Escola Politècnica Superior
d'Enginyeria de Vilanova i la Geltrú
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL: Estructures de grafs amb equivalències d'arestes aplicades a l'anàlisi de dades relacionals

AUTORS: Rodríguez Navas, Laura

DATA DE PRESENTACIÓ: Maig, 2017

COGNOMS: Rodríguez Navas **NOM:** Laura

TITULACIÓ: Grau en Enginyeria Informàtica

PLA: 2010

DIRECTOR: José Luis Balcázar Navarro

DEPARTAMENT: Ciències de la Computació

QUALIFICACIÓ DEL TFG

TRIBUNAL

PRESIDENT	SECRETARI	VOCAL
J. M. Merenciano Saladrigues	Bernardino Casas Fernández	Sergio Sánchez Lopez

DATA DE LECTURA: 31 de maig de 2017

Aquest Projecte té en compte aspectes mediambientals: □ Sí □ No

RESUM

En aquest treball es vol contribuir a la investigació de les 2-estructures mitjançant l'aportació d'una eina que faciliti el desenvolupament de treballs futurs que es fonamentin en la utilització de la teoria de les 2-estructures per a l'anàlisi de dades relacionals.

La teoria de les 2-estructures proporciona una infraestructura matemàtica per a la descomposició dels grafs. Permet representar múltiples grafs en una sola estructura algebraica, una 2-estructura, que es divideix en una descomposició única de 2-estructures més simples.

Durant el treball, s'ha dut a terme un estudi amb dues finalitats:

- El disseny i la implementació d'un paquet de software que analitzi i representi visualment les principals estructures involucrades en la teoria de les 2-estructures.

- La investigació i el desenvolupament de possibles aplicacions de les 2-estructures en l'anàlisi de dades relacionals que formen part de les bases de dades relacionals.

Paraules clau (màxim 10):

Python	Grafs	Ciències de la Computació	Algorítmica
NetworkX	2-estructures	Anàlisi de dades	PyDot
Graphviz	SQLite		

ABSTRACT

This project wants to contribute in the research of the 2-structures by providing a tool that facilitate the development of future works in the utilization of the theory of 2-structures for the relational data analysis.

The theory of 2-structures provides a mathematical infrastructure for the decomposition of graphs. Allows multiple graphs for the representation of a single algebraic structure, a 2-structure, that divides in a unique decomposition of 2-structures simpler.

During the project, has carried out a study with two purposes:

- The design and the implementation of a software package to analyse and visualize the main structures representations involved in the theory of 2-structures.

- The research and the development to the possible applications of the 2-structures in the analysis of relational data that form part of the relational databases.

Keywords (10 maximum):

Python	Graphs	Computer Science	Algorithmic
NetworkX	2-structures	Data Analysis	PyDot
Graphviz	SQLite		

AGRAÏMENTS

Aquest treball representa el final d'una segona etapa de la meva vida. Que va començar per casualitat quan vaig decidir deixar l'esport d'elit. Durant aquesta segona etapa he tingut la gran sort de conviure amb un grup de companys excepcionals, a l'EPSEVG i també a la FIB. La seva companyia ha fet el camí a recórrer més fàcil i divertit.

No intentaré anomenar-los a tots, ja que són molts i no me'n vull deixar cap. Confio que ells ja saben qui són i els hi dono mil gràcies per la seva fantàstica companyia durant aquests anys. Tot i així, estic segura que comprendran que faci una excepció i mencioni de manera especial a la meva gran amiga, i companya de pràctiques durant gairebé tota la carrera, Noemí. Durant el temps que vaig passar a la FIB com el que he passat a l'EPSEVG ha estat un pilar tan en l'àmbit acadèmic com en el personal.

Per motius semblants anomenats anteriorment tampoc faré una llistat de professors, dels que en gran part guardo un bon record y amb els que he après molt. No obstant, he d'agrair la confiança, la llibertat, la motivació, la inspiració i el recolzament rebut per en Jose Luís Balcázar alhora de desenvolupar aquest projecte. També m'agradaria agrair, en Bernardino Casas per ensenyar-me a programar i a la Neus Català per iniciar-me en el món de l'anàlisi de dades; aptituds bàsiques per a aquest projecte.

Per últim, el més important. No tinc paraules per expressar el meu agraïment per a tot el que els hi dec als meus pares, Lluís i Montse, que han estat al meu costat en els bons moments però també en els dolents. Sense ells, l'imminent enginyera informàtica que ha escrit aquestes línies no seria res.

A tots ells els hi dedico aquest treball.

ÍNDEX

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

PART I
INTRODUCCIÓ I PLANIFICACIÓ

1. INTRODUCCIÓ

En aquest capítol es descriuen els elements fonamentals en el que s'emmarca el projecte com poden ser l'abast, els objectius, l'estat de l'art, la metodologia emprada i l'avaluació tecnològica. A més s'indica de forma abreviada el contingut i la distribució de la resta dels capítols de la memòria.

1.1 ABAST I OBJECTIUS

Degut a que el temps disponible pel desenvolupament del treball és limitat, l'abast del treball és el d'analitzar i trobar la millor solució per a la representació visual de les 2-estructures que faciliti la comprensió en l'anàlisi de dades relacionals. A més, també incorpora la investigació de possibles noves funcionalitats que es podrien utilitzar en l'anàlisi de dades relacionals a partir de les 2-estructures.

Per tant, segons l'abast, podem agrupar i resumir els objectius en la consecució de dues fites. La primera fita del treball consisteix en el disseny i la implementació d'un paquet de software que compleix els requisits següents:

- Permet la creació i la consulta de propietats de la estructura principal que cobreix l'estudi fonamental sobre la teoria de les 2-estructures.
- Dibuixa gràficament la representació de la estructura principal anomenada anteriorment.
- Integra els components necessaris en un programa interactiu extern que permet la visualització i l'emmagatzematge d'aquesta estructura principal.

La segona fita té com a propòsit la investigació i el desenvolupament de possibles aplicacions de les 2-estructures en l'anàlisi de les dades relacionals.

Per aquest motiu es pren com a punt de partida el següent objectiu:

- Establiment de teoremes i/o resultats que permetin enfortir l'anàlisi de dades relacionals mitjançant l'ús de les 2-estructures.

Un altre dels objectius del treball inclou la utilització d'una aplicació gràfica.

