel·la pivot per a aquesta nova partició. A continuació, entra en un bucle: mentre la cel·la pivot sigui vàlida i no s'hagi arribat a l'última

partició, va fent refinaments de les particions que obté, ja sigui per vèrtex, quan la millor cel·la té un únic vèrtex, o per conjunt. En el cas de que el refinament sigui per conjunt mira si ha tingut èxit i, en cas contrari, marca la cel·la com no vàlida i torna a realitzar el refinament de la mateixa partició amb una altra cel·la pivot. El procediment acaba quan totes les cel·les són no vàlides (es troba en una partició equitativa) o s'ha arribat a l'última partició.

algorisme 5 Multirefinament per trobar la partició equitativa següent a partir de la partició equitativa $\mathcal{S}$.

Partition GIso::multirefinement(Partition $\mathcal{S}$)

1: $\mathcal{T} \leftarrow vertexRefinement(\mathcal{S})$;
2: - - Siguin $\mathcal{T} = \{T_1, ..., T_{r_i}\}, W = \cup_{i=0}^{r_i} T_i$ la partició refinada i el conjunt de vèrtexs resultant.
3: $bC \leftarrow bestCell(\mathcal{T})$;
4: **while** $(valid(bC) \quad \&\& \quad |\mathcal{T}| < |W|)$ **do**
5: **if** ($|bC| = 1$) **then**
6: $\mathcal{T}' \leftarrow vertexRefinement(\mathcal{T})$;
7: **else**
8: $success \leftarrow setRefinement(\mathcal{T})$;
9: - - Siguin $\mathcal{T}' = \{T'_1, ..., T'_{r_i}\}, W' = \cup_{i=0}^{r_i} T'_i$ la partició refinada a partir de $\mathcal{T}_i$ i el conjunt de vèrtexs resultant, respectivament.
10: **if** (!success) **then**
11: DeletePartiton($\mathcal{T}'$);
12: $bC \leftarrow bestCell(\mathcal{T})$;
13: continue;
14: **end if**
15: **end if**
16: $\mathcal{T} \leftarrow \mathcal{T}'$;
17: $bC \leftarrow bestCell(\mathcal{T})$;
18: **end while**
19: **return** $\mathcal{T}$

3.3 Seqüència de particions

3.3.1 Definició. Sigui $G = (V, E)$ un graf. La seva *seqüència de particions* és una terna $Q = \{\mathtt{S}, \mathtt{R}, \mathtt{P}\}$, on $\mathtt{S} = \{\mathcal{S}^1, ..., \mathcal{S}^t\}$ és el conjunt de particions obtingudes en els diferents refinaments, $\mathtt{R} = \{\mathcal{R}^1, ..., \mathcal{R}^{t-1}\}$ és el conjunt dels diferents tipus de refinaments aplicats a cada partició per obtenir la partició refinada següent i $\mathtt{P} = \{\mathcal{P}^1, ..., \mathcal{P}^{t-1}\}$ indica la cel·la pivot utilitzada en el refinament de cadascuna de les particions. Així es té, en cada *nivell i* de la seqüència, alguna de les situacions següents:

1. $\mathcal{R}^i = VERTEX$ vol dir que es refina la partició i per vèrtex i $\mathcal{P}^i$ indica la cel·la unitària (amb un únic vèrtex) utilitzada.
2. $\mathcal{R}^i = SET$ vol dir que es refina la partició i per conjunt i $\mathcal{P}^i$ indica la cel·la utilitzada (en aquest cas té més d'un vèrtex).
3. $\mathcal{R}^i = UNKNOWN$ significa que s'ha arribat a una partició equitativa que no es pot refinar igual que en les casos anteriors. En aquest cas, es fa un refinament per vèrtex,

però s'etiqueta el refinament com a *UNKNOWN* perquè més endavant es decidirà si es canvia el tipus de refinament a $VERTEX$ o $BACKTRACK$.

4. $\mathcal{R}^i = BACKTRACK$ implica que, en un nivell que estava etiquetat com a *UNKNOWN*, s'han provat refinaments amb els diferents vèrtexs de la partició, suposadament equivalents, obtenint alguns refinaments que no eren compatibles entre si i, per tant, arribant a la conclusió que no tots els vèrtexs de la partició són equivalents. Els nivells de tipus $BACKTRACK$ són més complicats d'estudiar, però també més rics en la cerca d'automorfismes del graf.

3.3.2 Definició. Siguin $G = (V_G, E_G)$ i $H = (V_H, E_H)$ dos grafs amb seqüències de particions $Q_G = \{\mathtt{S}_G, \mathtt{R}_G, \mathtt{P}_G\}$ i $Q_H = \{\mathtt{S}_H, \mathtt{R}_H, \mathtt{P}_H\}$ respectivament. Direm que Q_G i Q_H són seqüències *compatibles* si satisfan:

1. $|\mathtt{S}_G| = |\mathtt{S}_H| = t$
2. Siguin $\mathtt{S}_G = (\mathcal{S}^1, ..., \mathcal{S}^t)$ i $\mathtt{S}_H = (\mathcal{T}^1, ..., \mathcal{T}^t)$ aleshores $\forall i \in \{1, ..., t\}, |\mathcal{S}^i| = |\mathcal{T}^i| = r_i$.
3. Siguin $\mathtt{R}_G = \{\mathcal{R}_G^1, ..., \mathcal{R}_G^{t-1}\}$ i $\mathtt{R}_H = \{\mathcal{R}_H^1, ..., \mathcal{R}_H^{t-1}\}$, aleshores $\forall i \in \{1, ...t-1\}, R_G^i = R_H^i$.
4. Siguin $\mathtt{P}_G = \{\mathcal{P}_G^1, ..., \mathcal{P}_G^{t-1}\}$ i $\mathtt{P}_H = \{\mathcal{P}_H^1, ..., \mathcal{P}_H^{t-1}\}$, aleshores $\forall i \in \{1, ...t-1\}, \mathcal{P}_G^i = \mathcal{P}_H^i$.
5. $\forall i \in \{0, ..., t\}$, les particions $\mathcal{S}^i$ i $\mathcal{T}^i$ són *compatibles*. És a dir, si $\mathcal{S}^i = \{S_0^i, ..., S_{r_i}^i\}$ i $\mathcal{T}^i = \{T_0^i, ..., T_{s_i}^i\}$ se satisfà que $|\mathcal{S}^i| = |\mathcal{T}^i|$ $(s_i = r_i)$ i $\forall j \in \{1, ..., r_i\}, |S_j^i| = |T_j^i|$ i $ADeg(S_j^i, V_G, G) = ADeg(T_j^i, V_H, H)$.
6. $\forall i \in \{1, ..., t\}$ si $\mathcal{S}^i$ és equitativa, aleshores $\forall j, k \in \{1, ..., r_i\}$, $ADeg(S_j^i, S_k^i, G) = ADeg(T_j^i, T_k^i, H)$.
7. $\forall i \in \{1, ..., t\}$ si $\mathcal{S}^i$ no és equitativa, aleshores $\forall j \in \{1, ..., r_i\}$, $ADeg(S_j^i, V_G^i, G) = ADeg(T_j^i, V_H^i, H)$ on $V_G^i = \cup_{j=1}^{r_i} S_j^i$ i $V_H^i = \cup_{j=1}^{r_i} T_j^i$.

3.3.3 Definició. Sigui $Q = \{\mathtt{S}, \mathtt{R}, \mathtt{P}\}$ una seqüència de particions d'un graf G tal que $\mathcal{S}^i = \{S_1^i, ..., S_{r_i}^i\}, \forall i \in \{1, ..., t\}$ i $V^i = \cup_{j=1}^{r_i} S_j^i$. Direm que Q és una seqüència de particions *completa* si $\forall j \in \{1, ..., r_t\}$, $|S_j^t| = 1$ o bé $ADeg(S_j^t, V^t, G) = 0$. És a dir, en l'última partició totes les cel·les tenen un únic vèrtex o bé no tenen adjacències amb la resta de vèrtexs de totes les cel·les.

L'algorisme 6, *sequence*, és l'encarregat de la generació de la seqüència de particions d'un graf G.

algorisme 6 Generació de la seqüència de particions.

(Partition sequence) GIso::sequence($G, \mathcal{D}$)

1: $\mathcal{S}^0 \leftarrow \mathcal{D}$
2: $i \leftarrow 0$
3: **while** ($|\mathcal{S}^i| \neq |V^i|$) **do**
4: $bC \leftarrow bestCell(\mathcal{S}^i)$
5: **if** ($ADeg(bC, V^i, G) = 0$) **then**
6: **break**
7: **end if**
8: **if** ($|bC| = 1$) **then**
9: $\mathcal{S}^{i+1} \leftarrow vertexRefinement(\mathcal{S}^i)$
10: $\mathcal{R}^i \leftarrow VERTEX$
11: **else if** ($valid(S_i)$) **then**
12: $valid(S_i) \leftarrow FALSE$
13: $success \leftarrow setRefinement(\mathcal{S}^i)$
14: **if** (success) **then**
15: $\mathcal{R}^i \leftarrow SET$
16: **else**
17: $DeletePartition(\mathcal{S}^{i+1})$
18: continue
19: **end if**
20: **else**
21: $\mathcal{S}^{i+1} \leftarrow vertexRefinement(\mathcal{S}^i)$
22: $\mathcal{R}^i \leftarrow UNKNOWN$;
23: **end if**
24: $i \leftarrow i + 1$
25: **end while**
26: $t \leftarrow i$
27: $discardedUnlinkedCells(\mathcal{S}^t)$
28: $DV \leftarrow addVerticesOf(\mathcal{S}^t)$
29: $\{B^0, ..., B^t\} \leftarrow backtrackPartitions()$
30: $\{L^0, ..., L^t\} \leftarrow limitPartitions()$
31: **return** $\{\mathtt{S}, \mathtt{R}, \mathtt{P}, \{B^0, ..., B^t\}, \{L^0, ..., L^t\}\}$

L'algorisme comença inicialitzant la primera partició $\mathcal{S}^0$ a la partició per graus $\mathcal{D}$. A continuació, mentre no totes les cel·les siguin unitàries, procedeix a refinar la partició actual de la forma següent: primer determina la millor cel·la per portar a terme el refinament i si aquesta no té adjacències acaba l'algorisme. Si la millor cel·la té adjacències, mira la mida de la cel·la. Si té un únic vèrtex, procedeix a fer un refinament per vèrtex i etiqueta el refinament com a *VERTEX*. Si la cel·la té més d'un vèrtex, mira si és vàlida i, si ho és, la marca com a invàlida per no tornar-la a fer servir i s'intenta fer un refinament per conjunt. En cas que el refinament per conjunt hagi trencat alguna cel·la, s'etiqueta el tipus de refinament com a *SET*. En cas contrari, s'esborra la partició generada i torna a començar la iteració cercant una nova cel·la pivot. Quan la cel·la pivot escollida no és vàlida es fa un refinament per vèrtex etiquetant el tipus de refinament com a *UNKNOWN*.

Finalment, quan s'ha acabat de computar la seqüència la funció $discardedUnlinkedCells(\mathcal{S}^t)$, afegeix les cel·les descartades de l'última partició a un vector de cel·les descartades i també s'afegeixen els vèrtexs de l'última partició al vector de vèrtexs descartats DV. Per acabar, es crida les funcions *backtrackPartitions()* i *limitPartitions()* que s'explicaran més endavant.

3.3.4 Exemple. A continuació es mostra tot el procès per generar la seqüència de particions del graf de la figura 3.3:

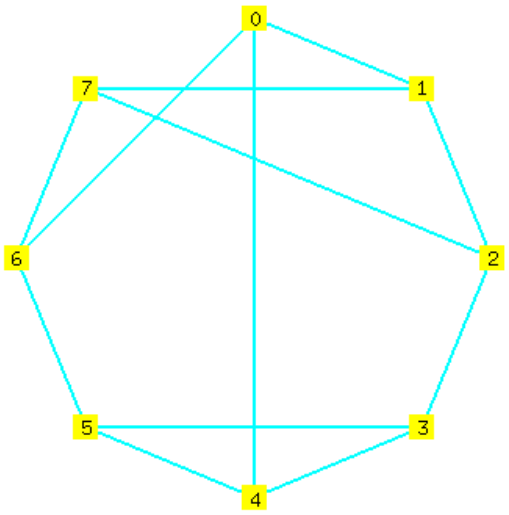


Figura 3.3: Graf LPTA de 8 vèrtexs.

Les imatges de la taula 3.1 es llegeixen d'esquerra a dreta i de dalt a baix.

La primera imatge mostra la partició per graus del graf LPTA: tots els vèrtexs tenen $ADeg(u, V, G) = (3, 0, 0)$. Per tant, només hi ha una cel·la, que és doncs la cel·la pivot i es marca com a no vàlida perquè només n'hi ha una. Així, s'etiqueta el refinament com a $\mathcal{R}^0 = UNKNOWN$. Es procedeix a fer el refinament per vèrtex utilitzant el vèrtex pivot 0: si $u \in \{1, 4, 6\}, ADeg(u, \{0\}, G) = (1, 0, 0)$ i, per a la resta de vèrtexs u, el grau accesible es $ADeg(u, \{0\}, G) = (0, 0, 0)$. Per tant la cel·la es trenca en dues cel·les que es mostren en la segona imatge.

Per a la partició 1, el vèrtex 0 ha estat descartat (totes les seves adjacències s'han utilitzat) i conté dues cel·les: $S_1 = \{1, 4, 6\}$ i $S_2 = \{2, 3, 5, 7\}$ que són vàlides per provenir d'una cel·la trencada. A continuació, es busca la millor cel·la pivot que resulta ser la S_1 per ser la més petita. Es fa un refinament per conjunt amb èxit, ja que, si $u \in \{2, 3\} \subset S_2, ADeg(u, S_1, G) = (1, 0, 0)$ i, si $u \in \{5, 7\}, \subset S_2, ADeg(u, S_1, G) = (2, 0, 0)$. Per tant, la cel·la S_2 es trenca en dues. Com hi ha hagut èxit, s'etiqueta el refinament com a $\mathcal{R}^1 = SET$ i la cel·la S_1 es marca com a no vàlida.

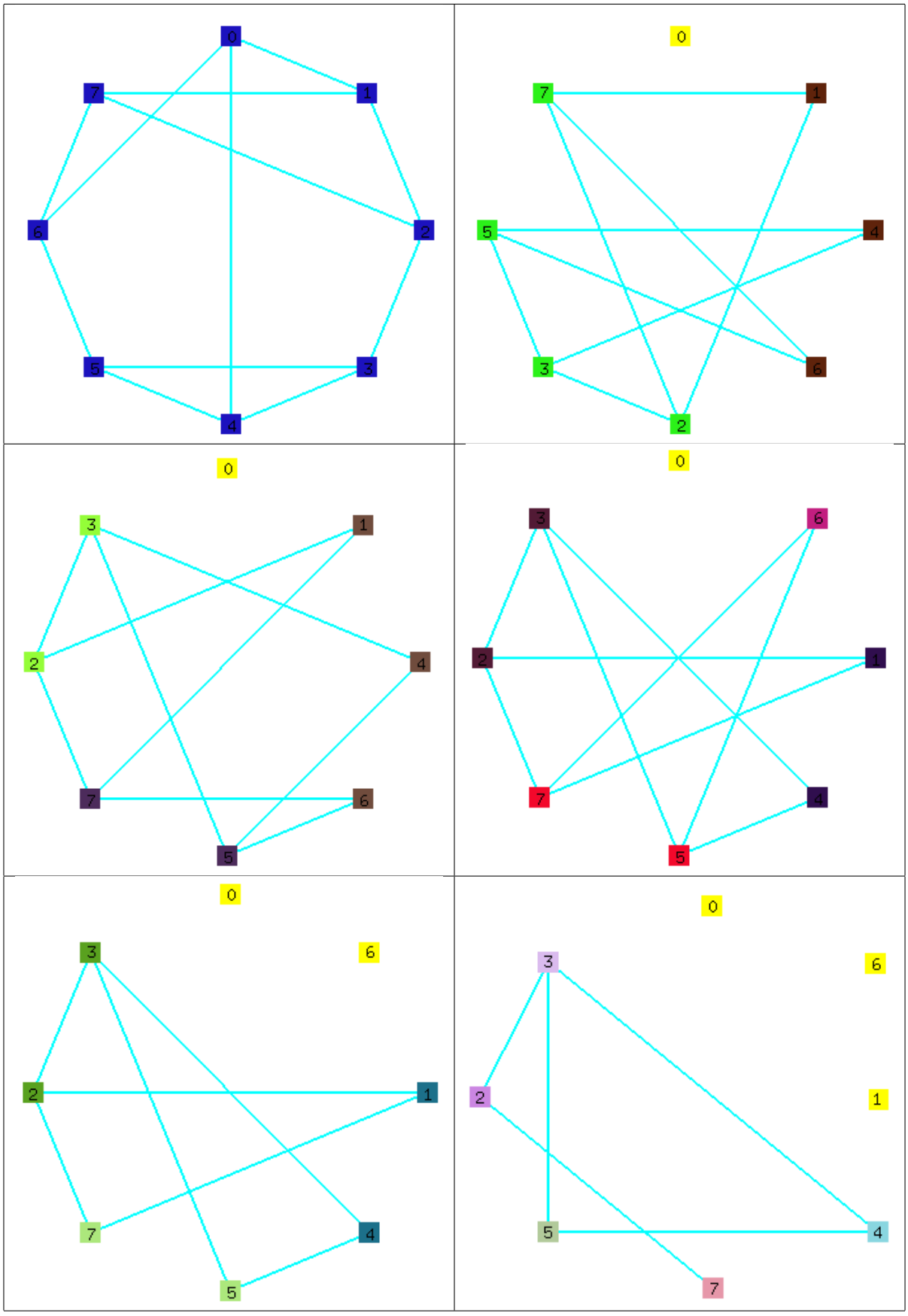
La partició 2 conté tres cel·les: $S_1 = \{1, 4, 6\}, S_2 = \{5, 7\}, S_3 = \{2, 3\}$ de les quals la cel·la S_2 és l'escollida per al refinament següent. Ara es veu que, si $u \in \{1, 4\} \subset S_1, ADeg\{u, S_2, G\} = (1, 0, 0)$ i $ADeg\{\{6\}, S_2, G\} = (2, 0, 0)$. Per tant, es trenca la cel·la S_1. La cel·la S_3 no es pot trencar. Com que hi ha hagut èxit, es marca el refinament com a $\mathcal{R}^2 = SET$. La cel·la S_2 queda etiquetada com a no vàlida.