Aquesta aplicació no està integrada en l'entorn del treball, és a dir, que s'hi accedeix a partir d'una crida al programari interactiu extern corresponent. De manera que l'usuari ha de sortir de l'entorn de treball per a realitzar les visualitzacions gràfiques de les 2-estructures des de l'aplicació externa.

Un dels objectius principals del projecte, tot i que l'usuari no hi interactua de manera directa, es la implementació d'un algoritme que dibuixa el model de visualització gràfica d'una 2-estructura. Es tracta d'un model basat en el llenguatge DOT [1]. Aquest llenguatge proporciona una manera simple de descriure gràfiques i les fa més entenedores per a l'usuari final.

Tots els models basats en el llenguatge DOT d'aquest treball s'han creat des de zero i l'algoritme encarregat de generar aquests models crearà la disposició dels elements adequada que facilita a l'usuari la tasca d'entendre els models resultants molt ràpidament.

1.2 ESTAT DE L'ART

1.2.1 Teoria de conjunts

La teoria de conjunts és la branca de les matemàtiques que estudia els conjunts.

Un conjunt és una col·lecció d'objectes ben definits. Exemples de conjunts:

- El conjunt dels nombres naturals és N = {1, 2, 3, . . .}.
- El conjunt dels nombres enters és Z = {. . . , −3, −2, −1, 0, 1, 2, 3, . . .}.
- El conjunt dels nombres racionals és Q = {m/n : m ∈ Z, n ∈ Z, n ≠ 0}.

El conjunt de nombres parells naturals és un subconjunt del conjunt de tots els nombres naturals.

Si A i B són conjunts, llavors:

A és un subconjunt de B (A ⊆ B) si cada element de A és també un element de B. Cada conjunt A és un subconjunt de si mateix, és a dir, A ⊆ A. El conjunt buit és un subconjunt de cada conjunt A, 0 ⊆ A. La relació binària entre els conjunts A i B és un subconjunt A x B.

a) La seva unió es defineix com $A \cup B = \{x: x \in A \text{ o } x \in B\}$.
b) La seva intersecció es defineix com $A \cap B = \{x: x \in A \text{ i } x \in B\}$.
c) La seva diferència es defineix com $A \setminus B = \{x: x \in A \text{ i } x \notin B\}$.
d) El producte cartesià entre A i B es defineix com $A \times B = \{(x, y): x \in A \text{ i } y \in B\}$.
e) El complement de A es defineix com $A^c = \{x: x \notin A\}$.

La relació binària $R \subseteq (A \times A)$ és una relació d'equivalència que sempre compleix les tres condicions següents:

a) R és reflexiva si per a cada element de $a \in A$ tenim que $(a, a) \in R$.
b) R és simètrica: per a cada a i b en A, si $(a, b) \in R$, llavors $(b, a) \in R$.
c) R és transitiva: per a cada a, b i c en A, si $(a, b) \in R$ i $(b, c) \in R$, llavors $(a, c) \in R$.

1.2.2 Teoria de grafs

El precursor de la teoria de grafs va ser Leonhard Euler, que la va iniciar tot intentant resoldre el problema dels set ponts de Königsberg (Euler, 1736) [2].

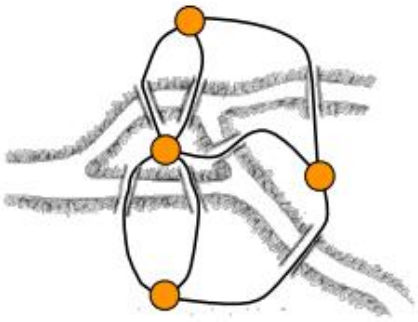


Figura 1. Problema dels set ponts de Königsberg.

La teoria de grafs és una branca de les matemàtiques i la informàtica que es dedica a l'estudi dels grafs, estructures matemàtiques utilitzades per a modelitzar relacions entre parelles d'objectes.

La teoria de grafs es troba estretament relacionada amb la teoria de les 2-estructures. Concretament, una 2-estructura es pot considerar un graf dirigit complet amb les arestes acolorides.

1.2.2.1 Conceptes bàsics

Els grafs s'utilitzen per a representar relacions (simples o d'ordre) entre objectes del mateix tipus. Els objectes reben el nom de nodes o vèrtexs i les relacions entre ells s'anomenen arestes.

Un graf G es pot definir com la parella G = (V, E) on V és un conjunt d'elements anomenats vèrtexs i E és un conjunt d'elements anomenats arestes. Cada element d'E és una parella de vèrtexs diferent.

Existeixen els grafs dirigits i els grafs no dirigits depenen de si les arestes estan orientades o no.

Exemple 1. A la figura 2 veiem una representació gràfica d'un graf no dirigit. En un graf d'aquest tipus les relacions o arestes són simètriques: E (u, v) = E (v, u).

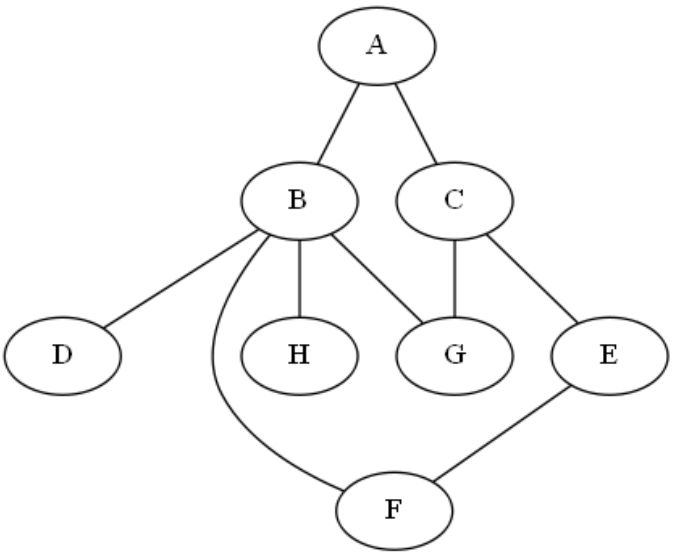


Figura 2. Graf no dirigit.

Exemple 2. A la figura 3 observem el graf de l'exemple 1 dirigit. En un graf dirigit les relacions no són simètriques i tenen una direcció: E (u, v) va del vèrtex u al vèrtex v, denotat amb una fletxa u → v.