La partició 3 està formada per 4 cel·les: $S_1 = \{6\}, S_2 = \{1, 4\}, S_3 = \{5, 7\}$ i $S_4 = \{2, 3\}$ de les quals la millor cel·la pivot és S_1 per tenir només un vèrtex. Fent un refinament per vèrtex amb

el vèrtex pivot 6, s'etiqueta $\mathcal{R}^3 = \mathit{VERTEX}$, no s'aconsegueix trencar cap cel·la perqué $\forall u \in S_2, ADeg\{u, 6, G\} = (0, 0, 0)$, $\forall u \in S_3, ADeg\{u, 6, G\} = (1, 0, 0)$ i $\forall u \in S_4, ADeg\{u, 6, G\} = (0, 0, 0)$, però es descarta el vèrtex $\{6\}$. Per tant, els vèrtexs descartats ara són $\{0, 6\}$ i resten els vèrtexs $V^4 = \{1, 2, 3, 4, 5, 7\}$.

La partició 4 té 3 cel·les: $S_1 = \{1, 4\}, S_2 = \{5, 7\}, S_3 = \{2, 3\}$ de les quals la S_2 és no vàlida. La millor cel·la pivot és la S_3 ja que té major grau accessible que la S_1 i es marca com a no vàlida. Si es fa un refinament per conjunt amb aquesta cel·la pivot, no hi ha èxit perquè $ADeg\{\{1\}, S_3, G\} = ADeg\{\{4\}, S_3, G\} = (1, 0, 0)$ i $ADeg\{\{5\}, S_3, G\} = ADeg\{\{7\}, S_3, G\} = (1, 0, 0)$ i, per tant, no es pot trencar cap cel·la. A continuació, es busca una altra cel·la pivot i es troba l'única que queda, la S_1. Es marca com a no vàlida i es fa un refinament per conjunt sense èxit, ja que, $ADeg\{\{2\}, S_1, G\} = ADeg\{\{3\}, S_1, G\} = (1, 0, 0)$ i $ADeg\{\{5\}, S_1, G\} = ADeg\{\{7\}, S_1, G\} = (1, 0, 0)$ i no es trenca cap cel·la. Es torna a iniciar el procès i se cerca la millor cel·la (ara totes són no vàlides) i es troba la cel·la S_1. Es fa un refinament per vèrtex usant el vèrtex $\{1\}$ i s'etiqueta el refinament com a $\mathcal{R}^4 = \mathit{UNKNOWN}$. Amb aquest refinament, s'aconsegueix trencar les dues cel·les que queden S_2 i S_3 degut a què $ADeg\{\{2\}, \{1\}, G\} = (1, 0, 0), ADeg\{\{3\}, \{1\}, G\} = (0, 0, 0)$ i $ADeg\{\{5\}, \{1\}, G\} = (0, 0, 0), ADeg\{\{7\}, \{1\}, G\} = (1, 0, 0)$.

S'obté finalment la partició 5 (última imatge de la taula) on totes les cel·les són unitàries i, per tant, finalitza el refinament. Per acabar, s'afegeixen els vèrtexs de l'última partició al conjunt de vèrtexs descartats que queda així: $\{0, 6, 1, 4, 5, 7, 2, 3\}$. Aquesta seqüència de particions defineix un ordre en el conjunt de vèrtexs V del graf.



Taula 3.1: Seqüència de particions

Capítol 4

Determinació d'automorfismes

4.1 Marc teòric

4.1.1 Definició. Sigui $Q = \{\mathtt{S}, \mathtt{R}, \mathtt{P}\}$ la seqüència de particions del graf G=(V,E) on $\mathtt{S} = \{\mathcal{S}^0, ..., \mathcal{S}^t\}$, $\mathtt{R} = \{\mathcal{R}^0, ..., \mathcal{R}^{t-1}\}$, $\mathtt{P} = \{\mathcal{P}^0, ..., \mathcal{P}^{t-1}\}$, per a tot $i \in \{0, ..., t\}, \mathcal{S}^i = \{S^i_1, ..., S^i_{r_i}\}$ i $V = \cup_{j=0}^{r_i} S^i_j$. L'*ordre induït* per Q en el conjunt de vèrtexs $(V \backslash V^t) \cup \{v \in V^t | ADeg(v, V^t, G) > 0\}$, que notarem $<_Q$, és tal que satisfà:

1. Per a tot $i \in \{0, ..., t-1\}$, per a tot $u \in V^i \backslash V^{i+1}, v \in V^{i+1}, u <_Q v$.
2. Per a tot $i \in \{0, ..., t-1\}$, $S^{i+1} = vertexRefinement(S^i, u, G) \Rightarrow \forall v \in V^i \backslash V^{i+1}, u <_Q v$.
3. Per a tot $i \in \{0, ..., t\}$:
   (a) Per a tot $x, y \in \{1, ..., r_i\}$ tals que $ADeg(S^i_x, V^i, G) = ADeg(S^i_y, V^i, G) = 0, x < y \Rightarrow \forall u \in S^i_x, v \in S^i_y, u <_Q v$.
   (b) Per a tot $x \in \{1, ..., r_i\}$ tal que $ADeg(S^i_x, V^i, G) = 0$ i $|S^i_x| > 1$, per a tot $u, v \in S^i_x, u <_Q v$ si, i només si, u precedeix v en ordre lexicogràfic.

Si la seqüència de particions és completa, aleshores indueix un ordre total en els vèrtexs de tot el conjunt V afegint la condició següent: $\forall u, v \in V^t$ tals que $ADeg(u, V^t, G) > 0$ i $ADeg(v, V^t, G) > 0$, existeixen $x, y \in \{1, ..., r_t\}$ tals que $\{u\} = S^t_x$ i $\{v\} = S^t_y$. Aleshores, $x < y \Rightarrow u <_Q v$.

4.1.2 Teorema. *Siguin $G = (V_G, E_G)$ i $H = (V_H, E_H)$ dos grafs. Els grafs G i H són isomorfs si, i només si, existeixen dues seqüències de particions Q_G i Q_H dels grafs G i H, respectivament, compatibles entre si.*

Demostració: Es suposa primer que els grafs G i H són isomorfs. Sigui $Q_G = (\mathtt{S}_G, \mathtt{R}_G, \mathtt{P}_G)$ una seqüència de particions del graf G amb $\mathtt{S}_G = (\mathcal{S}^0, ..., \mathcal{S}^t)$, $\mathtt{R}_G = (\mathcal{R}^0, ..., \mathcal{R}^{t-1})$, $\mathtt{P}_G = (\mathcal{P}^0, ..., \mathcal{P}^{t-1})$. Es nota $\mathcal{S}^i = (S^i_1, ..., S^i_{r_i})$ i $V^i = \cup_{j=1}^{r_i} S^i_j$. Sigui $m : V_G \to V_H$ l'isomorfisme entre els dos grafs. Es veurà que es pot construir una seqüència de particions pel graf H, Q_H, compatible amb Q_G, $\mathtt{S}_H = (\mathcal{T}^0, ..., \mathcal{T}^t)$, $\mathtt{R}_H = (\mathcal{R}^0, ..., \mathcal{R}^{t-1})$, $\mathtt{P}_H = (\mathcal{P}^0, ..., \mathcal{P}^{t-1})$ (s'usarà el mateix tipus de refinament que en el graf G amb el mateix conjunt pivot). Sigui $\mathcal{T}^0$ la partició per graus del graf H. Aquesta partició és compatible amb la partició per graus del graf G,

$\mathcal{S}^0$, perquè els grafs són isomorfs i han de tenir el mateix nombre de vèrtexs de cada grau, i l'isomorfisme m aplica vèrtexs d'una cel·la S_i^0 en vèrtexs de la cel·la corresponent, T_i^0, de la partició per graus del graf H. Es suposa ara que fins a un nivell l les seqüències Q_G i Q_H són compatibles. Sota aquestes condicions s'ha de comprovar que les particons $\mathcal{S}^{l+1}$ i $\mathcal{T}^{l+1}$ són compatibles. Si una cel·la S_x^l va ser descartada en el refinament des del nivell l fins al nivell $l+1$ és perquè els vèrtexs d'aquesta cel·la no tenen enllaços en aquest nivell, per tant, per la compatibilitat de les particions $\mathcal{S}^l$ i $\mathcal{T}^l$, la cel·la corresponent, T_x^l, tampoc té enllaços. Depenent del tipus de refinament realitzat tenim tres casos:

1. Si $\mathcal{R}^l = VERTEX$, es pot refinar la partició $\mathcal{T}^l$ per vèrtex usant la cel·la pivot $T_{P^l}^l$, que només té un vèrtex, i que és la imatge per m de la cel·la $S_{P^l}^l$. Per la compatibilitat de les particions $\mathcal{S}^l$ i $\mathcal{T}^l$ (hipòtesi d'inducció), els enllaços del vèrtex $S_{P^l}^l$ amb la resta de cel·les són iguals als enllaços del vèrtex $T_{P^l}^l$ amb els de les respectives cel·les de la partició $\mathcal{T}^l$ i, per tant, les noves cel·les generades en el refinament tindran el mateix nombre de vèrtexs i el mateix tipus d'adjacències que les corresponents cel·les de la partició $\mathcal{S}^l$. Així, les particions $\mathcal{S}^{l+1}$ i $\mathcal{T}^{l+1}$ seran compatibles i l'isomorfisme m aplicarà els vèrtexs de cel·les de $\mathcal{S}^{l+1}$ en els vèrtexs de les corresponents cel·les de $\mathcal{T}^{l+1}$.

2. Si $\mathcal{R}^l = SET$, igual que abans es pot refinar la partició $\mathcal{T}^l$ per conjunt usant la cel·la pivot $T_{P^l}^l$. Usant la hipòtesi d'inducció, les cel·les de la partició $\mathcal{T}^l$ tenen les mateixes adjacències amb la cel·la pivot $T_{P^l}^l$ que les corresponents cel·les de $\mathcal{S}^l$ amb la cel·la pivot $S_{P^l}^l$. Per tant, les noves cel·les generades en el refinament tindran les mateixes adjacències amb la cel·la pivot en els dos grafs i l'isomorfisme m aplicarà cel·les de la partició $\mathcal{S}^{l+1}$ en les cel·les corresponents de la partició $\mathcal{T}^{l+1}$. D'aquí es conclou la compatibilitat de les particions $\mathcal{S}^{l+1}$ i $\mathcal{T}^{l+1}$.

3. Si $\mathcal{R}^l = UNKNOWN$, el vèrtex $u \in S_{P^l}^l$ usat com a pivot es pot aplicar, per l'isomorfisme m, a qualsevol vèrtex de la cel·la pivot $T_{P^l}^l$, però només s'aplicarà a un. Sigui $v = m(u)$ aquest vèrtex, aleshores es pot fer un refinament per vèrtex usant v com a pivot que portarà a una partició, $\mathcal{T}^{l+1}$. Aplicant el mateix raonament que el cas de refinament per vèrtex tenim la compatibilitat de les particions $\mathcal{S}^{l+1}$ i $\mathcal{T}^{l+1}$.

D'aquesta manera s'arribaria fins a la partició final t. La compatibilitat de les particions finals $\mathcal{S}^t$ i $\mathcal{T}^t$ es deriva del fet que en aquestes particions totes les cel·les estan aïllades o són unitàries. Com que m aplica vèrtexs de cel·les en vèrtexs de cel·les corresponents es té que si dos vèrtexs u i v pertanyen a cel·les unitàries, $m(u)$ i $m(v)$ pertanyen a les corresponents cel·les unitàries de $\mathcal{T}^t$ i, per tant, tenen les mateixes adjacències. D'aquí se'n dedueix la compatibilitat de $\mathcal{S}^t$ i $\mathcal{T}^t$.

Es considera ara el cas de què existeixin dues seqüències de particions compatibles, Q_G i Q_H, pels grafs G i H, respectivament. Siguin M^G i M^H les matrius d'adjacència dels grafs G i H, respectivament. Siguin $\mathtt{S}_G = (\mathcal{S}^0, ..., \mathcal{S}^t)$ i $\mathtt{S}_H = (\mathcal{T}^0, ..., \mathcal{T}^t)$ les seqüències compatibles. Les particions finals $\mathcal{S}^t = (S_1^t, ..., S_r^t)$ i $\mathcal{T}^t = (T_1^t, ..., T_r^t)$ seran compatibles i, per tant, $\forall x, y \in \{1, ..., s\}$ es tindrà $ADeg(S_x^t, S_y^t, G) = ADeg(T_x^t, T_y^t, H)$. En aquestes particions les cel·les són totes aïllades o són unitàries, per tant, $\forall i, j \in \{1, ..., s\}, s = |V_G^t| = |V_H^t|$, és té $M^G_{\lambda(n-s+i)\lambda(n-s+j)} = M^H_{\lambda'(n-s+i)\lambda'(n-s+j)}$, on λ i λ' són els ordres induïts en les seqüències Q_G i Q_H, respectivament. Per tant, els grafs $G_{V_G^t}$ i $H_{V_H^t}$ són isomorfs i $m(\lambda(i)) = \lambda'(i)$ és un

isomorfisme entre ells. S'aplica ara el mètode d'inducció i es suposa que els grafs $G_{V_G^l}$ i $H_{V_H^l}$ són isomorfs i $m(\lambda(i)) = \lambda'(i)$, restringida als vèrtexs de V_G^l i V_H^l és un isomorfisme entre ells. S'afegeixen ara els vèrtexs $V_G^{l-1} \backslash V_G^l$ i $V_H^{l-1} \backslash V_H^l$ (que provenen de cel·les sense enllaços o s'han utilitzat com a vèrtex pivot en un refinament per vèrtex) per comprovar que els grafs V_G^{l-1} i V_H^{l-1} són isomorfs. Si afegim els vèrtexs aïllats de V_G^{l-1} a V_G^l i els de V_H^{l-1} a V_H^l, m seguirà sent un isomorfisme entre aquests grafs perquè els vèrtexs aïllats no influeixen. Ara, si el refinament de les particions $\mathcal{S}^l$ i $\mathcal{T}^l$ s'ha fet per vèrtex, siguin $u \in V_G^{l-1} \backslash V_G^l$ i $v \in V_H^{l-1} \backslash V_H^l$ els vèrtexs pivots corresponents. De la compatibilitat de les particions es té que $\forall x \in \{1, ..., s\}$, $ADeg(\{u\}, S_x^l, G) = ADeg(\{v\}, T_x^l, H)$ i, per tant, l'aplicació m, restringida als vèrtexs $\{u\} \cup V_G^l$ i $\{v\} \cup V_H^l$, és un isomorfisme entre els grafs $G_{\{u\} \cup V_G^l}$ i $H_{\{v\} \cup V_H^l}$. A més, u precedeix en ordre a tots els vèrtexs de V_G^l i v precedeix en ordre a tots els vèrtexs de V_H^l, d'on $v = m(u)$.