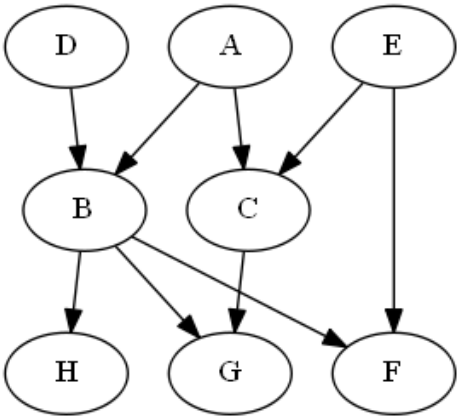


Figura 3. Graf dirigit.

Els grafs (dirigits o no dirigits) poden estar etiquetats o no etiquetats en funció de si les arestes tenen o no informació associada.

Donat un domini V de vèrtexs i un domini E d'etiquetes, es defineix un graf dirigit i etiquetat G com una funció que associa etiquetes a parelles de vèrtexs.

$$G \in \{ f: V \times V = E \}$$

Per a un graf no etiquetat la definició és similar. El graf G és una funció que associa un booleà a parelles de vèrtexs.

Exemple 3. A la figura 4 observem el graf de l'exemple 1 ponderat. Un graf ponderat és un tipus particular de graf etiquetat on la informació associada a l'aresta és un nombre real positiu que normalment representa un cost.

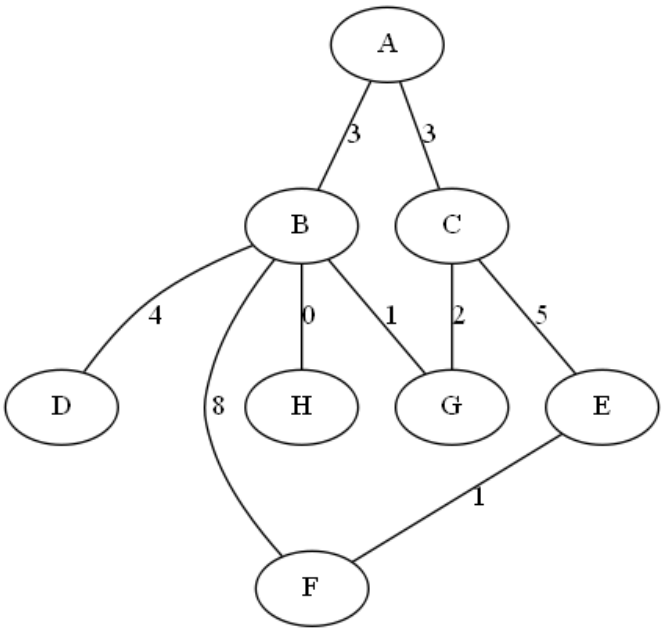


Figura 4. Graf ponderat.

Alguns conceptes que també s'han de tenir en compte:

- El grau d'un vèrtex v, definit com a deg(v), és el nombre d'arestes incidents sobre v.
- El camí d'un graf és una seqüència de vèrtexs connectats per les arestes. Un camí és un cicle, si comença i acaba en el mateix vèrtex. Un camí simple és un camí sense cap vèrtex repetit. Un graf està connectat si hi ha un camí entre cada parella de vèrtexs.

1.2.3 Teoria de les 2-estructures

La teoria de les 2-estructures va ser introduïda per Ehrenfeucht i Rozenberg el 1990.

Aquesta teoria proporciona una infraestructura matemàtica per a la descomposició i la transformació dels grafs. Es tracta d'un formalisme molt potent i robust que permet representar els grafs en una sola estructura algebraica, una 2-estructura, i derivar-ne d'ella una descomposició única en 2-estructures més simples.

Per a l'elaboració d'aquest treball s'ha treballat principalment amb l'estudi previ [3] on s'estableixen els fonaments de la teoria de les 2-estructures i que recopila i amplia el contingut sobre la teoria actual fins a la data de la seva publicació.

La part interessant que tracta el treball sobre la teoria de les 2-estructures (del capítol 3 al 10) es refereix principalment a la descomposició i a la no-descomposició. Del capítol 3 al 7 es considera la 2-estructura no etiquetada que es defineix per una relació d'equivalència en les arestes. Les 2-estructures no etiquetades representen l'estructura principal del paquet de software desenvolupat en aquest treball.

La teoria de les 2-estructures enriqueix l'estudi de les gramàtiques dels grafs posant a disposició les eines de la teoria de conjunts.

1.2.3.1 Conceptes bàsics

S'utilitza el nom genèric de 2-estructura per tal d'emfatitzar que investiguem els sistemes de relacions binàries en un entorn abstracte. Particularment en el treball, s'investiguen les representacions jeràrquiques de les 2-estructures a través d'arbres on la relació local o recursiva (quan es defineix en termes de si mateixa o del seu tipus) entre descendents directes d'un node interior es dóna a través d'una estructura binària.

La noció d'una 2-estructura és més general que la noció d'un graf i menys general que la noció d'una estructura relacional.

Una característica molt important de les 2-estructures és que poden descomposar-se en 2-estructures més simples. Aquesta descomposició és única per a les 2-estructures i consisteix en trobar subconjunts d'un graf, anomenats clans.

Exemple 4. A la figura 5 veiem una representació gràfica d'una 2-estructura. Cada 2-estructura es representa com una estructura en forma d'arbre. En aquest arbre en concret, cada node que es troba en la 2-estructura representa l'element d'un graf o bé un subconjunt d'aquest. Més endavant explicarem el procés de descomposició amb el qual s'obté aquesta representació.

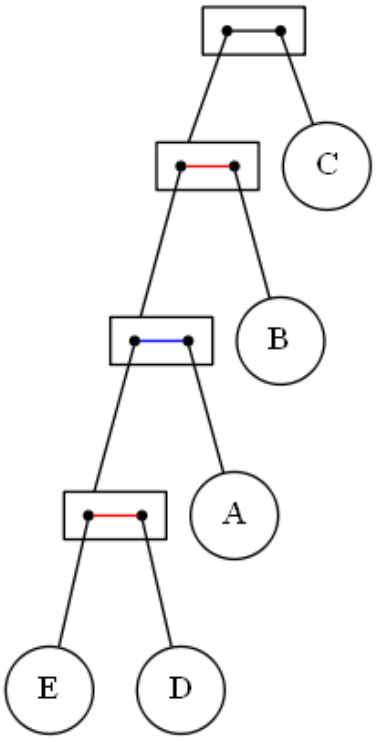


Figura 5. 2-estructura de l'exemple 4.