Així, es té que m pot ser extesa a tots els vèrtexs de V_G^{l-1} i V_H^{l-1}. Per tant, m és un isomorfisme entre els grafs G i H. □

4.1.3 Corol·lari. *Siguin $Q_G = \{\mathcal{S}, \mathcal{R}, \mathcal{P}\}$ i $Q_H = \{\mathcal{T}, \mathcal{R}, \mathcal{P}\}$ dues seqüències de particions dels grafs G i H, respectivament, compatibles entre si. Siguin $\mathcal{S} = \{\mathcal{S}^0, ..., \mathcal{S}^t\}$ i $\mathcal{T} = \{\mathcal{T}^0, ..., \mathcal{T}^t\}$. Siguin $\mathcal{S}^i = \{S_1^i, ..., S_{r_i}^i\}$ i $\mathcal{T}^i = \{T_1^i, ..., T_{r_i}^i\}$ per a tot $i \in \{0, ..., t\}$. Per a cada $x \in \{1, ..., |V|\}$, es nota $\lambda(x)$ el vèrtex en la posició x segons l'ordre induït per la seqüència Q_G i $\lambda'(x)$ el vèrtex en la posició x segons l'ordre induït per la seqüència Q_H. Aleshores, $\lambda(x) \in S_k^j$ per a algun $j \in \{0, ..., t\}$ i algun $k \in \{1, ..., r_j\}$ si, i només si, $\lambda'(x) \in T_k^j$.*

4.1.4 Definició. Sigui $G = (V, E)$ un graf i siguin $u, v \in V$ dos vèrtexs del graf. Direm que u i v són *equivalents* si existeix un automorfisme π de G tal que $\pi(u) = v$.

4.1.5 Proposició. *La relació anterior és una relació d'equivalència en el conjunt de vèrtexs.*

Demostració:

1. La identitat pertany a $Aut(G)$. Per tant $Id(u) = u, \forall u \in V$. Tot vèrtex u és equivalent a ell mateix.

2. Siguin $u, v \in V$ de manera que u és equivalent a v. Aleshores existeix $\pi \in Aut(G)$ tal que $v = \pi(u)$. Com que $Aut(G)$ és un grup, l'invers de l'automorfisme π també pertany a $Aut(G)$, $\pi^{-1} \in Aut(G)$ i, per tant $\pi^{-1}(v) = \pi^{-1}\pi(u) = u$. Així u és equivalent a v.

3. Siguin $u, v, w \in V$ tals que v és equivalent a u i w és equivalent a v. Aleshores existeixen $\pi_1, \pi_2 \in Aut(G)$ amb $v = \pi_1(u)$ i $w = \pi_2(v)$. La composició d'aquests dos automorfismes és un altre automorfisme $\pi_3 = \pi_2\pi_1$ que satisfà $\pi_3(u) = \pi_2(\pi_1(u)) = \pi_2(v) = w$. Per tant, u i w també són equivalents.

□

4.1.6 Definició. Sigui $G = (V, E)$ un graf i sigui $Aut(G)$ el seu grup d'automorfismes. Un *conjunt de generadors* de $Aut(G)$ és un subconjunt $\Gamma = \{\gamma_1, ..., \gamma_n\} \subseteq Aut(G)$ tal que qualsevol element $g \in Aut(G)$ es pot escriure com a composició d'elements de Γ: $g = \gamma_{i_1}\gamma_{i_2}...\gamma_{i_r}$ on, per a cada j, $1 \leq i_j \leq n$. Els elements del conjunt Γ s'anomenen *generadors* del grup d'automorfismes i s'escriu $< \Gamma >= Aut(G)$.

4.1.7 Definició. Sigui $G = (V, E)$ un graf i sigui $Aut(G)$ el seu grup d'automorfismes. L'*òrbita* d'un vèrtex $u \in V$ es defineix com el conjunt $Orb(v) = \{\pi(v) | \pi \in Aut(G)\}$.

4.1.8 Observació. Sigui $< \Gamma >= Aut(G)$. Aleshores $\forall u \in V$, $Orb(u) = \{\pi(u) | \pi \in \Gamma\}$.

4.1.9 Definició. Sigui $G = (V, E)$ un graf. La *partició per òrbites* del graf és una partició $O = \{O_1, ..., O_n\}$ de V tal que $\forall i \in \{1, .., n\}$, $u, v \in O_i \Rightarrow u$ i v són equivalents.

4.1.10 Definició. Sigui $G = (V, E)$ un graf i sigui $\Gamma \subseteq Aut(G)$ el generador del seu grup d'automorfismes. Sigui $\Gamma' \subset \Gamma$ un subconjunt del conjunt de generadors. Una *semiòrbita* d'un vèrtex $u \in V$ es defineix com el conjunt $PartialOrb(u, \Gamma') = \{\pi(u) | \pi \in \Gamma'\}$.

4.1.11 Lema. *Sigui $Q_G = \{\mathcal{S}, \mathcal{R}, \mathcal{P}\}$ una seqüència de particions del graf G tal que $\mathcal{S} = (\mathcal{S}^0, ..., \mathcal{S}^t)$, $\mathcal{R} = (\mathcal{R}^0, ..., \mathcal{R}^{t-1})$, $\mathcal{P} = (\mathcal{P}^0, ..., \mathcal{P}^{t-1})$. Siguin $l \in \{0, ..., t\}$, $\mathcal{S}^l = (S_1^l, ..., S_{r_l}^l)$ i $V^l = \cup_{i=0}^{r_l} S_i^l$, tal que existeix algun $k \in \{1, ..., r_l\}$ amb $ADeg(S_k^l, V^l, G) = 0$ i $|S_k^l| > 1$. Aleshores, $\forall u, v \in S_k^l$ u i v són equivalents.*

El lema 4.1.11 garanteix que, si durant la determinació de la seqüència de particions es troba una cel·la sense adjacències, es poden intercanviar dos vèrtexs qualssevol d'aquesta cel·la i s'obté un automorfisme del graf. En l'algorisme *sequence* es crida la funció *discardedUnlinkedCells* que és l'encarregada de preparar els vèrtexs de les cel·les sense links per computar les permutacions pertinents. En general, per obtenir els generadors associats, es guarden com a generadors les permutacions (transposicions) del primer vèrtex de la cel·la sense links amb la resta de vèrtexs d'aquesta cel·la. La resta de permutacions es poden obtenir a partir d'aquestes via composició.

Altres automorfismes poden sorgir quan apareixen particions amb refinaments etiquetats com a *UNKNOWN* durant la generació de la seqüència de particions. A partir d'aquí es poden produir les situacions següents:

1. Que tots els vèrtexs de la cel·la pivot en la partició etiquetada com a *UNKNOWN* siguin equivalents.

2. Que hi hagin alguns vèrtexs no equivalents en la cel·la pivot.

3. Que cap vèrtex de la cel·la pivot sigui equivalent a un altre vèrtex d'aquesta cel·la.

4.1.12 Proposició. *Sigui $Q_G = \{\mathcal{S}, \mathcal{R}, \mathcal{P}\}$ una seqüència de particions del graf G tal que $\mathcal{S} = (\mathcal{S}^0, ..., \mathcal{S}^t)$, $\mathcal{R} = (\mathcal{R}^0, ..., \mathcal{R}^{t-1})$, $\mathcal{P} = (\mathcal{P}^0, ..., P^{t-1})$ amb $\mathcal{R}^l = \{UNKNOWN\}$ per a algun $l \in \{0, ..., t-1\}$. Sigui $\mathcal{S}^l = (S_1^l, ..., S_{r_l}^l)$ i sigui $u \in S_{Pl}^l$ el vèrtex pivot usat per fer el refinament durant la generació de la seqüència de particions en el nivell l. Es pren $v \in S_{\mathcal{P}_l}^l, v \neq u$, un altre vèrtex de la cel·la pivot. Sigui Q'_G la seqüència de particions generada utilitzant el vèrtex v i suposem que és compatible amb la seqüència Q_G. Siguin $\leq_{Q_G}$ i $\leq_{Q'_G}$ els ordres induïts per Q_G i Q'_G en el conjunt de vèrtexs V, respectivament. Siguin $\lambda_{Q_G}(i)$ i $\lambda_{Q'_G}(i)$ els vèrtexs en la posició i-èssima respecte als ordres $\leq_{Q_G}$ i $\leq_{Q'_G}$, respectivament. Es considera l'aplicació m definida com $m(\lambda_{Q_G}(i)) = \lambda_{Q'_G}(i)$ per a tot $i \in \{1, ..., |V|\}$ que és un automorfisme de G. Aleshores existeix $k \in \{1, ..., |V|\}$ tal que $u = \lambda_{Q_G}(k)$ i $v = \lambda_{Q'_G}(k)$ i es satisfà que $\forall j \in \{k, ..., |V|\}$, $\lambda_{Q_G}(j)$ i $\lambda_{Q'_G}(j)$ són equivalents, i m fixa tots els vèrtexs $\lambda_{Q_G}(1), ..., \lambda_{Q_G}(k-1)$.*

4.1.13 Lema. *Si dos vèrtexs u i v són equivalents en el nivell l, aleshores són equivalents en tots els nivells $i \in \{0, ..., l-1\}$.*

4.1.14 Observació. Quan es troben dos vèrtexs equivalents en un determinat nivell, es pot assegurar que tots els vèrtexs de les seves respectives semiòrbites són equivalents, perquè es tracta d'una relació d'equivalència. Per tant, quan es troba una relació així, es modifica la partició d'òrbites per unir les semiòrbites dels vèrtexs equivalents.

4.1.15 Lema. *Siguin G i H dos grafs i siguin Q_G i Q_H dues seqüències de particions compatibles de G i H, respectivament. Si tots els vèrtexs d'una cel·la pivot en la seqüència Q_G són equivalents, aleshores els vèrtexs de la corresponent cel·la pivot en la seqüència Q_H també són equivalents.*

4.2 Cerca d'automorfismes

L'algorisme 7 cerca els automorfismes del graf G. Primer crea un grup de permutacions amb el nombre de vèrtexs del graf i posa com a base de les permutacions el vector de vèrtexs ordenats segons l'ordenació induïda per la seqüència de particions Q_G trobada. Aquesta base la col·loca en el vector de vèrtexs descartats DV. A continuació, començant per l'última partició i ascendint fins a la primera, grava els generadors dels automorfismes trobats gràcies a les celles que no tenen adjacències en cada partició usant la funció *unlinked_cells_transpositions($\mathcal{S}^i$)*. Després de gravar els automorfismes trobats gràcies a les cel·les sense adjacències de la partició que està tractant, explora la partició si està etiquetada com *UNKNOWN* en ordre ascendent i crida la funció *search_automorphism($\mathcal{S}^i$)* que cerca automorfismes en els possibles nivells de tipus *BACKTRACK*. Els automorfismes es guarden en un fitxer com seqüències ordenades dels vèrtexs. D'aquesta forma, un automorfisme vindrà donat per la posició dels vèrtexs en cada seqüència alternativa compatible respecte a la seqüència original o base.

algorisme 7 Cerca d'automorfismes d'un graf.

void GIso::Automorphisms()

1: $\Gamma \leftarrow Id$
2: $DV \leftarrow setBase()$
3: **for** $(i = t; i \leq 0; i--)$ **do**
4: $\quad \Gamma \leftarrow \Gamma \cup unlinked\_cells\_transpositions(\mathcal{S}^i)$;
5: $\quad$ **if** $(\mathcal{R}^i = UNKNOWN)$ **then**
6: $\quad\quad \mathcal{R}^i \leftarrow$ search_automorphism($\mathcal{S}^i$);
7: $\quad$ **end if**
8: **end for**
9: **return**

En l'algorisme *sequence* es cridaven dues funcions: *backtrackPartitions* i *limitPartitions*. Aquests funcions s'encarreguen de determinar per a cada partició o nivell $l \in \{0, ..., t-1\}$ el seu nivell de backtracking i el seu nivell límit, respectivament.

4.2.1 Definició. Sigui $Q_G = \{\mathtt{S}, \mathtt{R}, \mathtt{P}\}$ una seqüència de particions del graf G tal que $\mathtt{S} = (\mathcal{S}^0, ..., \mathcal{S}^t)$, $\mathtt{R} = (\mathcal{R}^0, ..., \mathcal{R}^{t-1})$, $\mathtt{P} = (\mathcal{P}^0, ..., \mathcal{P}^{t-1})$. Donada una partició $\mathcal{S}^i, i \in \{0, ..., t-1\}$ el seu *nivell de backtracking* es defineix com l'índex de partició $B^i \in \{0, ..., i-1\}$ tal que si $\mathcal{R}^i \neq BACKTRACK, B^i = i-1$ i si $\mathcal{R}^i = BACKTRACK$ aleshores B^i és l'índex de la

primera partició, en ordre ascendent, anterior a i ($i > B^i$) tal que $\mathcal{R}^{B^i} = BACKTRACK$ i $\mathcal{S}^i$ no és subpartició de $\mathcal{S}^{B^i}$. Si cap partició no satisfà les condicions, s'agafa $B^i = 0$.

4.2.2 Definició. Sigui $Q_G = \{\mathtt{S}, \mathtt{R}, \mathtt{P}\}$ una seqüència de particions del graf G tal que $\mathtt{S} = (\mathcal{S}^0, ..., \mathcal{S}^t)$, $\mathtt{R} = (\mathcal{R}^0, ..., \mathcal{R}^{t-1})$, $\mathtt{P} = (\mathcal{P}^0, ..., \mathcal{P}^{t-1})$. Donada una partició $\mathcal{S}^i, i \in \{0, ..., t-1\}$ el seu *nivell límit* es defineix com l'índex de partició $L^i \in \{i+1, ..., t\}$ tal que si $\mathcal{R}^i \neq BACKTRACK, L^i = t$ (última partició) i, si $\mathcal{R}^i = BACKTRACK$, aleshores L^i és l'índex de la primera partició, en ordre descendent, posterior a i ($i < L^i$) tal que $\mathcal{R}^{L^i} = BACKTRACK$ i $\mathcal{S}^{L^i}$ és subpartició de $\mathcal{S}^i$.

Suposem que $Q = \{\mathtt{S}, \mathtt{R}, \mathtt{P}\}$ una seqüència de particions del graf G tal que $\mathtt{S} = (\mathcal{S}^0, ..., \mathcal{S}^t)$, $\mathtt{R} = (\mathcal{R}^0, ..., \mathcal{R}^{t-1})$, $\mathtt{P} = (\mathcal{P}^0, ..., \mathcal{P}^{t-1})$ i que tenim dues particions $\mathcal{S}^k$ i $\mathcal{S}^l$, $0 \leq k < l \leq t$, equitatives, tals que $\mathcal{S}^l$ és subpartició de $\mathcal{S}^k$ i no existeix cap altre subpartició de $\mathcal{S}^k$ entre les dues. Siguin $p \in \mathcal{S}^k_{\mathcal{P}^k}$ el vèrtex pivot usat en el refinament en la seqüència Q en el nivell k i $q \in S^k_{\mathcal{P}^k}, q \neq u$ un altre vèrtex de la cel·la pivot en el nivell k. Suposem que fent un refinament amb el vèrtex q obtenim una seqüència de particions $Q' = (\mathcal{S}^0, ..., \mathcal{S}^k, \mathcal{T}^{k+1}, ..., \mathcal{T}^l), (\mathcal{R}^0, ..., \mathcal{R}^{l-1}), (\mathcal{P}^0, ..., \mathcal{P}^{l-1})$ que és compatible amb la seqüència original Q.

Siguin $V^k = \cup_{i=0}^{r_k} S^k_j, V^l = \cup_{i=0}^{r_l} S^l_j, W^l = \cup_{i=0}^{r_l} T^l_j$. A partit d'aquests conjunts obtenim els conjunts $\hat{V}^k, \hat{V}^l, \hat{W}^l$ eliminant dels conjunts originals totes aquelles cel·les sense adjacències. En aquestes condicions, $|\mathcal{S}^l| = |\mathcal{T}^l| \leq |\mathcal{S}^k|$ perquè són subparticions de $\mathcal{S}^k$ i de cada cel·la de $\mathcal{S}^k$ com a molt en pot sorgir una en una subpartició seva. Podem suposar $|\mathcal{S}^l| = |\mathcal{T}^l| = |\mathcal{S}^k| = r$ si considerem que algunes cel·les de $\mathcal{S}^l$ i $\mathcal{T}^l$ poden estar buides. Aleshores definim els conjunts següents:

- $E_i = S^k_i \backslash \hat{V}^l$. Conjunt de vèrtexs de la cel·la S^k_i descartats entre les particions k i l en la seqüència original Q per no tenir adjacències.
- $E'_i = S^k_i \backslash \hat{W}^l$. Conjunt de vèrtexs de la cel·la S^k_i descartats entre les particions k i l en la seqüència alternativa Q' per no tenir adjacències.
- $A_i = E_i \cap E'_i$. Conjunt de vèrtexs de la cel·la S^k_i descartats entre les particions k i l en les dues seqüències.
- $B_i = E_i \backslash A_i$. Conjunt de vèrtexs de la cel·la S^k_i descartats entre les particions k i l en la seqüència original Q però no en l'alternativa.
- $B'_i = E'_i \backslash A_i$. Conjunt de vèrtexs de la cel·la S^k_i descartats entre les particions k i l en la seqüència alternativa Q' però no en l'original.
- $D_i = S^k_i \cap \hat{V}^l \cap \hat{W}^l$. Conjunt de vèrtexs de la cel·la S^k_i no descartats entre les particions k i l en cap de les dues seqüències.
- $E = \hat{V}^k \backslash \hat{V}^l$. Conjunt de vèrtexs no aïllats de la partició $\mathcal{S}^k$ descartats entre les particions k i l en la seqüència original Q per no tenir adjacències.
- $E' = \hat{V}^k \backslash \hat{W}^l$. Conjunt de vèrtexs no aïllats de la partició $\mathcal{S}^k$ descartats entre les particions k i l en la seqüència alternativa Q' per no tenir adjacències.
- $A = E \cap E'$. Conjunt de vèrtexs no aïllats de la partició $\mathcal{S}^k$ descartats entre les particions k i l en les dues seqüències.

- $B = E \backslash A$. Conjunt de vèrtexs no aïllats de la partició $\mathcal{S}^k$ descartats entre les particions k i l en la seqüència original Q però no en l'alternativa.
- $B' = E' \backslash A$. Conjunt de vèrtexs no aïllats de la partició $\mathcal{S}^k$ descartats entre les particions k i l en la seqüència alternativa Q' però no en l'original.
- $D = \hat{V}^l \cap \hat{W}^l$. Conjunt de vèrtexs no aïllats de la partició $\mathcal{S}^k$ no descartats entre les particions k i l en cap de les dues seqüències.

4.2.3 Lema. *Siguin λ l'orde induït per la seqüència Q i λ' l'orde induït per la seqüència Q'. Per a tot $u \in V \backslash \hat{V}^l$, sigui $u = \lambda(i)$ i sigui $u' = \lambda'(i)$. Aleshores, $\forall j \in \{1, ..., r\}$ i $\forall v, w \in S_j^l, v', w' \in T_j^l$, $M_{uv} = M_{uw} = M_{u''v'} = M_{u'w'}$, on M és la matriu d'adjacència del graf.*

4.2.4 Corol·lari. *Sigui $a \in A$ un vèrtex descartat en les dues seqüències entre les particions k i l. Aleshores, $\forall b \in B_i, b' \in B_i'$, $M_{ab} = M_{ab'}$.*

Demostració: Siguin $a \in A, b \in B_i, d \in D_i$. Tenim que $a \in V \backslash \hat{W}^l$ i $b, d \in T_i^l$, ja que ni b ni d han estat descartats en la seqüència alternativa. A partir del lema anterior es té que $M_{ab} = M_{ad}$. De la mateixa manera $\forall b' \in B_i$, $M_{ab'} = M_{ad}$, d'on $M_{ab} = M_{ab'}$. □

4.2.5 Lema. *Per a tot $i, j \in \{1, ..., r\}$ i per a tot $u \in B_i, v \in B_i', w \in D_i, u' \in B_j, v' \in B_j', w' \in D_j$ es satisfà que $M_{uv'} = M_{uw'} = M_{vu'} = M_{vw'} = M_{wu'} = M_{wv'}$, on M és la matriu d'adjacència del graf.*

Demostració: Les igualtats $M_{uv'} = M_{uw'}$ i $M_{vu'} = M_{vw'}$ són consequüència directa del lema 4.2.3. La igualtat $M_{wu'} = M_{wv'}$ també es pot deduir del lema 4.2.3 tenint en compte que, encara que la matriu d'adjacència M no és simètrica, es satisfà que si $M_{uv} = M_{uw}$, aleshores $M_{vu} = M_{wu}$. Per tant, pel lema 4.2.3, $M_{vu'} = M_{wu'}$, igualtat aplicada a la seqüència original, i que $M_{uv'} = M_{wv'}$ aplicada a la sequüència alternativa. □

4.2.6 Definició. Siguin λ l'orde induït per la seqüència Q i λ' l'orde induït per la seqüència Q' en els vèrtexs $V \backslash \hat{V}^l$. Es defineix l'etiquetatge $\gamma : \{1, ..., |V|\} \longrightarrow V$ com:

$$\gamma(i) = \begin{cases} \lambda'(i) & si \quad 1 \leq i \leq |V| - |\hat{V}^l| \\ f(i) & si \quad |V| - |\hat{V}^l| < i \leq |V| \end{cases} \quad \text{on } f(i) = \begin{cases} \lambda(i) & si \quad \lambda(i) \notin E' \\ f((\lambda')^{-1}(\lambda(i))) & si \quad \lambda(i) \in E' \end{cases}$$

4.2.7 Teorema. *L'aplicació $m : V \longrightarrow V$ tal que $m(\lambda(i)) = \gamma(i)$ defineix un automorfisme del graf G.*

Demostració: Cal demostrar que $\forall u, v \in V$ es satisfà que $M_{uv} = M_{m(u)m(v)}$. Siguin $u = \lambda(i)$ i $v = \lambda(j)$. Es consideren els casos següents:

1. $u, v \in V \backslash \hat{V}^l \Rightarrow m(u) = \lambda'(i), m(v) = \lambda'(j)$. Com que les seqüències Q i Q' són compatibles m defineix un automorfisme en el graf $G_{V \backslash \hat{V}^l}$ i, per tant, $M_{uv} = M_{m(u)m(v)}$.
2. $u \in V \backslash \hat{V}^l, v \in \hat{V}^l \backslash E' \Rightarrow v \in \hat{W}^l \Rightarrow v \in D \Rightarrow m(v) = f(j) = \lambda(j) = v$. Pel lema 4.2.3, $M_{uv} = M_{\lambda'(i)v}$.

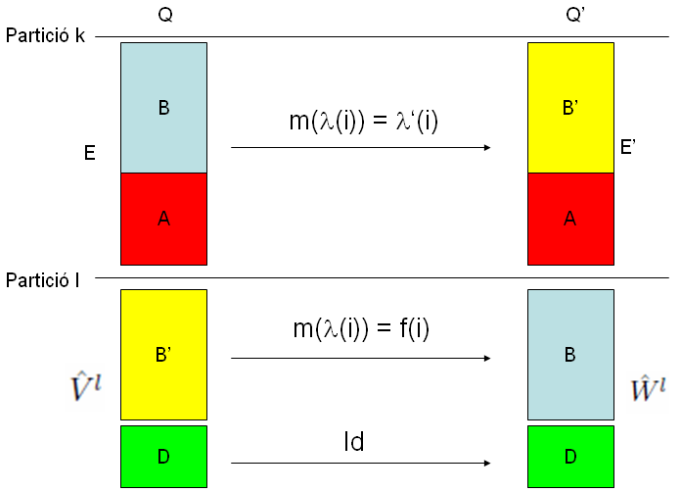


Figura 4.1: Definició de la funció γ

3. Si $u \in V \backslash \hat{V}^l, v \in \hat{V}^l \cap E'$, és a dir que $v \in B'$, aleshores $f(j) = f((\lambda')^{-1}(\lambda(j))) \equiv v_1$, per tant, $m(v_1) = v$. Siguin $\{v_1, ..., v_n\} \subseteq A$ tals que $\forall i, 2 \leq i \leq n-1, m(v_{i+1}) = v_i$ i $m(v') = v_n$ amb $v' \in B$, per tant, $v' = m(v) \in \hat{W}^l$. A més, $v \in S_s^l \Rightarrow v' \in T_s^l$ i, pel lema 4.2.3, s'obté que $M_{uv} = M_{\lambda'(i)v'}$.

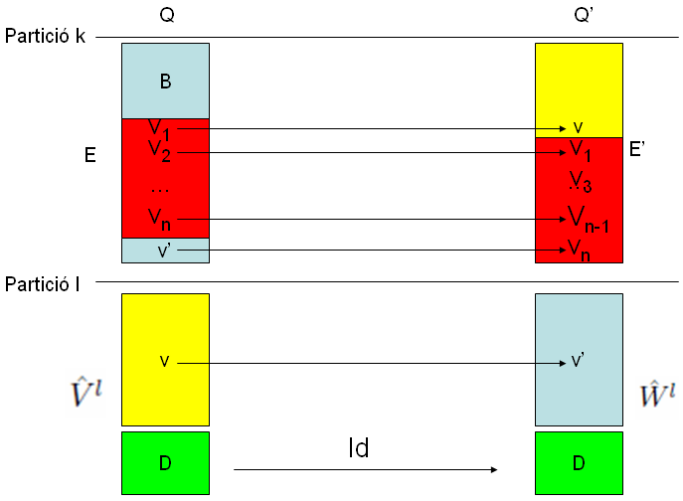


Figura 4.2: Cas particular d'automorfisme

4. $u, v \in \hat{V}^l \cap D \Rightarrow m(u) = u, m(v) = v$, això és, l'automorfisme trivial.

5. $u, v \in \hat{V}^l, u \in D, v \notin D \Rightarrow m(u) = u, v \in E'$. Aplicant el mateix raonament que en el cas anterior per al vèrtex v, es pot generar una successió $\{v_1, ..., v_n\} \subseteq A$ tal que $m(v_{i+1}) = v_i$ i $m(v') = v_n$, amb $v' \in B$. Aleshores, com que $v_i \in A$, pel lema 4.2.3, $M_{v_1 u} = M_{vu}$, $M_{v_2 u} = M_{v_1 u}$, ..., $M_{v_n u} = M_{v_{n-1} u}$, $M_{v'u} = M_{v_n u} \Rightarrow M_{vu} = M_{v'u}$.

6. Si $u, v \in \hat{V}^l \backslash D$, suposant que $u \in S_s^l$ i $v \in S_r^l$ i es té que $u, v \in B'$. Aplicant el raonament

anterior hi ha dos vèrtexs $u', v' \in B \cap (\hat{W}^l \backslash D)$ tals que $u' = m(u) \in T_s^l$ i $v' = m(v) \in T_r^l$. De la compatibilitat de les seqüències Q i Q' fins a nivell l, es té que $M_{uv} = M_{u'v'}$

□

4.2.8 Teorema. *Siguin $G = (V_G, E_G)$ i $H = (V_H, E_H)$ dos grafs i $Q_G = \{\mathtt{S}, \mathtt{R}, \mathtt{P}\}$ una seqüència de particions completa del graf G tal que $\mathtt{S} = (\mathcal{S}^0, ..., \mathcal{S}^t)$, $\mathtt{R} = (\mathcal{R}^0, ..., \mathcal{R}^{t-1})$, $\mathtt{P} = (\mathcal{P}^0, ..., \mathcal{P}^{t-1})$. Sigui $((\mathcal{T}^0, ..., \mathcal{T}^l), (\mathcal{R}^0, ..., \mathcal{R}^{l-1}), (\mathcal{P}^0, ..., \mathcal{P}^{l-1}))$ una seqüència de particions del graf H tal que $l \leq t$ i $\mathcal{T}^l$ és equitativa i compatible amb amb la seqüència de particions $((\mathcal{S}^0, ..., \mathcal{S}^l), (\mathcal{R}^0, ..., \mathcal{R}^{l-1}), (\mathcal{P}^0, ..., \mathcal{P}^{l-1}))$ del graf G. Suposem que no existeix cap seqüència de particions completa del graf H que comenci per $((\mathcal{T}^0, ..., \mathcal{T}^l), (\mathcal{R}^0, ..., \mathcal{R}^{l-1}), (\mathcal{P}^0, ..., \mathcal{P}^{l-1}))$ i sigui compatible amb Q_G. Sigui $k, 0 \leq k < l$ tal que $\mathcal{T}^l$ és una subpartició de $\mathcal{T}^k$. Aleshores no existeix cap seqüència de particions completa del graf H que comenci amb la seqüència $((\mathcal{T}^0, ..., \mathcal{T}^k), (\mathcal{R}^0, ..., \mathcal{R}^{k-1}), (\mathcal{P}^0, ..., \mathcal{P}^{k-1}))$ i sigui compatible amb Q_G.*

DEMOSTRACIÓ: Se suposa que existeix una seqüència completa del graf H que comença amb la seqüència $((\mathcal{T}^0, ..., \mathcal{T}^k), (\mathcal{R}^0, ..., \mathcal{R}^{k-1}), (\mathcal{P}^0, ..., \mathcal{P}^{k-1}))$ i és compatible amb una seqüència completa del graf G. Pel teorema 4.2.7, com que $\mathcal{T}^l$ és subpartició de $\mathcal{T}^k$, es pot trobar una seqüència completa compatible que comença amb $((\mathcal{T}^0, ..., \mathcal{T}^l), (\mathcal{R}^0, ..., \mathcal{R}^{l-1}), (\mathcal{P}^0, ..., \mathcal{P}^{l-1}))$ i, així, arribem a contradicció. □

El teorema 4.2.8 permet estalviar temps de càlcul reduint l'espai a explorar perquè el programa *Conauto* comença la cerca d'automorfismes per l'últim punt de *BACKTRACKING* trobat i va pujant cap amunt. D'aquesta manera, com que s'ha computat prèviament el *nivell límit* de cada partició es pot saber quins nivells de tipus *BACKTRACK* no cal explorar perquè el teorema 4.2.8 ens garanteix que no aportaran cap automorfisme.

algorisme 8 Cerca d'automorfismes d'un graf a partir de la partició $\mathcal{S}$.

(refinement type) GIso::search_automorphism(Partition $\mathcal{S}$)

```
1: R ← VERTEX
2: S_P ← pivotCellOf(S)
3: p ← pivotVertexOf(S_P)
4: for all u ∈ (S_P\{p}) do
5:    if (OrbitOf(u) ≠ OrbitOf(p)) then
6:       (T, Γ', f) ← generate_automorphism(S, u, Γ);
7:       if ( T ≠ S) then
8:          Γ ← Γ ∪ Γ';
9:       else
10:         mark_new_mismatch( S, mismatch_marked, go.vertices, k )
11:         R ← BACKTR
12:      end if
13:   else
14:      mark_old_mismatch( S, mismatch_marked, go.vertices, k );
15:   end if
16: end for
17: return R;
```

algorisme 9 Genera automorfismes d'un graf a partir de la partició $\mathcal{S}$.

Partition GIso::generate_automorphism($\mathcal{S}^l$, u, Γ)

1: **if** ($vertex\_ref\_compat(G, \mathcal{S}^l, \mathcal{S}^l)$) **then**
2: $(\mathcal{T}, \Gamma', f) \leftarrow subtree\_compat(\mathcal{S}^{l+1}, E, \mathcal{T}', l)$;
3: **return** $(\mathcal{T}, \Gamma', f)$;
4: **end if**
5: **return** $(\mathcal{S}^l, \Gamma, f)$;

L'algorisme 9 genera els automorfismes del graf G cridant la funció recursiva subtree_compat.

L'algorisme primer fa un nou refinament per vèrtex de la partició equitativa, $\mathcal{S}^l$, usant un vèrtex $u \in S^l_{\mathcal{P}_l}$ que no és el pivot original. La funció encarregada d'aquesta tasca és *vertex_ref_compat*, que retorna TRUE si aconsegueix fer un refinament compatible amb l'original, i a continuació, crida la funció recursiva *subtree_compat*, que procedeix a fer refinaments successius. Si el refinament per vèrtex no ha resultat compatible amb la partició original, es retorna la partició S^l inicial.

L'algorisme 10 és una algorisme recursiu que tracta d'obtenir una seqüència de particions alternativa compatible amb l'original a partir del nivell l. Primer comprova si la partició original $\mathcal{S}^l$ i l'alternativa $\mathcal{T}^l$ són iguals, perquè, en aquest cas, la resta de particions també ho seran i no cal continuar. Després comprova si la partició $\mathcal{T}^l$ és la partició límit de $\mathcal{S}^k$, i.e., és subpartició de $\mathcal{S}^k$ i és equitativa, i que no sigui l'última partició, aleshores s'aplica el teorema 4.2.7 que permet trobar de forma més ràpida els automorfismes mitjançant la funció *process_subpartition*. Si no es satisfan les condicions anteriors es comprova si l és l'última partició i si $\mathcal{S}^l$ i $\mathcal{T}^l$ són compatibles i en cas afirmatiu s'aplica el teorema 4.1.2 per trobar un automorfisme a partir de l'ordre induït per la seqüència de particions alternativa. En altre cas es comprova el tipus de refinament utilitzat per refinar la partició original $\mathcal{S}^l$ i si es *BACTRACKING* s'aprofundeix en el refinament usant la funció *deepen_in_backtrack*. En cas que el refinament sigui de tipus *SET* o *VERTEX* es fa un refinament del mateix tipus usant les funcions *set_ref_compat* o *vertex_ref_compat*, respectivament. Si en el refinament s'obté una partició compatible amb $\mathcal{S}^{l+1}$, es torna a aprofundir en el refinament tornant a cridar la funció *subtree_compat* (recursivament). En el cas que el tipus de refinament sigui *VERTEX* i no s'aconsegueixi una seqüència compatible, es grava una errada de valor (-1) perquè suposadament no hauríem d'haver trobat un error ja que, en un moment donat, es va canviar el tipus de refinament de *UNKNOWN* a *VERTEX* perquè tots els vèrtexs de la cel·la havien resultat equivalents. En cas que l'algorisme no hagi tingut èxit en el refinament, retorna el nivell de *backtracking* de la partició original $\mathcal{S}^l$. L'algorisme retorna una terna $(\mathcal{S}, \Gamma, f)$ on $\mathcal{S}$ és la partició fins on s'ha arribat en el refinament alternatiu, Γ és el conjunt de generadors trobat fins al moment i f és l'error trobat, si n'hi ha.