1.3 METODOLOGIA I AVALUACIÓ TECNOLÒGICA

1.3.1 Metodologia

Per a la organització del treball la metodologia triada, si bé és senzilla, ha resultat molt efectiva en el desenvolupament del treball. Com que era un treball on l'equip de desenvolupament consistia en una única persona (en aquest cas l'estudiant que realitza el treball), es va decidir seguir una metodologia similar a *Scrum*. Que és una metodologia àgil pensada sobretot pel desenvolupament de software i l'adaptabilitat de qualsevol canvi com a mitjà per augmentar les possibilitats d'èxit d'un projecte.

Una de les principals claus de l'èxit de *Scrum* és que està basat en la premissa de que durant el desenvolupament dels productes, els clients poden canviar les seves opinions sobre què volen i què necessiten. A més, els imprevistos no presenten una gran dificultat pel desenvolupament dels productes.

Per tant, es va decidir realitzar les reunions necessàries amb el director del treball constantment, amb l'objectiu de facilitar la proposta de solucions i la obtenció de *feedback*.

Els rols es van assignar de la següent manera:

- *Scrum Master*: és el que s'encarrega de fer el seguiment d'allò que realitza l'equip de desenvolupament, en aquest cas és el director del treball.
- *Product Owner*: és qui representa els interessats en el projecte, en aquest cas també és el director del treball.
- L'Equip de desenvolupament: és el que s'encarrega de desenvolupar el producte, en aquest cas és l'estudiant que realitza el treball.

També es va decidir utilitzar el model *Kanban* per a la organització de les tasques del treball mitjançant l'aplicació web Wrike [4]. Ja que el model *Kanban* proporciona un sistema de gestió del procés molt visual que ajuda a la presa de dedicions.

- Wrike: és una eina en línia per a la gestió de projectes i la col·laboració en el treball. Que permet als seus usuaris gestionar i realitzar un seguiment dels projectes, terminis, horaris i altres processos d'un flux de treball. També permet als usuaris col·laborar entre si. Wrike es pot trobar per a dispositius iOS i Android.

  Pràcticament totes les versions de Wrike compten amb un cicle de treball que actualitza els usuaris de qualsevol activitat realitzada per els altres usuaris dels grups de treball. Les característiques socials també estan integrades. A més també s'integra amb un seguit de productes diferents com: Google Apps, Microsoft Outlook, Microsoft Excel, Microsoft Project, Google Drive, Dropbox, Apple Mail Box, IBM Connections, i altres.

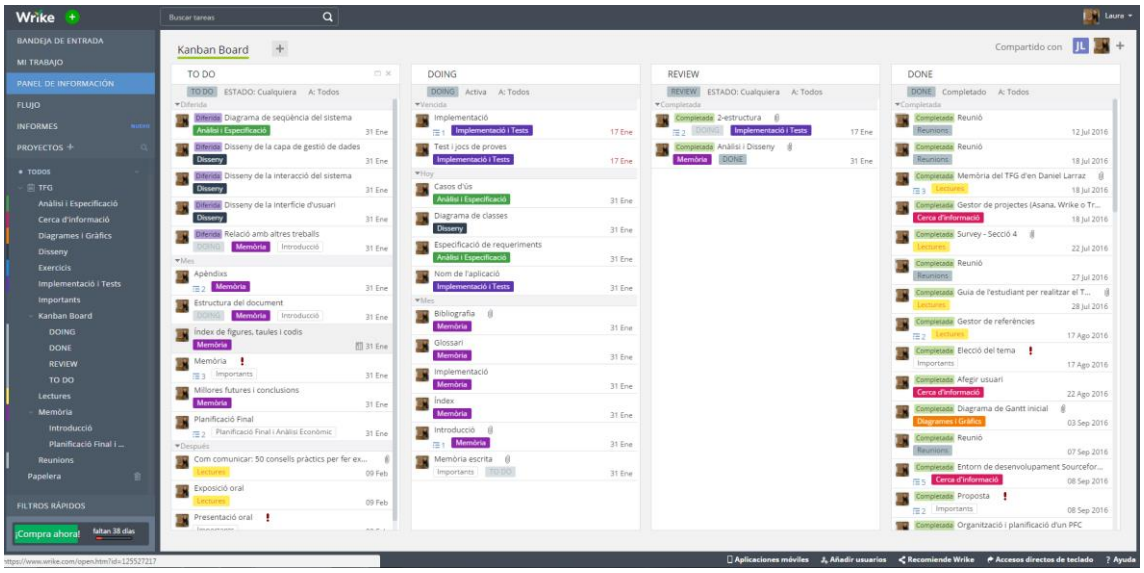


Figura 6. Taulell *Kanban* a Wrike.

Wrike incorpora la generació automàtica del diagrama de Gantt. Però per a la simplificació del diagrama es va optar per a importar les tasques de Wrike al format de fitxer XML. Triar les tasques més importants que mostrava i crear un nou diagrama de Gantt amb el programari GanttProject [5]. Perquè GanttProject està dissenyat seguint les bases del principi KISS. Aquest principi afirma que els sistemes que funcionen millor són els que es mantenen simples, i que s'ha d'evitar qualsevol complexitat afegida que no resulti clarament necessària.

- GanttProject és una aplicació d'escriptori multi plataforma per a la programació i gestió de projectes. La seva funcionalitat principal és la creació de diagrames Gantt. Facilita la exportació dels diagrames a formats de fitxers PNG, PDF i HTML. També s'integra amb Microsoft Project a l'hora d'importar i exportar projectes, i es complementa amb aplicacions que fan servir fulls de càlcul en formats de fitxer CSV. També permet la compartició dels projectes amb altres mitjançant WebDAV [6].

Figura 7. Logotip del programari GanttProject.

Per a la implementació i les proves del paquet de software desenvolupat, a mesura que s'anava avançant i es van començar a plantejar els primers diagrames de classes, es va triar la metodologia orientada a objectes.

Les bases d'aquest tipus de metodologia són:

- Tot és un objecte i cada objecte té una identitat pròpia.
- Un programa és un conjunt d'objectes que interactuen entre ells.
- Cada objecte pertany a un tipus d'objecte concret: una classe.
- Objectes del mateix tipus tenen un comportament idèntic.

El mètode d'avaluació proposat per a la implementació es divideix en:

a) Periòdicament es posava a prova la implementació mitjançant un conjunt de jocs de prova que es composen de casos senzills i aleatoris per assegurar-se de la pròpia correctesa.
b) Per comprovar la correcta visualització de la implementació mitjançant el programari interactiu extern es realitzen proves a nivell d'usuari (*beta-testing*) per a assegurar-se de que és suficientment intuïtiu i es comprova el funcionament en tots els seus àmbits. Òbviament aquestes comprovacions es realitzen en fases avançades del treball.

1.3.2 Avaluació tecnològica

1.3.2.1 Llenguatges de programació

El llenguatge de programació triat pel desenvolupament del paquet de software és Python [7]. Perquè és un llenguatge molt utilitzat en entorns de computació científica, ja que ofereix un entorn de programació obert i flexible, en el que és fàcil iniciar-se, gràcies a la seva filosofia de disseny, que busca llegibilitat en el codi, i la seva sintaxi, que permet expressar conceptes en menys línies de codi del que seria possible en altres llenguatges, com per exemple Java o C.

Figura 8. Logotip del llenguatge de programació Python.

Python està dissenyat per ser un llenguatge molt visual, i com a característica principal utilitza el sagnat. Això és poc comú en llenguatges de programació, ja que molts utilitzen delimitadors. A més, té una llibreria estàndard molt gran i aporta eines ja programades que poden crear una gran varietat de funcions.

Una de les característiques de Python que també s'ha utilitzat és la combinació amb altres tipus de llenguatge, C i C++, i els diferents mòduls per a connectar bases de dades relacionals.

El paquet de software implementat és compatible amb les versions de Python 2 i 3.

1.3.2.2 Llibreries

Pel tractament de les bases de dades relacionals es va utilitzar el mòdul sqlite3 [8] que proporciona una interfície SQL compatible amb l'especificació DB-API 2.0 (PEP 249 [9]).

- SQLite és una llibreria de C que proporciona una base de dades que no requereixen el procés d'un servidor independent i permet l'accés a la base de dades utilitzant una variant no estàndard del llenguatge de consultes SQL. També és possible crear prototips d'una aplicació que utilitza SQLite i després exportar el codi a una base de dades més gran, com ara PostgreSQL o Oracle.

En la creació i la manipulació dels grafs es va utilitzar la llibreria NetworkX [10].

- NetworkX és un paquet de programari de Python per a la creació, manipulació, i l'estudi de l'estructura, la dinàmica i les funcions de les xarxes complexes. L'audiència potencial de NetworkX inclou matemàtics, físics, biòlegs i informàtics. A NetworkX, els nodes poden ser qualsevol objecte, per exemple, una taula hash (estructura de dades que associa claus o claus amb valors), una cadena de text, una imatge, un objecte XML, un altre graf, etc.

  NetworkX proporciona:

  - Estructures de dades per a grafs, digrafs i multi-grafs.
  - Molts algoritmes de grafs estàndards.
  - Generadors de grafs simples, grafs aleatoris, etc.
  - Llicència BSD de codi obert [11].
  - Una interfície d'algoritmes numèrics existents i codificats en C, C++ i Fortran.
  - La capacitat de tractar grans quantitats de dades.
  - Fiabilitat amb més de 1800 proves d'unitat.

Per a generar tots els subconjunts possibles d'un graf, que té un alt cost de processament, i altres recorreguts, es va pensar en utilitzar la llibreria itertools [12] de Python. Ja que proporciona funcions per a recórrer conjunts de dades molt eficients. La llibreria s'inspira en construccions APL [13], Haskell [14] i SML [15].

Una vegada creats els grafs i les 2-estructures es necessitava una eina de visualització per a observar els resultats. La llibreria triada va ser PyDot [16].

- PyDot és una interfície per a Graphviz, que pot analitzar i bolcar en llenguatge DOT (llenguatge que utilitza el programari Graphviz) i està escrit en Python pur. A més, NetworkX pot convertir els seus grafs en aquest format a partir de la interfície PyDot. La compatibilitat entre les dues llibreries va ser un punt a favor.

El paquet de software implementat és compatible amb diferents sistemes operatius (Windows, Mac OS X i Ubuntu) i amb diferents versions de Python com s'ha comentat anteriorment. Per a aconseguir aquest objectiu s'han utilitzat les diferents llibreries: os, sys, subprocess i six, amb les seves funcionalitats respectives.

- Mòdul os: és una llibreria que permet l'ús de funcionalitats pròpies d'un sistema operatiu.
- Mòdul sys: és una llibreria que permet l'accés a algunes variables que introdueixen els usuaris utilitzades per a l'intèrpret d'ordres, que té la capacitat de traduir-les, i les funcions que interactuen amb ell.
- Mòdul subprocess: és una llibreria creada amb la intenció de reemplaçar la llibreria os.
- Mòdul six: llibreria de compatibilitat entre Python 2 i 3.

1.3.2.3 Programari

Les particularitats de la representació de la descomposició d'un graf en una 2-estructura i la creació d'una 2-estructura va provocar la cerca d'un programa extern per a la visualització. Aquesta eina és Graphviz [17].

- Graphviz és un programari de codi obert de visualització gràfica que converteix els programes de disseny en format DOT a diagrames o imatges.

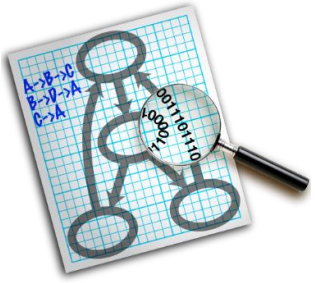


Figura 9. **Logotip del programari Graphviz.**

Exemple 5. A la figura 10 es pot veure el resultat de la conversió que efectua el programari Graphviz. A partir d'un fitxer dissenyat en format DOT (a l'esquerra de la figura 10) es crea la visualització corresponent (a la dreta de la figura 10).

```
digraph G {

        subgraph cluster_0 {
                style=filled;
                color=lightgrey;
                node [style=filled,color=white];
                a0 -> a1 -> a2 -> a3;
                label = "process #1";
        }

        subgraph cluster_1 {
                node [style=filled];
                b0 -> b1 -> b2 -> b3;
                label = "process #2";
                color=blue
        }
        start -> a0;
        start -> b0;
        a1 -> b3;
        b2 -> a3;
        a3 -> a0;
        a3 -> end;
        b3 -> end;

        start [shape=Mdiamond];
        end [shape=Msquare];
}
```