L'algorisme 11 realitza el procés de bactracking. Comença per calcular les òrbites parcials dels vèrtexs usant la funció *generate_partial_orbits* i selecciona els vèrtexs vàlids usant la funció *partial_valid_indices*. Tot seguit marca les òrbites vàlides. Seguidament tracta de fer un refinament en el punt de *BACKTRACKING* usant un vèrtex de la cel·la pivot, u, diferent del pivot original, però abans marca l'òrbita d'aquest vèrtex com no vàlida per tal d'evitar futurs refinaments amb vèrtexs de la mateixa òrbita. Si el refinament per vèrtex fet per la funció *vertex_ref_compat* proporciona una partició $\mathcal{T}^{l+1}$ compatible amb la corresponent de la seqüència original es crida la funció recursiva *subtree_compat* que segueix amb el backtracking.

algorisme 10 Funció recurrent que cerca seqüències alternatives a partir de la partició $\mathcal{S}$.

(Partition, Set of generators, integer) GIso::subtree_compat($\mathcal{S}^l, E, T, k$)

1: $f \leftarrow 0$
2: **if** ($\mathcal{S}^l = \mathcal{T}^l$) **then**
3: **for all** $i \in \{l+1, ..., t\}$ **do**
4: $\mathcal{T}^i \leftarrow S^i$
5: **end for**
6: $\pi \leftarrow InducedOrder((\mathtt{T}, \mathtt{R}, \mathtt{P}))$
7: **return** $\mathcal{S}^l$
8: **end if**
9: **if** ($l = L^k$ && $l \neq t$) **then**
10: $\pi \leftarrow process\_subpartition(\mathcal{S}, \mathcal{T})$
11: **return** $\mathcal{S}^l$
12: **end if**
13: **if** ($\mathcal{S}^l$ is compatible with $\mathcal{T}^l$ && $l = t$) **then**
14: $\pi \leftarrow InducedOrder((\mathtt{T}, \mathtt{R}, \mathtt{P}))$
15: **return** $\mathcal{S}^l$
16: **end if**
17: **switch** ($\mathcal{R}^l$)
18: **case** "BACKTRACKING":
19: **return** deepen_in_backtrack(l, E, T, k)
20: **break**
21: **case** "SET":
22: **if** (set_ref_compat(G, $\mathcal{S}^l$, $\mathcal{T}^l$) **then**
23: $(\mathcal{T}', \Gamma, f) \leftarrow subtree\_compat(S^{l+1}, E, T, k)$
24: **if** ($\mathcal{T}' \neq \mathcal{S}^l$) **then**
25: **return** $\mathcal{T}'$
26: **end if**
27: **end if**
28: **break**
29: **case** "VERTEX":
30: **if** (vertex_ref_compat(G, $\mathcal{S}^l$, $\mathcal{T}^l$) **then**
31: $(\mathcal{T}', \Gamma, f) \leftarrow subtree\_compat(S^{l+1}, E, T, k)$;
32: **if** ($\mathcal{T}' \neq \mathcal{S}^l$) **then**
33: **return** $\mathcal{T}'$
34: **end if**
35: **end if**
36: **if** ($|\mathcal{T}^l_{\mathcal{P}^l}| > 1$) **then**
37: $f \leftarrow (-1)$
38: **end if**
39: **break**
40: **end switch**
41: **return** $(\mathcal{S}^{B^l}, \Gamma, f)$;

algorisme 11 Funció recurrent que cerca seqüències alternatives a partir de la partició $\mathcal{S}$.
(Partition, Set of generators, integer) GIso::deepen_in_backtrack(l, E, T, k)

1: $F \leftarrow \emptyset$
2: $(O, \Delta') \leftarrow generate\_partial\_orbits(\mathcal{T}, \Delta, k, l)$;
3: $\mathcal{I} \leftarrow partial\_valid\_indices(\mathcal{S}, O)$;
4: **for all** $u \in (\mathcal{T}^l_{\mathcal{P}_l} \cap \mathcal{I})$ **do**
5: **if** $(valid(orbitOf(u, O)))$ **then**
6: $valid(orbitOf(u, O))) \leftarrow FALSE$
7: **if** (vertex_ref_compat(G, $\mathcal{S}^l$, $\mathcal{T}^l$) **then**
8: $(\mathcal{T}', \Gamma', f) \leftarrow subtree\_compat(S^{l+1}, E, T, k)$;
9: **if** ($\mathcal{T}' \neq \mathcal{S}^l$) **then**
10: **return** $(\mathcal{T}', \Gamma', f)$
11: **end if**
12: **end if**
13: $F \leftarrow F \bigcup \{\cup_{i=1}^{|orbitOf(u,O)|}\{f\}\}$
14: **if** ($F \nsubseteq F^l$) **then**
15: **return** $(\mathcal{S}^{B^l}, \Gamma, f)$
16: **end if**
17: **end if**
18: **end for**
19: **return** $(\mathcal{S}^{B^l}, \Gamma, f)$;

Si la funció *subtree_compat* té èxit en el refinament retorna una partició amb un nivell $l' > l$ i, aleshores, la funció *deepen_in_backtrack* retorna aquesta partició. En cas que la funció *subtree_compat* no se'n surti, es grava $|orbitOf(u, O)|$ (nombre de vèrtexs de l'òrbita del vèrtex usat com a pivot en el refinament alternatiu) vegades l'errada en el multi conjunt d'errades F i es comprova que aquest no estigui inclòs en el multiconjunt F^l; si hi és es retorna la partició corresponent al nivell de backtracking de la partició S^l. Si el multiconjunt d'errades no està inclòs en el multiconjunt F^l, es prova de refinar amb el vèrtex següent de la cel·la pivot que sigui d'una òrbita vàlida.

Quan l'algorisme no aconsegueix fer cap refinament per vèrtex amb cap dels vèrtexs de la cel·la pivot que sigui compatible amb el de la seqüència original, es retorna la partició corresponent al nivell de backtracking de la partició $\mathcal{S}^l$, des de la qual s'ha cridat la funció originalment, per tal de fer el backtracking.

L'algorisme 12 s'encarrega de reorganitzar el grup d'automorfismes del graf. Comença amb un multiconjunt de generadors $\Delta = \{\Gamma^1, ..., \Gamma^n\}$ i posa tots els vèrtexs en una òrbita individual. Tot seguit, per a cada conjunt de generadors Γ^j crea un nou conjunt Γ' on posa el primer generador de Γ^j, γ^j_1, que s'anomena líder. Després repassa tots els generadors de Γ^j per cercar aquells $\gamma' \in \Gamma^j$ tals que $((\gamma')^{-1} \circ \gamma^j_1)(u) = u, \forall u \in E'$ (fixen tots els vèrtexs u que han estat descartats a la seqüència alternativa entre els nivells k i l) i posa tots els vèrtexs de la bestCell del nivell l, $\mathcal{T}^l_{\mathcal{P}^l}$, que tenen la mateixa antiimatge pels dos generadors γ^j_1 i γ' en la mateixa òrbita. Finalment, si Γ' té més d'un element vol dir que hem trencat el conjunt original de generadors i s'afegeix aquest nou conjunt de generadors Γ' al nou multiconjunt Δ'. Un cop

algorisme 12 Funció que uneix òrbites de vèrtexs de la bestCell equivalents.

(Orbits, Multiset of generators) GIso::generate_partial_orbits($\mathtt{T}, \Delta, k, l$)

1: $\Delta' \leftarrow \emptyset$
2: $O \leftarrow \{\{v_i\} : v_i \in V\}$
3: **for all** ($\Gamma^j \in \Delta$ && $|\Gamma^j| > 1$) **do**
4: $\Gamma' \leftarrow \gamma_1^j$
5: **for all** $\gamma' \in \Gamma^j \backslash \gamma_1^j$ **do**
6: **if** ($\forall u \in E', (\gamma_1^j)^{-1}(u) = (\gamma')^{-1}(u)$) **then**
7: $\Gamma' \leftarrow \Gamma' \cup \{\gamma'\}$
8: **for all** ($u, v \in T^l_{\mathcal{P}^l}$ | $(\gamma')^{-1}(u) = (\gamma_1^j)^{-1}(v)$) **do**
9: $O \leftarrow merge(O, orbitOf(u, O), orbitOf(v, O))$
10: **end for**
11: **end if**
12: **end for**
13: **if** ($|\Gamma'| > 1$) **then**
14: $\Delta' \leftarrow \Delta' \cup \{\Gamma'\}$
15: **end if**
16: $\Gamma^j \leftarrow \Gamma^j \backslash \Gamma'$
17: **end for**
18: **return** (O, Δ')

analitzats tots els conjunts de generadors que hi ha en el multiconjunt Δ, l'algorisme retorna el conjunt d'òrbites O i el nou multiconjunt de generadors Δ'.

En la figura 4.3 es mostra una part de l'arbre de cerca d'automorfismes on hi ha 4 particions equitatives en els nivells k_1, k_2, k_3 i k_4. Es representa una seqüència de particions original i vàries alternatives generades a partir dels vèrtexs de la cel·la pivot en la partició equitativa del nivell k_4. Els punts representen les particions i els de color blau serien particions equitatives. L'algorisme de generació d'automorfismes començaria a fer la cerca a partir de la partició k_4 provant tots els vèrtexs de la cel·la pivot diferents del vèrtex original utilitzat per fer el refinament. Per a cada vèrtex, fa un refinament per vèrtex i, si obté una partició compatible amb l'original, crida la funció *subtree_compat*, que intenta fer un refinament, fins a l'última partició, del mateix tipus que en la seqüència original, cridant-se a si mateixa de forma recursiva en cada refinamet que fa. En la figura es veu com trobaria 4 automorfismes; les boles vermelles indican que hi ha una imcompatibilitat en la seqüència i, per tant, no pot trobar una seqüència compatible amb l'original per aquella branca de l'arbre. En cas de fallada, l'algorisme retorna el nivell de *BACKTRACK* de la partició on s'ha trobat la incompatibilitat; en l'exemple representat en la figura, es troba la incompatibilitat en una partició que no és de tipus *BACKTRACK* i, per tant, es retorna la partició anterior (back) com a nivell de *BACKTRACK*. La crida a l'algorisme des de la partició anterior rep aquesta partició i retorna el seu propi nivell de *BACKTRACK*, que correspondria a la seva partició precedent; d'aquesta manera es va retornant la partició anterior en cadascuna de les crides recursives fins arribar a la crida feta en el nivell k_4, que rebria la partició $\mathcal{S}^{k_4}$ en la seva crida i, per tant, detectaria que no ha tingut èxit en la generació d'un automorfisme i provaria el vèrtex següent de la cel·la pivot.

Els automorfismes trobats fixarien els vèrtexs de la seqüència original fins a nivell menor que k_4 i permutarien els vèrtexs de l'original descartats a partir de k_4 amb els vèrtexs descartats en cada alternativa a partir de k_4 segons l'ordre en què hagin estat descartats.

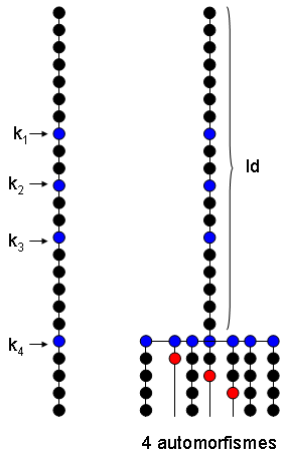


Figura 4.3: Seqüència de particions i alternatives amb reetiquetatge a *BACKTRACK*

En la figura 4.4 es pot observar una situació similar a l'anterior amb la diferència que, en aquest cas, tots els vèrtexs de la cel·la pivot han generat una seqüència alternativa compatible amb l'original i, per tant, l'etiqueta de la partició $\mathcal{S}^{k_4}$ es canvia de *UNKNOWN* a *VERTEX*.

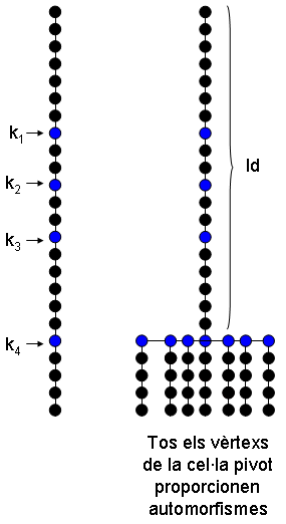


Figura 4.4: Seqüència de particions i alternatives amb reetiquetatge a *VERTEX*

En la figura 4.5 tindríem un cas en què no es troba cap seqüència alternativa compatible amb l'original i, per tant, no es troben automorfismes.

Un cop l'algorisme de cerca d'automorfismes ha explorat l'últim nivel *UNKNOWN*, en aquest cas el k_4, l'algorisme 7 passa a fer la cerca d'automorfismes des del nivell *UNKNOWN* anterior fins al final. En la figura 4.6 s'observa com l'algorisme explora la primera branca de l'arbre, a partir de la partició de nivell k_3, i, quan arriba a la partició k_4, que també és equitativa, torna a provar tots els vèrtexs de la cel·la pivot per intentar arribar fins al final i trobar una

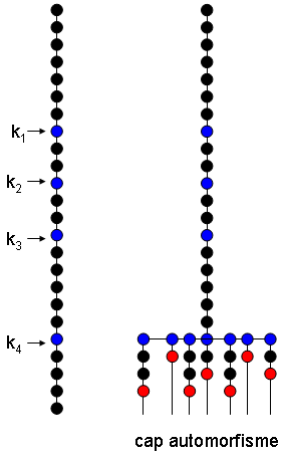


Figura 4.5: Seqüència de particions i alternatives sense automorfismes

seqüència compatible amb l'original. La diferència amb el cas anterior és que ara detindrà l'exploració dels vèrtexs en el nivell k_4 quan trobi la primera seqüència compatible. És a dir, no calcularà tots els automorfismes possibles. En la figura es representa com, per la a primera branca, la primera seqüència a partir del nivell k_4 falla, però la segona proporciona un èxit i, per tant, totes les crides recursives a l'algorisme subtree_compat retornen la partició final $\mathcal{S}^t$, indicant que s'ha tingut èxit en la generació d'un automorfisme, fins arribar a la crida feta des de la partició $\mathcal{S}^{k_3}$, des de la qual es continuarà l'exploració del vèrtex següent d'aquest nivell sense mirar la resta de vèrtexs del nivell k_4. D'aquesta forma es trobarien 3 automorfismes que fixarien els vèrtexs descartats de la seqüència original des de la primera partició fins a la de nivell k_3.

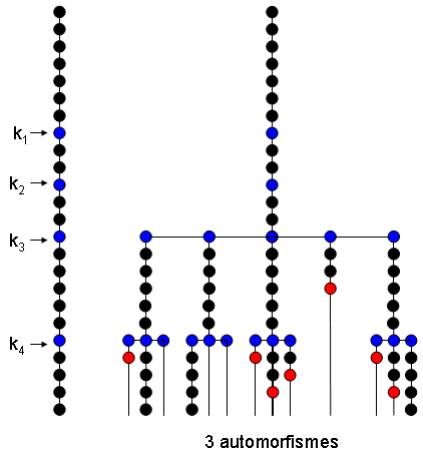


Figura 4.6: Seqüència de particions i alternatives a partir del segon nivell *UNKNOWN*

Un altre cas que es pot donar consisteix en què la partició de tipus *BACKTRACK* $\mathcal{S}^{k_4}$ sigui subpartició de la partició $\mathcal{S}^{k_3}$. Aquesta situació es coneix gràcies al càlcul previ dels nivells límit, L^k, de cada partició $\mathcal{S}^k$; aleshores el teorema de subpartició assegura l'existència d'algun automorfisme i crida l'algorisme *process_subpartition* per trobar-los. La figura 4.7 mostraria aquest cas.

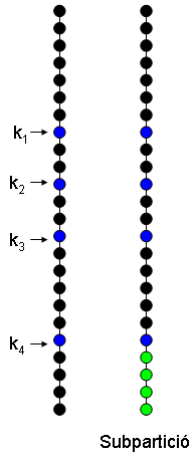


Figura 4.7: Seqüència de particions i alternativa amb subpartició

L'última situació possible, en aquest nivell, consistiria en què, quan s'exploren tots els vèrtexs de la cel·la pivot de la partició k_3, es troben seqüències compatibles amb l'original fins al nivell k_4, però a partir d'aquest nivell no es troba una seqüència compatible completa. Aquesta situació està representanda en la figura 4.8 i, en aquest cas, com que no s'ha trobat cap seqüència compatible amb l'original, no existeix cap automorfisme en aquesta branca de l'arbre.

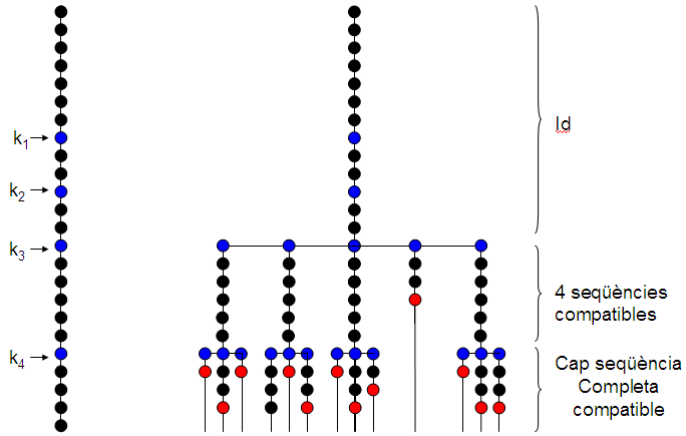


Figura 4.8: Seqüència de particions i alternativa no completa

En el supòsit que ara s'estigui explorant el nivell de tipus *UNKNOWN*, k_2, tal i com es pot veure en la figura 4.9, i que la partició $\mathcal{S}^{k_4}$ sigui subpartió de $\mathcal{S}^{k_3}$es tindrà que el nivell de *BACKTRACK* de la partició $\mathcal{S}^{k_4}$ serà la partició $\mathcal{S}^{k_2}$. En arribar al nivell equitatiu següent, k_3, es comencen a explorar els vèrtexs de la cel·la pivot d'aquest nivell. S'observa com el primer vèrtex de k_3 proporciona una seqüència compatible fins arribar al nivell k_4, on es tornarà a ramificar l'arbre d'exploració amb una branca per cada vèrtex de la cel·la pivot del nivell k_4. Segons la imatge els tres vèrtexs de la cel·la pivot del nivell k_4 fallen a l'hora de proporcionar una seqüència alternativa compatible i, com que $\mathcal{S}^{k_4}$ és subpartició de $\mathcal{S}^{k_3}$, el teorema 4.2.8 assegura que no es trobarà cap seqüència completa compatible des de la partició $\mathcal{S}^{k_3}$. Per tant,

l'algorisme no retornarà al nivel k_3 per seguir explorant la resta de vèrtexs de la cel·la pivot, si no que retornarà el nivell de *BACKTRACK*, $\mathcal{S}^{k_2}$, i, en conseqüència, saltarà directament al nivell k_2 (backjumping) per continuar amb la cerca per l'arbre a partir del vèrtex candidat següent a proporcionar un automorfisme. Segons la figura la resta de vèrtexs de k_3 porten a situacions similars a l'anterior i s'aplica el backjumping. Això pot comportar un gran estalvi de temps de càlcul si les cel·les pivot són d'una mida considerable. En l'exemple s'aprecia com l'algorisme s'estalviaria l'exploració de 6 vèrtexs del nivell k_3.

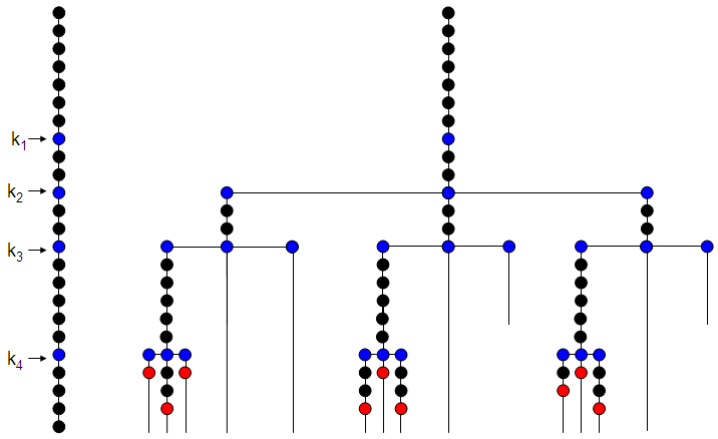


Figura 4.9: Seqüència de particions i alternativa: backjumping

Finalment, la figura 4.10 mostra una situació similar a l'exposada en el cas anterior on $\mathcal{S}^{k_4}$ és subpartició de $\mathcal{S}^{k_3}$. En aquest cas, durant l'exploració del primer i del tercer vèrtexs de k_2 s'aplicaria el teorema 4.2.8, però durant l'exploració del segon vèrtex de k_2 es trobaria una seqüència alternativa compatible gracies al teorema de subpartició.

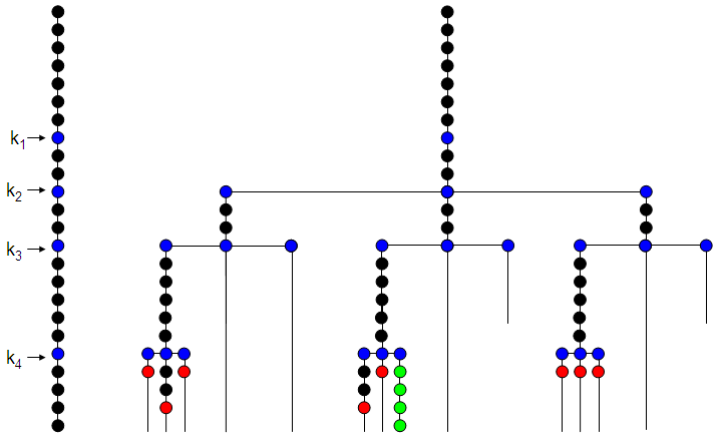


Figura 4.10: Seqüència de particions i alternativa: backjumping i subpartició

4.3 Determinació del grup complet d'automorfismes

Donat un graf G, el programa *Conauto* determina un conjunt d'automorfismes del graf que no és el grup complet d'automorfismes, però que genera el grup. El que no està clar és que el grup de generadors calculat sigui minimal. Per determinar el grup complet d'automorfismes del graf es pot usar un algorisme basat en la força bruta.

algorisme 13 Algorisme que computa tot el grup d'automorfismes d'un graf G a partir del seu conjunt de generadors Γ.

(Aut(G)) GIso::gen_Auto(Γ)

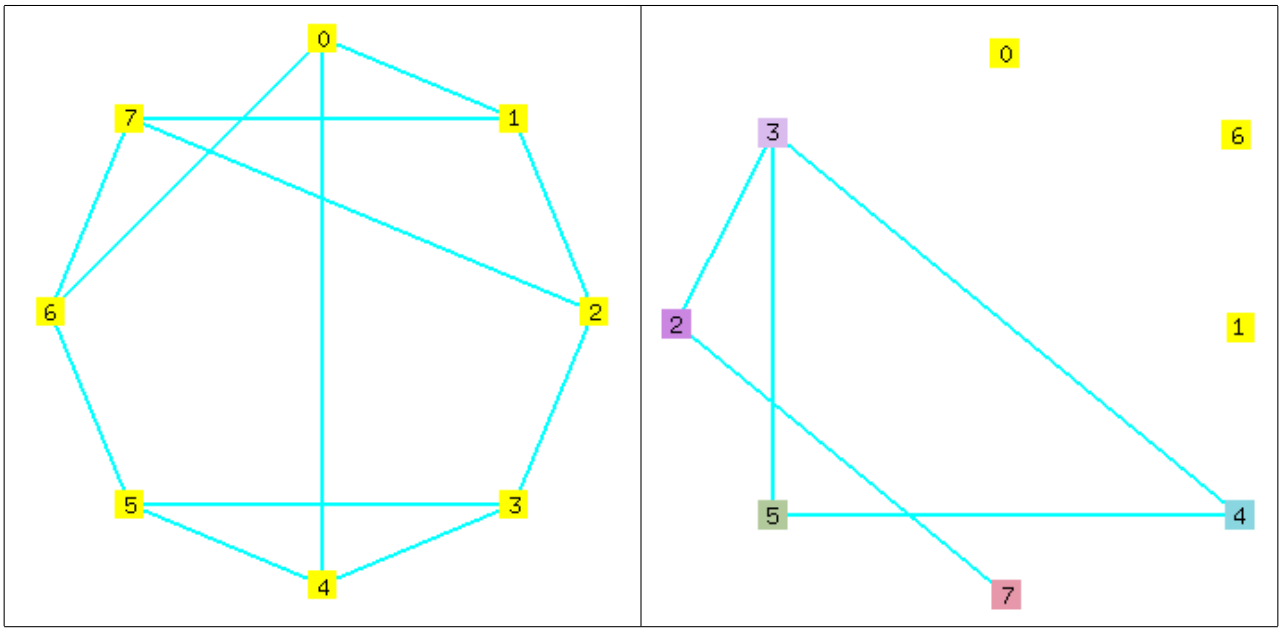
1: $H \leftarrow \emptyset$
2: $H_{new} \leftarrow \{\mathbf{Id}\}$
3: **while** $(H_{new} \neq \emptyset)$ **do**
4: $\quad H \leftarrow H \cup H_{new}$
5: $\quad H_{last} \leftarrow H_{new}$
6: $\quad H_{new} \leftarrow \emptyset$
7: $\quad$ **for all** $(\gamma \in \Gamma)$ **do**
8: $\quad\quad$ **for all** $(h \in H_{last})$ **do**
9: $\quad\quad\quad f \leftarrow \gamma \circ h$
10: $\quad\quad\quad$ **if** $(f \notin H)$ **then**
11: $\quad\quad\quad\quad H_{new} \leftarrow H_{new} \cup \{f\}$
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end for**
14: $\quad$ **end for**
15: **end while**
16: **return** H

L'algorisme 13 genera el grup complet d'automorfismes $Aut(G)$ del graf G a partir del conjunt de generadors Γ. Per fer-ho, calcula totes les composicions dels elements del conjunt Γ i comprova, per a cada composició, si ja pertany al grup. En cas negatiu afegeix l'automorfisme computat al grup $Aut(G)$. Com és obvi, aquest algorisme realitza molts càlculs i és molt ineficient perquè un mateix element del grup pot sorgir moltes vegades en computar diferents composicions dels elements del conjunt de generadors.

4.3.1 Exemple. En el graf LPTA, del qual s'ha mostrat ja la seqüència de particions, l'algorisme troba dos automorfismes. Els automorfismes es guarden en un fitxer com a seqüències ordenades dels vèrtexs. D'aquesta forma, un automorfisme vindrà donat per la posició dels vèrtexs en cada seqüència respecte a la seqüència original o base. En el graf original la seqüència base era: $\{0, 6, 1, 4, 7, 5, 2, 3\}$.

La taula 4.3 mostra els tres automorfismes del graf LPTA. Els dos primers han estat calculats pel programa *Conauto* i el tercer per l'algorisme 13. La primera columna de la taula mostra el número de la seqüència, que s'inicia en 0. En la posició 0 hi ha la seqüència base del graf. Així, el primer automorfisme trobat és el que passa de la seqüència base $\{0, 6, 1, 4, 7, 5, 2, 3\}$ a la primera seqüència $\{0, 6, 4, 1, 5, 7, 3, 2\}$

Així, el primer automorfisme γ_1 passaria la seqüència base 0 a la seqüència 1 i s'escriuria en notació clàssica $\gamma_1 = (0)(6)(14)(57)(23)$ i fixaria els vèrtexs 0 i 6, permutaria els vèrtexs 1 i 4, permutaria el 5 i el 7 i, finalment, permutaria el 2 i el 3.



Taula 4.1: Graf LPTA8 i la seva seqüència base.

0:	0	6	1	4	7	5	2	3
1:	0	6	4	1	5	7	3	2
2:	6	0	5	7	4	1	3	2
3:	6	0	7	5	1	4	2	3

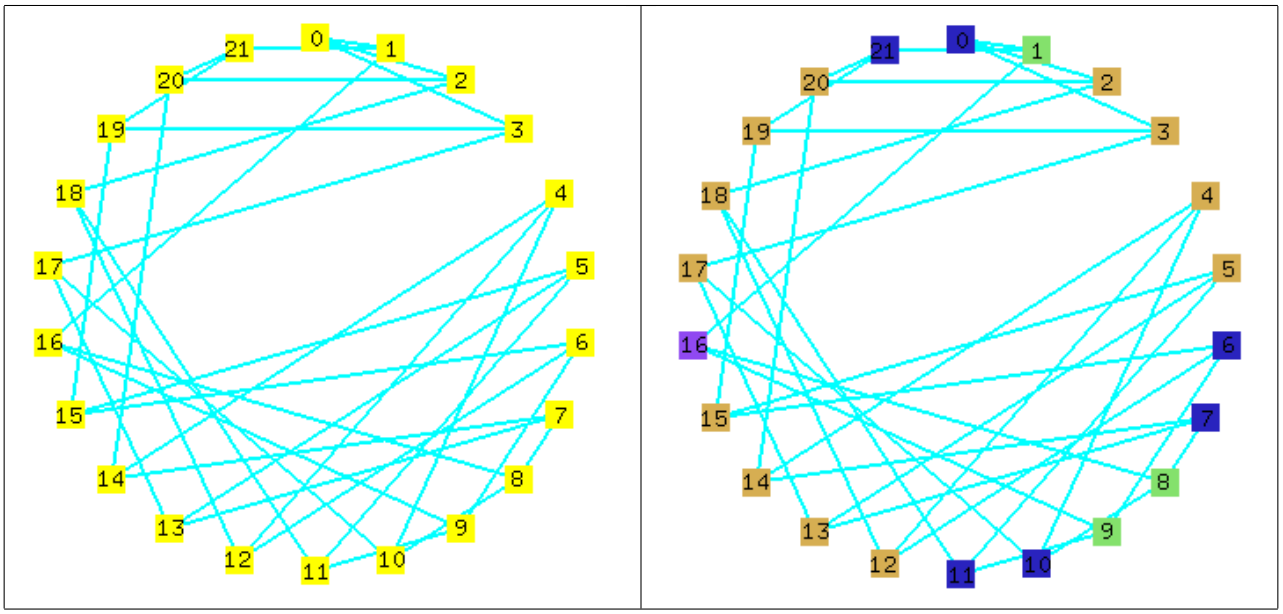
Taula 4.2: Automorfismes del Graf LPTA8.

De la mateixa forma, s'ha computat la taula producte del grup (4.3). Ara els índexs de cada posició de la taula representen l'automorfisme, on γ_0 seria la identitat.

$\circ$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Taula 4.3: Taula producte del grup d'automorfismes del Graf LPTA.

4.3.2 Exemple. Seguint la mateixa notació que en l'exemple anterior es mostra un altre graf a la figura 4.4 junt amb la seva partició per òrbites.



Taula 4.4: Graf CHH_cc_(1_1)_22-1 i les seves òrbites.

El grup d'automorfismes i la taula producte del grup es poden observar a les taules 4.5 i 4.6, respectivament.