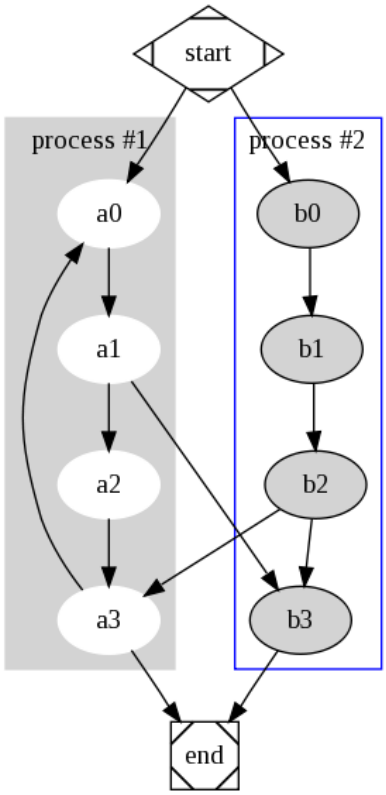


Figura 10. **Conversió del programari Graphviz, de format DOT a imatge.**

Durant el desenvolupament es va utilitzar l'entorn de programació integrat i multi plataforma PyCharm [18] i es va mantenir una copia actualitzada del paquet de software en un emmagatzematge remot, mitjançant GitHub [19].

- PyCharm està desenvolupat per la companyia Jetbrains, basat en IntelliJ IDEA, l'entorn de programació integrat de la mateixa companyia però enfocat cap al llenguatge de programació Java (que ja havia utilitzat anteriorment durant l'assignatura de Projecte de Programació) i la base d'Android Studio.

Principalment permet escriure bon codi (seguint les normes PEP-8 [20]), ràpid i eficient, gràcies a les funcions que incorpora. Ofereix compilació de codi intel·ligent, correccions de codi, ressaltat d'error en temps real i completesa de codi, juntament amb la refactorització de codi automatitzat i capacitats de navegació. Proporciona suport de primera classe per a Python, JavaScript, CoffeScript, TypeScript, CSS, i altres. També utilitza la cerca intel·ligent per saltar a qualsevol classe, arxiu o símbol, o fins i tot qualsevol finestra d'acció de l'entorn de desenvolupament o eina.

Figura 11. Logotip de PyCharm.

Entre les característiques més rellevants de PyCharm trobem:

- Forta incidència en la no-linealitat dels canvis.
- Gestió distribuïda. Els canvis s'importen com a ramificacions, i es poden barrejar de la manera en què ho fa una ramificació en l'emmagatzematge local.
- Els magatzems d'informació poden publicar-se per a HTTP, FTP, SSH, rsync o mitjançant un protocol natiu, a part de ser possible emular CVS [21].

PyCharm pot integrar el sistema de control de versions Git [22], que utilitza l' emmagatzematge remot GitHub. Aquest fet va afavorir la manipulació alhora d'utilitzar el control de versions.

- Git és un programari de sistema de control de versions dissenyat per Linus Torvalds, pensat per a l'eficiència i la confiabilitat del manteniment de versions d'aplicacions amb una enorme quantitat de fitxers de codi font. Proporciona una gestió eficient de projectes grans, gràcies a la rapidesa de gestió de diferents arxius, entre altres millores d'optimització en la velocitat d'execució.

Figura 12. Logotip de Git.

Per a accelerar el processament de les dades relacionals es va utilitzar el programari Apriori [23].

Apriori és un programa que utilitza l'algoritme Apriori [24]. L'algoritme Apriori és un algoritme molt utilitzat en la mineria de dades que s'utilitza per a trobar els conjunts d'elements més freqüents d'una base de dades. Es va triar perquè és un algoritme molt ràpid ja que utilitza un arbre definit anteriorment per a organitzar el conjunt d'elements més freqüents a cercar. Treballa amb els formats de fitxers ARFF i TXT.

1.3.2.4 Sistemes Gestors de Base de Dades (SGDB)

En la última fase del treball es va dur a terme la investigació de les possibles aplicacions de les 2-estructures en l'anàlisi de dades relacionals. I per a l'anàlisi i la gestió d'aquest tipus de dades es van triar les bases de dades SQLite.

- SQLite és una base de dades relacional continguda en una llibreria escrita en C. Incorpora un motor de bases de dades SQL independent. Que a diferència d'altres bases de dades relacionals, no és un sistema que funciona amb el paradigma client-servidor, sinó que s'integra a dins d'altres programes. Les bases de dades SQLite s'emmagatzemen en un fitxer (normalment amb extensió sqlite o db) que conté tant la definició de l'estructura de les dades com les mateixes dades.

Figura 13. Logotip de SQLite.

Bàsicament es va triar aquest tipus de base de dades relacional perquè el codi per a SQLite és de domini públic, multi plataforma i és la base de dades de major desplegament al món. A més, alhora de començar una línia d'investigació és molt important la senzillesa i la generalització que aporta.

1.4 Estructura del document

El treball s'estructura en dues parts: memòria i apèndixs. A la memòria es sintetitza les aportacions realitzades per l'estudiant. I a l'apèndix es dóna una guia d'instal·lació i ús de l'entorn d'anàlisi, i l'anàlisi dels algoritmes més importants de la implementació.

El contingut dels capítols de la memòria és el següent. En el capítol 2 es mostra la planificació que s'ha dut a terme durant el desenvolupament del treball. En el capítol 3 i 4 es presenta l'anàlisi econòmic i l'anàlisi de sostenibilitat. En el capítol 5 es mostra l'anàlisi i el disseny del paquet de software desenvolupat.

En els capítols 6, 7, 8, 9 i 10 es documenta la implementació del paquet de software. Concretament, en el capítol 6 es descriuen les dades d'entrada relacionals que necessita. En el capítol 7 es descriu la generació dels grafs. En el capítol 8 s'explica com funciona la descomposició dels grafs. El capítol 9 s'associa explícitament a les 2-estructures. I finalment, en el capítol 10 es desenvolupa un exemple que agrupa els capítols anteriors.

A l'últim capítol, l'11, es resumeix el treball realitzat, es dóna una valoració personal d'aquest i es relaciona amb les diferents maneres possibles de continuar-lo.

2. PLANIFICACIÓ TEMPORAL

2.1 DESCRIPCIÓ DE LES TASQUES

Inicialment el treball es va planificar fins al torn de presentació de febrer. No obstant, per la falta de temps es va prorrogar fins al torn de presentació de maig.

Per tant, la planificació del treball s'ha estès de la manera següent:

Fase 1 (Fase inicial): des del 11/07/2016 al 31/10/2016 al diagrama de Gantt. Fase d'entrar en contacte amb l'entorn de desenvolupament i valorar les diferents opcions i algoritmes a implementar. L'anàlisi i el disseny de la implementació estan inclosos en aquesta fase.

Fase 2 (Fase d'implementació bàsica): des del 2/11/2016 al 31/01/2017 al diagrama de Gantt. Fase on, un cop decidits els algoritmes, les llibreries de visualització i altres detalls, es comencen a implementar els algoritmes bàsics sobre unes dades mínimament realistes en un entorn de visualització senzill.

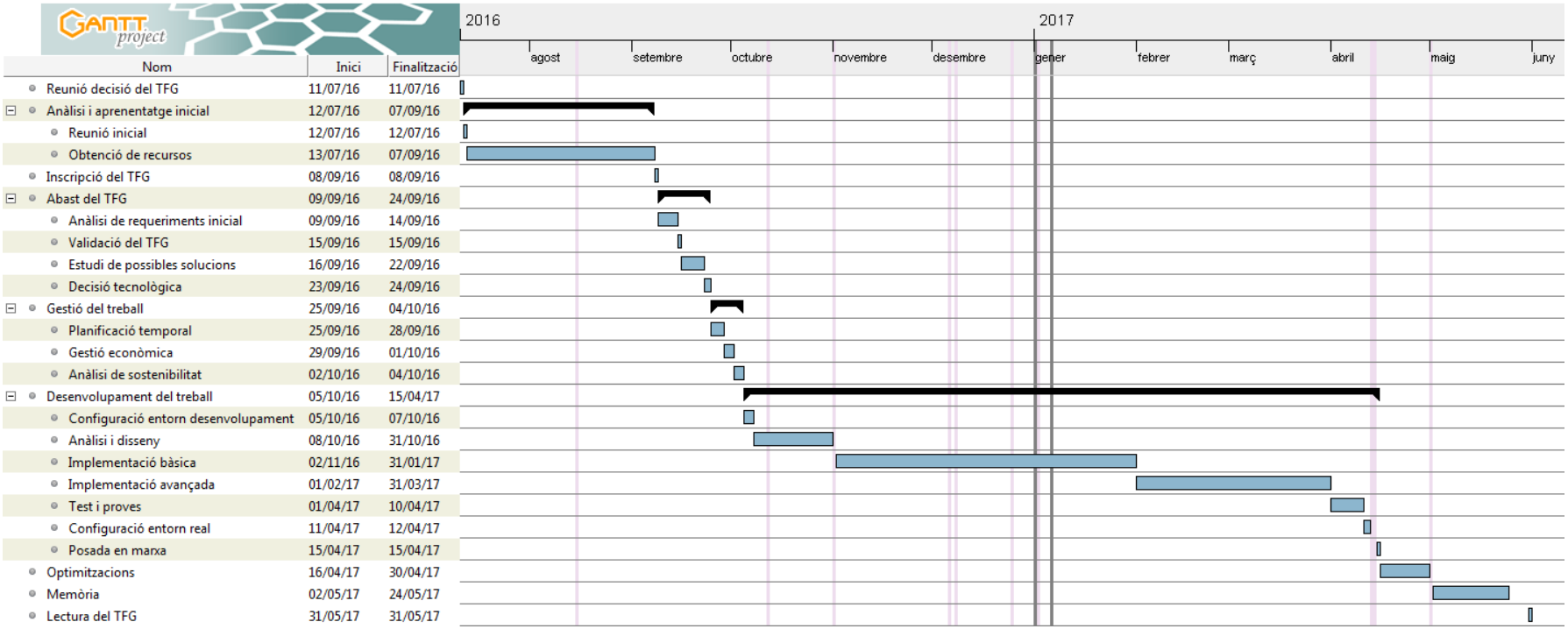
Fase 3 (Fase d'implementació avançada): des del 1/02/2017 al 31/03/2017 al diagrama de Gantt. Fase on, una vegada tenim la base, podem realitzar el procediment per acabar d'implementar els algoritmes definitius del treball i la integració d'aquests amb les bases de dades relacionals.

Fase 4 (Fase d'adaptació): des del 1/04/2017 al 15/04/2017 al diagrama de Gantt. Fase on, ja desenvolupats els algoritmes, els adaptem per a que siguin més específics i realistes de cara als aspectes de l'anàlisi de dades relacionals. Els tests de la implementació estan inclosos en aquesta fase.

Fase 5 (Fase d'optimitzacions i millores): des del 16/04/2017 al 30/04/2017 al diagrama de Gantt. Acabat el programari principal, es busquen tot tipus d'optimitzacions per a millorar el temps i els resultats del paquet de software.

2.2 DIAGRAMA DE GANTT

Aquí s'observa el diagrama de Gantt de la planificació temporal:



Nom	Inici	Finalització
Reunió decisió del TFG	11/07/16	11/07/16
Anàlisi i aprenentatge inicial	12/07/16	07/09/16
Reunió inicial	12/07/16	12/07/16
Obtenció de recursos	13/07/16	07/09/16
Inscripció del TFG	08/09/16	08/09/16
Abast del TFG	09/09/16	24/09/16
Anàlisi de requeriments inicial	09/09/16	14/09/16
Validació del TFG	15/09/16	15/09/16
Estudi de possibles solucions	16/09/16	22/09/16
Decisió tecnològica	23/09/16	24/09/16
Gestió del treball	25/09/16	04/10/16
Planificació temporal	25/09/16	28/09/16
Gestió econòmica	29/09/16	01/10/16
Anàlisi de sostenibilitat	02/10/16	04/10/16
Desenvolupament del treball	05/10/16	15/04/17
Configuració entorn desenvolupament	05/10/16	07/10/16
Anàlisi i disseny	08/10/16	31/10/16
Implementació bàsica	02/11/16	31/01/17
Implementació avançada	01/02/17	31/03/17
Test i proves	01/04/17	10/04/17
Configuració entorn real	11/04/17	12/04/17
Posada en marxa	15/04/17	15/04/17
Optimitzacions	16/04/17	30/04/17
Memòria	02/05/17	24/05/17
Lectura del TFG	31/05/17	31/05/17

Figura 14. Diagrama de Gantt.