0:	0	21	1	16	2	17	12	11	4	18	3	20	19	9	8	13	10	14	15	5	6	7
1:	0	21	1	16	3	18	13	10	5	17	2	19	20	8	9	12	11	15	14	4	7	6
2:	4	13	8	16	10	20	5	18	0	11	14	17	7	9	1	21	2	3	6	12	15	19
3:	4	13	8	16	14	11	21	2	12	20	10	7	17	1	9	5	18	6	3	0	19	15
4:	5	12	9	16	11	19	4	17	0	10	15	18	6	8	1	21	3	2	7	13	14	20
5:	5	12	9	16	15	10	21	3	13	19	11	6	18	1	8	4	17	7	2	0	20	14
6:	12	5	9	16	18	7	0	20	4	2	6	11	15	1	8	13	14	10	19	21	3	17
7:	12	5	9	16	6	2	13	14	21	7	18	15	11	8	1	0	20	19	10	4	17	3
8:	13	4	8	16	17	6	0	19	5	3	7	10	14	1	9	12	15	11	20	21	2	18
9:	13	4	8	16	7	3	12	15	21	6	17	14	10	9	1	0	19	20	11	5	18	2
10:	21	0	1	16	20	15	4	7	12	14	19	2	3	8	9	5	6	18	17	13	10	11
11:	21	0	1	16	19	14	5	6	13	15	20	3	2	9	8	4	7	17	18	12	11	10

Taula 4.5: Automorfismes del Graf CHH_cc_(1_1)_22-1

4.4 Algorisme de Schereir-Sims

L'algorisme de Schereir-Sims permet calcular el grup complet d'automorfismes d'una forma més eficient. Sigui $G = (V, E)$ un graf amb el conjunt de vèrtexs $V = (0, 1, ..., n-1)$ i sigui Aut(G) el seu grup d'automorfismes. Considerem els conjunts següents:

$$G_0 = \{\pi \in Aut(G) : \pi(0) = 0\}$$

$$G_1 = \{\pi \in G_0 : \pi(1) = 1\}$$

∘	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	2	4	3	5	6	8	7	9	10	11
1	1	0	3	5	2	4	7	9	6	8	11	10
2	2	4	0	1	6	8	3	5	10	11	7	9
3	3	5	1	0	7	9	2	4	11	10	6	8
4	4	2	6	8	0	1	10	11	3	5	9	7
5	5	3	7	9	1	0	11	10	2	4	8	6
6	6	8	4	2	10	11	0	1	9	7	3	5
7	7	9	5	3	11	10	1	0	8	6	2	4
8	8	6	10	11	4	2	9	7	0	1	5	3
9	9	7	11	10	5	3	8	6	1	0	4	2
10	10	11	8	6	9	7	4	2	5	3	0	1
11	11	10	9	7	8	6	5	3	4	2	1	0

Taula 4.6: Taula producte del grup del graf del Graf CHH_cc_(1_1)_22-1

$$G_2 = \{\pi \in G_1 : \pi(2) = 2\}$$

$$G_{n-1} = \{\pi \in G_{n-2} : \pi(n-1) = n-1\} = \{\mathbf{Id}\}$$

Aquests conjunts satisfan que $\{\mathbf{Id}\} = \mathcal{G}_{n-1} \subseteq G_{n-2}... \subseteq G_1 \subseteq \mathcal{G}_0 \subseteq Aut(G)$. Així el conjunt $G_i, i \in V$, seria el conjunt format pels automorfismes de G que deixen invariant el conjunt $\{j \in V : j \leq i\}$.

4.4.1 Definició. L'*òrbita* del vèrtex $\{0\}$ sota $Aut(G)$ és $orb(0) = \{\pi(0) : \pi \in Aut(G)\}$. Considerem $n_0 = |orb(0)|, n_0 \leq n$, i notem el conjunt $orb(0) = \{x_{0,1}, x_{0,2}, ..., x_{0,n_0}\}$. Aleshores, per a cada $i, 1 \leq i \leq n_0$, es tria un element $h_{0,i} \in Aut(G)$ tal que $h_{0,i}(0) = x_{0,i}$ i es considera el conjunt $U_0 = \{h_{0,1}, h_{0,2}, ..., h_{0,n_0}\}$.

Per a la resta de vèrtexs, es defineixen les seves òrbites i els conjunts associats d'una forma lleugerament diferent a la del vèrtex $\{0\}$.

4.4.2 Definició. Per a tot $i, 1 \leq i \leq n-1$, l'*òrbita* del vèrtex $\{i\}$ és $orb(i) = \{\pi(i) : \pi \in G_{i-1}\} = \{x_{i,1}, x_{i,2}, ..., x_{i,n_i}\}$ on $n_i = |orb(i)|, n_i \leq n$. Aleshores, per a cada $j, 1 \leq j \leq n_i$, s'escull un element $h_{j,i} \in G_{i-1}$ tal que $h_{j,i}(i) = x_{j,i}$ i es considera el conjunt $U_i = \{h_{i,1}, h_{i,2}, ..., h_{i,n_i}\}$.

4.4.3 Teorema. *Siguin $Aut(G)$, G_0 i U_0 definits anteriorment. Aleshores tot element $\pi \in Aut(G)$ es pot escriure, de forma única, com $\pi = h_{0,i}\pi'$, per a algun $\pi' \in G_0$.*

4.4.4 Teorema. *Siguin $Aut(G)$, G_i i U_i definits anteriorment. Aleshores, $\forall i \in \{1, .., n-1\}$, tot element $\pi \in G_{i-1}$ es pot escriure, de forma única, com $\pi = h_{i,j}\pi'$, per a algun $\pi' \in G_i$.*

4.4.5 Corol·lari. *Tot element $\pi \in Aut(G)$ es pot expressar, de forma única, com*

$$g = h_{0,i_0} h_{1,i_1} \cdots h_{n-1,i_{n-1}}$$

.

L'estructura de dades

$$\vec{U} = \{U_0, U_1, ..., U_{n-1}\}$$

s'anomena *representació de Schreier-Sims* del grup Aut(G).

4.4.6 Observació. Si es poden construir els conjunts $U_i, i \in \{0, ..., n-1\}$ es disposa d'una bona eina per calcular tots els automorfismes d'un graf d'una forma ràpida i eficient mitjançant un senzill procediment de backtracking, tal i com es mostra en els algorismes 14 i 15.

algorisme 14 Algorisme que calcula tots els automorfismes d'un graf a partir de l'estructura de dades de Schreier-Sims.

(Aut(G)) GIso::gen_Auto_back($Aut(G), l, \vec{U}, \pi$)

1: **if** $l = |V|$ **then**
2: $Aut(G) \leftarrow Aut(G) \cup \{\pi\}$
3: **return** $Aut(G)$
4: **else**
5: **for all** $h \in U_l$ **do**
6: $f_l \leftarrow h\pi$
7: $Aut(G) \leftarrow gen\_Auto\_back(Aut(G), l+1, \vec{U}, f_l)$
8: **end for**
9: **end if**

algorisme 15 Algorisme principal de càlcul de tots els automorfismes d'un graf a partir de l'estructura de dades de Schreier-Sims.

void GIso::gen_Auto_main( )

Require: $\vec{U} = \{U_0, U_1, ..., U_{n-1}\}$

1: $Aut(G) \leftarrow gen\_Auto\_back(\emptyset, 0, \vec{U}, \mathbf{I})$
2: **return**

Finalment, es pot obtenir l'estructura de Schreier-Sims per un graf G a partir d'un conjunt de generadors Γ del seu grup d'automorfismes. Aquesta és la situació que es presenta en el programa *Conauto*, que troba un conjunt de generadors donat un graf.

L'algorisme 16 decideix, a partir d'una estructura $\vec{U}$ inicial incompleta, en quin conjunt U_i s'ha de col·locar un automorfisme π que rep com a paràmetre d'entrada. A més, l'algorisme modifica l'automorfisme π de forma adequada per tal que satisfaci les propietats dels elements dels conjunts de $\vec{U}$. En cas que l'automorfisme π no pugui ser inclòs en l'estructura $\vec{U}$, retorna el valor $n = |V|$.

algorisme 16 Algorisme que indica el conjunt de l'estructura de Schreier-Sims on s'ha de col·locar un determinant automorfisme.

int GIso::test($\pi, \vec{U}$)

1: **for** $i \leftarrow 0$ to $|V|$ **do**
2: $x \leftarrow \pi(i)$
3: **if** $\exists h \in U_i : h(i) = x$ **then**
4: $\pi' \leftarrow gh^{-1}$
5: **for** $j \leftarrow 0$ to $n-1$ **do**
6: $\pi(j) \leftarrow \pi'(j)$
7: **end for**
8: **else**
9: **return** i
10: **end if**
11: **end for**
12: **return** $|V|$

algorisme 17 Algorisme recursiu que actualitza l'estructura de Schreier-Sims amb nous automorfismes.

void GIso::enter($\pi, \vec{U}$)

1: $i \leftarrow test(\pi, \vec{U})$
2: **if** $i = |V|$ **then**
3: **return**
4: **else**
5: $U_i \leftarrow U_i \cup \pi$
6: **for** $j \leftarrow i$ to $n-1$ **do**
7: **for all** $h \in U_j$ **do**
8: $f \leftarrow h\pi$
9: enter($f, \vec{U}$)
10: **end for**
11: **end for**
12: **end if**

L'algorisme 17 utilitza l'algorisme 16 per generar tots les elements $h_{i,j}$ possibles, no coneguts encara, a partir d'un automorfisme conegut π i d'una estructura $\vec{U}$ inicial incompleta, de forma recursiva, i actualitza l'ementada estructura quan troba nous elements. Si no es poden trobar elements nous de l'estructura de dades l'algorisme retorna.

Finalment, l'algorisme principal 18 parteix d'una estructura de dades formada per conjunts iguals a la identitat i d'un conjunt de generadors, Γ, del grup d'automorfismes del graf per generar tota l'estructura de dades $\vec{U}$ usant els algorismes anteriors.

algorisme 18 Algorisme que genera l'estructura de Schreier-Sims a partir d'un conjunt de generadors.

void GIso::main_enter()

1: **for** $i \leftarrow 0$ to $n-1$ **do**
2: $\quad U_i \leftarrow \{\mathbf{I}\}$
3: **end for**
4: **for all** $\gamma \in \Gamma$ **do**
5: $\quad$ enter($\gamma, \vec{U}$)
6: **end for**

Capítol 5

Determinació d'isomorfismes

Per determinar si dos grafs G i H són isomorfs basta trobar dues seqüències de particions completes Q_G i Q_H que siguin compatibles. L'algorisme 19 és l'encarregat de, donats dos grafs G i H, determinar si són isomorfs retornant un valor booleà: *TRUE* si són isomorfs o *FALSE* si no ho són. Per fer-ho, primer comprova una sèrie d'invariants trivials que permeten determinar fàcilment la impossibilitat de trobar un isomorfisme: comprova que el nombre de vèrtexs i d'enllaços dels dos grafs coincideixin i, en cas que no sigui així, retorna *FALSE*. La comprovació següent es basa en la partició per graus, determina la partició per graus de cada graf i comprova la seva compatibilitat retornant, *FALSE* en cas d'incompatibilitat entre les particions. En cas d'haver superat aquestes proves, es procedeix a calcular la seqüència de particions del graf G i se cerquen els generadors del seu grup d'automorfismes. Si en la seqüència de particions no han aparegut nivells de tipus *BACKTRACK*, es crida directament la funció *match*, algorisme 20, perquè cerqui un isomorfisme a partir de la partició inicial, buscant en el graf H una seqüència de particions que sigui compatible amb la trobada pel graf G; si troba aquesta seqüència, l'algorisme retorna TRUE i, en cas contrari, retorna FALSE. Quan en la seqüència del graf G hi ha nivells de tipus *BACKTRACK* se cerca la seqüència de particions i el grup de generadors del graf H i es computa el seu nombre de nivells de tipus *BACKTRACK*. Abans de cercar un isomorfisme, es comprova un altre invariant, calcula el nombre d'automorfismes del graf G i el del graf H i comprova que siguin iguals; en cas d'igualtat, es crida la funció *match* passant-li com a guia o base de cerca la seqüència del graf G o H que tingui un menor nombre de nivells de tipus *BACKTRACK*, d'aquesta manera es minimitza la feina de càlcul. La funció *match* tractarà de trobar una seqüència de particions alternativa en el graf amb major nombre de nivells de tipus *BACKTRACK* compatible amb la seqüència original de l'altre graf; si la troba haurà determinat que els dos grafs són isomorfs i haurà trobat un isomorfisme entre ells, retornant l'última partició, $\mathcal{S}^t$, de manera que l'algorisme 19 retorna TRUE. En cas de no trobar una seqüència compatible, la funció *match* retorna la primera partició, $\mathcal{S}^0$, i es pot afirmar que els dos grafs no són isomorfs, de manera que l'algorisme 19 retorna FALSE.

L'algorisme 20 és un algorisme recursiu, molt semblant a la funció subtree_compat, que fa la cerca d'una seqüència alternativa completa (fins a l'últim nivell t) en el graf H a partir de la partició $\mathcal{T}$, que es troba en el nivell l, que és compatible amb la partició $\mathcal{S}^l$ del graf G. La diferència amb l'algorisme 10 radica en què la funció *match* es comença a cridar des de la partició 0 i que la funció acaba en el moment en què troba una seqüència completa compatible, mentre que la funció subtree_compat cercava diverses seqüències completes compatibles per trobar un conjunt de generadors d'automorfismes i començava la cerca des de l'ultima partició

algorisme 19 Funció que determina si dos grafs són isomorfs.

(bool) GIso::areIsomorphics(G, H)

1: **if** ($|V_G| \neq |V_H|$ || $|E_G| \neq |E_H|$) **then**
2: **return false**
3: **end if**
4: $\mathcal{S}^0 \leftarrow degreePartition(G)$
5: $\mathcal{T}^0 \leftarrow degreePartition(H)$
6: **if** ($\mathcal{S}^0$ and $\mathcal{T}^0$ are not compatible) **then**
7: **return false**
8: **end if**
9: $Q_G \leftarrow G.sequence()$
10: $E_G \leftarrow search\_automorphims(\mathcal{S}^0)$
11: **if** ($num\_backtrack\_levels(E_G) = 0$) **then**
12: **return** (match$(0, G, H, E_G, \mathcal{T}^0, \{\Gamma_H\}\ ) \neq \mathcal{S}^0$)
13: **end if**
14: $Q_H \leftarrow H.sequence()$
15: $E_H \leftarrow search\_automorphims(\mathcal{T}^0)$
16: **if** ($|Aut(G)| \neq |Aut(H)|$ || $|OrbitsOf(E_G)| \neq |OrbitsOf(E_G)|$) **then**
17: **return false**
18: **end if**
19: **if** ($num\_backtrack\_levels(E_G) \leq num\_backtrack\_levels(E_H)$) **then**
20: **return** (match$(0, G, H, E_G, \mathcal{T}^0, \{\Gamma_H\}\ ) \neq \mathcal{S}^0$)
21: **else**
22: **return** (match$(0, H, G, E_H, \mathcal{S}^0, \{\Gamma_G\}\ ) \neq \mathcal{T}^0$)
23: **end if**

de tipus *BACKTRACK*. L'algorisme 20 fa un refinament de la partició $\mathcal{T}$, del mateix tipus que el que s'ha fet en la partició $\mathcal{S}^l$ de la seqüència original, amb la corresponent cel·la pivot $T_{\mathcal{P}^l}$, i obté una nova partició $\mathcal{T}'$, pel graf H, a nivell $l+1$. Aleshores comprova si les particions $\mathcal{S}^{l+1}$ i $\mathcal{T}'$ són compatibles entre si i, en cas afirmatiu, es crida recursivament a si mateix per trobar un refinament, a partir de la partició $\mathcal{T}'$, en el nivell $l+1$.

El funcionament és el següent:

- Si l'algorisme rep com argument l'última partició (nivell t), comprova que aquesta partició, $\mathcal{T}$, del graf H i la partició S^t, del graf G, siguin compatibles i, si és així, retorna la partició del nivell t, S^t (ha trobat una seqüència compatible).

- Si encara no s'ha arribat a l'última partició ($l < t$) i $\mathcal{R}^l = VERTEX$, fa un refinament de la partició rebuda, $\mathcal{T}$, per vèrtex, obtenint una nova partició, $\mathcal{T}'$, i comprova la seva compatibilitat amb la partició original, $\mathcal{S}^{l+1}$, mitjançant la funció *vertex_ref_compat*. Si les particions són compatibles fa una crida recursiva a la funció *match* passant-li com arguments el nou nivell $l+1$, la nova particó $\mathcal{T}'$ i el nou conjunt de generadors del graf H obtingut, Δ'. En la crida recursiva l'algorisme retorna una partició, $\mathcal{T}''$, que pot ser posterior a la partició original de crida, $\mathcal{S}^l$, aleshores aquesta partició ha de ser l'última i hem trobat un isomorfisme; en canvi, si retorna la mateixa partició, $\mathcal{T}'$, o una partició anterior, vol dir que no ha pogut trobar una seqüència compatible per aquella branca

de l'arbre de cerca i s'ha de fer backtracking. La partició retornada en aquesta crida recursiva, $\mathcal{T}''$, pot ser igual a la partició des de la qual s'ha fet la crida a la funció (figura 5.1), $\mathcal{T}'$, i, per tant, la funció retorna el nivell de backtrack d'aquesta partició, $\mathcal{S}^{B^l}$, o una partició anterior (5.2) i, per tant, es podrà fer backjumping des d'aquesta partició perquè la funció retornarà directament aquesta partició, T''. En l'exemple de la figura 5.2 la partició retornada seria $\mathcal{S}^{k_1}$, ja que seria el nivell de backtracking de la partició $\mathcal{S}^{k_3}$, on s'ha trobat la incompatibilitat. Cal observar que el nivell l és posterior al nivell k_3 ($l < k_3$). Finalment, en cas que la partició refinada, $\mathcal{T}'$, i la partició original, $\mathcal{S}^{l+1}$, no siguin compatibles, es retorna la partició del nivell de bactracking S^{B^l}, per tornar a la partició anterior.

– Si $\mathcal{R}^l =$ "SET", l'algorisme actua igual que en el cas anterior, exceptuant que el refinament es fa per conjunt mitjançant l'algorisme *set_ref_compat*.

– Quan $\mathcal{R}^l =$ "$BACKTRACK$", l'algorisme disposa de diverses possibilitats per trobar un isomorfisme. S'haurien de provar refinaments per a tots els vèrtexs de la cel·la pivot fins trobar-ne un que proporcioni un isomorfisme o descartar-los tots. Per tal de podar l'arbre de cerca, es computen les òrbites parcials del graf H des del nivell 0 fins al nivell l, d'aquesta manera només es farà un refinament per un vèrtex de cada òrbita, ja que els vèrtexs d'una mateixa òrbita portarien a resultats similars. Per això, es determinen les òrbites dels vèrtexs vàlids que pertanyen a la cel·la pivot $T_{\mathcal{P}^l}$ i, per aquests vèrtexs vàlids, es realitza un refinament per vèrtex, cridant la funció *vertex_ref_compat*, sempre i quan el vèrtex pivot que s'utilitzarà per fer el refinament pertanyi a una òrbita que tingui la mateixa mida que l'òrbita del vèrtex $p \in S^l_{\mathcal{P}^l}$, del graf G, usat per fer el refinament, en aquest nivell, en la seqüència original (això també permet reduir l'espai de cerca). El refinament per vèrtex realitzat proporciona una nova partició, $\mathcal{T}'$; si aquesta partició no és compatible amb la partició S^{l+1}, aleshores es descarta i es passa a provar un nou refinament amb el vèrtex vàlid següent de la cel·la pivot. Quan la partició refinada, $\mathcal{T}'$, és compatible amb la refinada original, $\mathcal{S}^{l+1}$, es procedeix a fer una crida recursiva a l'algorisme *match*. Igual que en els casos anteriors, la crida recursiva retorna una partició. Si la partició retornada és la mateixa que l'original des de la qual s'ha invocat la funció, aleshores (no s'ha trobat una seqüència compatible) es prova el vèrtex següent per fer un nou refinament; si la partició retornada és posterior, vol dir que és l'última, i hem trobat un isomorfisme. Finalment, si la partició retornada és anterior, només pot ser degut a que s'ha arribat a una partició equitativa, que és subpartició de $\mathcal{T}'$, i no ha trobat cap seqüència compatible a partir d'ella; aleshores ha retornat el seu nivell de backtracking (la primera partició equitativa anterior de la qual no n'és subpartició) i, per tant, es fa backjumping, aplicant el teorema 4.2.8, fins aquesta partició.

Si un cop provats tots els vèrtexs vàlids de la cel·la pivot no hem trobat cap compatibilitat, aleshores es retorna la partició de bactracking $\mathcal{S}^{B^l}$, que és la partició on al menys hi ha dos cel·les de la partició actual que encara no han estat diferenciades.

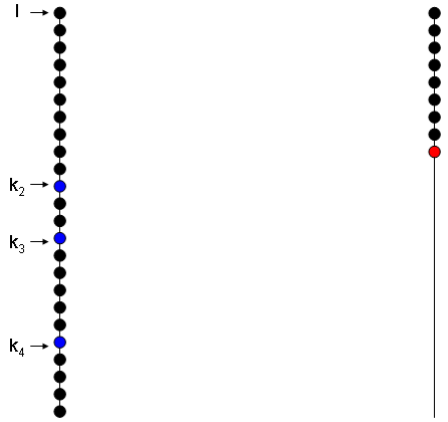


Figura 5.1: Seqüència de particions i alternativa: back

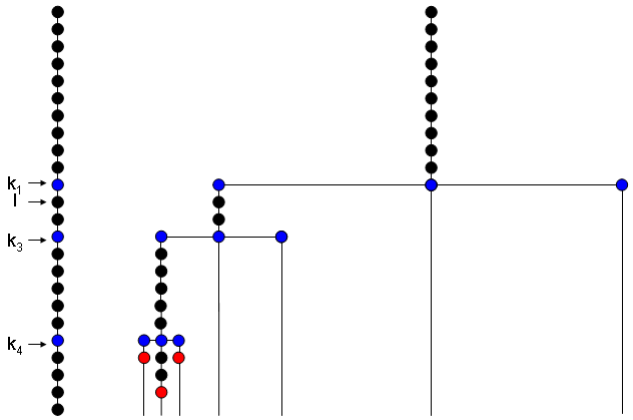


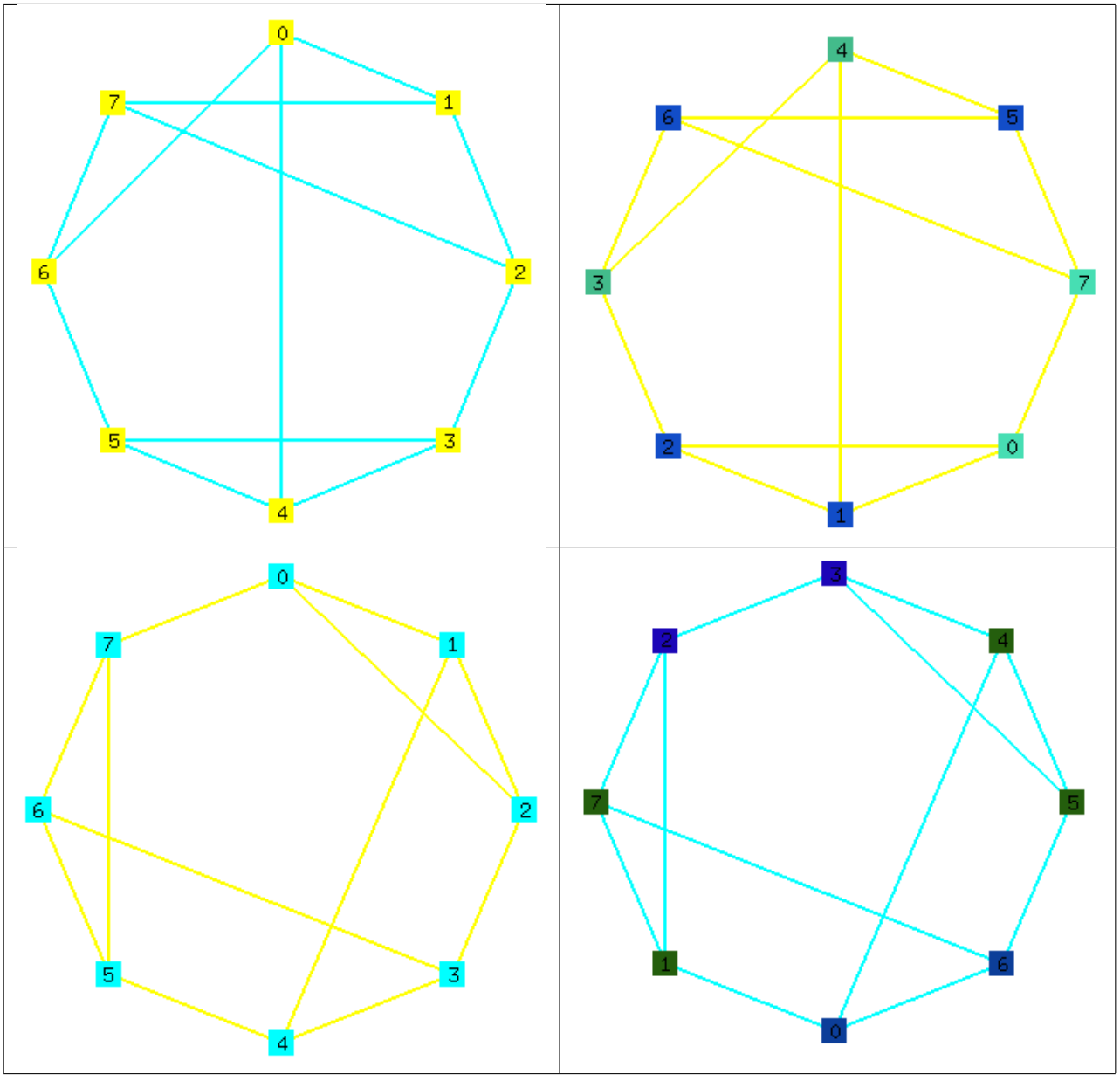
Figura 5.2: Seqüència de particions i alternativa: backjumping

algorisme 20 Funció que cerca dues seqüències de particions completes compatibles.

Partition GIso::match$(l, G, H, E_G, \mathcal{T}, \Delta)$

1: **if** $(l = t)$ **then**
2: **if** $(\forall i, j \in \{1, ..., r_l\},\ ADeg(S_i, S_j, G) = ADeg(T_i, T_j, H))$ **then**
3: **return** $\mathcal{S}^t$
4: **end if**
5: **else**
6: **switch** ($\mathcal{R}^l$)
7: **case** *"VERTEX"*:
8: **if** $(vertex\_ref\_compat(H, \mathcal{S}^l, \mathcal{T}))$ **then**
9: $\mathcal{T}'' \leftarrow match(l+1, G, H, E_G, \mathcal{T}', \Delta')$
10: **if** ($\mathcal{T}'' \neq S^l$) **then**
11: **return** $\mathcal{T}''$
12: **end if**
13: **end if**
14: **break**
15: **case** *"SET"*:
16: **if** ($set\_ref\_compat(H, \mathcal{S}^l, \mathcal{T})$) **then**
17: $\mathcal{T}'' \leftarrow match(l+1, G, H, E_G, \mathcal{T}', \Delta')$
18: **if** ($\mathcal{T}'' \neq S^l$) **then**
19: **return** $\mathcal{T}''$
20: **end if**
21: **end if**
22: **break**

5.0.7 Exemple. En aquest exemple es mostra la part gràfica on es pot observar l'isomorfisme entre dos grafs. La taula 5.1 mostra, a la primera columna, els grafs LPTA i LPTB, respectivament. A la segona columna es visualitza, al costat de cada graf, l'altre graf, representat de forma que es faci patent l'isomorfisme.



Taula 5.1: Isomorfisme entre els grafs LPTA i LPTB.

Així, si s'observa la primera fila s'aprecia, a la primera imatge, el graf LPTA i, a la segona imatge, el graf LPTB representat de forma que es puguin comparar les arestes i vèrtexs amb els de l'altre graf i es vegi l'isomorfisme. Aquest isomorfisme aplicaria el conjunt de vèrtexs del graf LPTA, $\{0, 1, 2, 3, 4, 5, 6, 7\}$, en el conjunt de vèrtexs del graf LPTB, $\{4, 5, 7, 0, 1, 2, 3, 6\}$, és a dir, el vèrtex 0 del graf LPTA l'aplicaria en el vèrtex 4 del graf LPTB, el vèrtex 1 del graf LPTA l'aplicaria en el vèrtex 5 del graf LPTB, i així successivament. També es poden observar, en la representació del graf LPTB, les òrbites dels vèrtexs distingides per diferents colors; les

òrbites serien $\{0,7\}$, $\{1,2\}$, $\{3,4\}$ i $\{5,6\}$.

A la segona fila es veu un isomorfisme entre el graf LPTB i el graf LPTA; la primera imatge representa el graf LPTB original i la segona representa el graf LPTA, dibuixat de forma que es pugui observar fàcilment l'isomorfisme existent. En aquest cas, l'isomorfisme aplica el conjunt de vèrtexs del graf LPTB, $\{0,1,2,3,4,5,6,7\}$, en el conjunt de vèrtexs del graf LPTA, $\{3,4,5,6,0,1,7,2\}$, seguint l'ordre en què estan escrits els vèrtexs. Les òrbites del graf LPTA s'observen en dirferents colors i són $\{0,6\}$, $\{1,7\}$, $\{2,3\}$ i $\{4,5\}$.

Es pot apreciar com els isomorfismes apliquen vèrtexs d'una òrbita del graf LPTA en vèrtexs de la corresponent òrbita del graf LPTB.

D'altra banda, la taula 5.2 mostra tots els isomorfismes existents entre els grafs LPTA i LPTB, que han estat calculats, pel programa adaptat, via composició de l'isomorfisme trobat pel programa *Conauto* amb tots els automorfismes del graf LPTA trobats prèviament. D'aquesta forma s'obtenen els 4 isomorfismes entre aquests dos grafs.

Vèrtexs graf LPTA - vèrtexs graf LPTB								
1 :	0-3	1-2	2-0	3-7	4-6	5-5	6-4	7-1
2 :	0-4	1-1	2-0	3-7	4-5	5-6	6-3	7-2
3 :	0-3	1-6	2-7	3-0	4-2	5-1	6-4	7-5
4 :	0-4	1-5	2-7	3-0	4-1	5-2	6-3	7-6

Taula 5.2: Taula d'isomorfismes entre els grafs LPTA i LPTB.

Capítol 6

Conclusions

El primer dels objectius d'aquest treball de comprendre, analitzar i fer una exposició més aclaridora els conceptes teòrics i dels algorismes que cimenten el programa *Conauto*, s'ha portat a terme al llarg de tot aquest document.

Pràcticament tots els aspectes involucrats en l'algorisme s'han clarificat, a excepció dels conceptes relacionats amb les fallades durant la cerca d'automorfismes. A [4] es fa referència a un invariant per automorfismes relacionat amb les fallades durant la cerca en l'arbre de seqüències del graf, però el fet és que en aquest treball de final de carrera no està definit el concepte de fallada ni s'ha trobat cap teorema ni demostració que el relacioni amb un invariant. Malgrat tot, el concepte s'utilitza en la programació del programa *Conauto* i sembla que funciona correctament.

S'ha implementat una part gràfica utilitzant la llibreria *OpenGl* amb *glut* que ha permès visualitzar els grafs i les seves seqüències de particions, passant d'una partició a la següent o a l'anterior per poder veure com s'agrupen els vèrtexs en les diferents cel.les. També ha permès comprovar de forma gràfica les diferents òrbites dels vèrtexs i comparar, en paral.lel, dos grafs i observar directament en pantalla l'isomorfisme trobat entre ells.

Finalment, s'ha incorporat a l'algorisme el codi pertinent per determinar tots els automorfismes del graf a partir del conjunt de generadors trobat inicialment. S'han implementat les dues versions, tant la de força bruta com la basada en l'algorisme de Schreier-Sims. L'algorisme calcula i grava en un fitxer de sortida tots els automorfismes del graf a partir dels trobats pel programa *Conauto* i també calcula la taula producte del grup. Referent a aquest últim punt, una possible millora de l'algorisme consistiria en computar la *representació de Schreier-Sims*, necessària per la determinació de tot el grup d'automorfismes, durant el procés de generació de la seqüència de particions i/o el càlcul del conjunt de generadors. No s'ha analitzat aquesta possibilitat amb profunditat i és possible que no sigui factible; un dels principals inconvenients que s'hauria de superar radica en la diferència en la definició de les òrbites usades en el programa *Conauto* i les que fa servir l'algorisme de Schreier-Sims.

Es considera de gran importància la realització d'aquest treball fi de grau que ha permès posar en valor diverses competències apreses durant els estudis. D'una banda, la comprensió de noves realitats, l'abstracció de les mateixes, la formalització matemàtica i algorísmica, la comunicació dels conceptes implicats i dels resultats obtinguts que han donat lloc a la realització d'aquesta memòria. D'altra banda, la posada en funcionament de tot el bagatge previ i de l'aprenentatge fet al llarg del treball han permès anar més enllà i poder continuar la implementació del programa amb diverses eines d'interès.

Bibliografia

[1] Donald L. Kreher and Douglas R. Stinson. *Combinatorial algorithms: generation, enumeration and search.* The CRC Press Series on Discrete Mathematics and its Applications. CRC Press LLC, Boca Raton, Florida, 1998.

[2] José Luis López Presa and Antonio Fernández Anta. *Fast algorithm for graph isomorphism testing.* In Jan Vahrenhold, editor, SEA, volume 5526 of Lecture Notes in Computer Science, pages 221-232. Springer, 2009.

[3] José Luis López Presa. *Efficient Algorithms for Graph Isomorphism Testing.* Doctoral thesis, Escuela Técnica Superior de Ingeniería de Telecomunicación, Universidad Rey Juan Carlos, Madrid, Spain, March 2009. Available at http://www.diatel.upm.es/jllopez/tesis/thesis.pdf.

[4] Luis Felipe Núnez Chiroque. *Use of Automorphisms in Conauto-2.0.* Proyecto final de carrera, Escuela Técnica Superior de Ingeniería de Telecomunicación, Universidad Rey Juan Carlos, Madrid, Spain, October 2011.

[5] Mark Goldberg. *The graph isomorphism problem.* In Jonathan L. Gross and Jay Yellen, editors, Handbook of graph theory, Discrete Mathematics and its Applications, chapter 2.2, pages 68-78. CRC Press, 2003.

[6] José Luis López Presa, Luis Felipe Núnez Chiroque. The Conauto page. 2013. https://sites.google.com/site/giconauto/.

[7] Brendan D. McKay. The nauty page. Computer Science Department, Australian National University, 2013. http://cs.anu.edu.au/~bdm/nauty/.