3. GESTIÓ ECONÒMICA I PRESSUPOST

3.1 IDENTIFICACIÓ DE COSTOS

Podem dividir els costos involucrats d'aquest treball en: costos de recursos humans, de maquinari i de programari.

3.2 RECURSOS HUMANS

Pel desenvolupament d'aquest treball es necessita com a mínim un/a programador/a que tingui coneixement previ en ciències de la computació i tingui per tant el coneixement sobre la teoria de grafs i sobre la cerca i l'anàlisi d'informació massiva.

El sou brut anual d'un perfil d'aquest estil, amb una titulació superior en enginyeria informàtica, pot ser aproximadament uns 32.000 €. Sense tenir en compte els extres, aproximadament, uns 2.667 € al mes.

Per calcular els costos reals que suposa pel contractant hem d'afegir-hi les cotitzacions socials del 29,70% (23,60% de contingències comuns, 5,50% d'atur i 0,60% de formació professional).

Costos pel contractant	**Preus**
Sou brut mensual	2.667 €
+23,60% Cotitzacions de contingències comuns	629,41 €
+5,50% Cotitzacions d'atur	146,68 €
+0,60% Cotitzacions de formació professional	16 €
Total	3.459,09 €

Taula 1. Costos que suposa el treballador pel contractant.

Pel que fa al/a treballador/a se li descompta del sou brut mensual: el 6,35% de cotitzacions socials (4,70% de contingències comuns, 1.55% d'atur, 0,10% de formació professional) i el 20% de IRPF.

Costos pel treballador	Preus
Sou brut mensual	2.667 €
-4,70% Cotitzacions de contingències comuns	-125,35 €
-1,55% Cotitzacions d'atur	-41,34 €
-0,10% Cotitzacions de formació professional	-2,67 €
-20% de IRPF	-533,40 €
Total	1.964,24 €

Taula 2. Sou net que rep el treballador.

3.3 Recursos de maquinari

Els recursos de maquinari necessaris per a aquest treball són molt limitats. Bàsicament es necessita un únic ordinador de gama alta per a la realització de les proves dels algoritmes, el control de versions extern, la gestió i l'administració d'una base de dades i per a l'execució del programari corresponent.

No obstant, com que ja es disposava d'un ordinador de gama alta, els costos de maquinari són mínims i es resumeixen de la següent manera:

Maquinari	Preus
Ordinador de gama alta	(1.200 €) 0 €
Total	0 €

Taula 3. Preu del maquinari.

3.4 Recursos de programari

Tots els recursos de programari que s'utilitzen no comporten cap despesa. Fins i tot, el programari que normalment és de pagament, s'ha pogut demanar la llicència estudiant durant un any i s'ha utilitzat gratuïtament.

També es tenen en compte les dades relacionals que s'utilitzen en aquest treball, però que tampoc tenen cap cost degut a que són proporcionades pel director del treball.

Programari	Preus
Llibreries (NetworkX, PyDot, etc.)	0 €
Programari Graphviz	0 €
Programari PyCharm Professional	(199 €/any) 0 €
Aplicació Web Wrike Professional	(588 €/any) 0 €
Programari Astah Professional	(57 €/any) 0 €
Control de versions GitHub	0 €
Data Sources	0 €
Total	0 €

Taula 4. Preu del programari.

3.5 RESUM DE COSTOS

Finalment, podem resumir els diferents costos mensuals de recursos humans, de maquinari i de programari de la següent manera:

Concepte	Preus
Salaris	3.459,09 €
Amortitzacions de maquinari	0 €
Manteniment de llicències de programari	0 €
Total	3.459,09 €

Taula 5. Resum de costos mensuals.

I el cost total d'aquest treball, si tenim en compte que es realitza durant 11 mesos, aproximadament, es resumiria de la següent manera:

Concepte	Preus
Salaris	38.049,99 €
Amortitzacions de maquinari	0 €
Manteniment de llicències de programari	0 €
Total	38.049,99 €

Taula 6. Resum de costos totals.

S'ha de tenir en compte que el cost real d'aquest projecte, al estar realitzat dins de l'àmbit d'un Treball Final de Grau de modalitat A, seria nul en el concepte de salaris.

4. SOSTENIBILITAT I COMPROMÍS SOCIAL

4.1 VALORACIÓ DE LA SOSTENIBILITAT DEL TREBALL

La viabilitat econòmica d'aquest treball és alta, principalment perquè els recursos a invertir són mínims i si el treball acaba sent utilitzat per alguna empresa, la devolució de la inversió resultant seria elevada. A més els costos de materials són mínims, ja que és un treball purament de programació, i el material que s'ha utilitzat ja s'havia adquirit anteriorment. Per aquest motiu, pel fet de reutilitzar material existent, la valoració ambiental és molt alta. I una vegada s'acabi la vida útil del material d'aquest projecte es pot seguir reutilitzant, per exemple donant-l'hi a la associació de la UPC, Tecnologia per a Tothom (TxT) [25], que permet reutilitzar ordinadors antics per a donar-los a gent més necessitada.

La viabilitat social d'aquest treball també és bona, ja que amb la seva aplicació es podrà millorar l'eficiència de l'anàlisi de dades relacionals, i per tant suposarà la millora tant com per a les empreses que perden molt de temps amb un munt de dades per a analitzar com pels treballadors que les analitzen.

En els següents apartats es detallen els aspectes de sostenibilitat econòmics, socials i ambientals responent un conjunt de preguntes.

4.2 ASPECTES ECONÒMICS

- *El cost del treball ho faria viable si hagués de ser competitiu?*

Sí. Perquè només es necessita un sol programador/a amb coneixement de ciències de la computació. I el maquinari i el programari són limitats o directament gratuïts com s'ha comentat a l'anàlisi econòmic. No obstant, s'hauria de comptar amb la col·laboració d'alguna empresa del sector Big Data que tingui accés a més dades. Cosa que faria augmentar el cost del treball. Però, tot i així seria viable.

- *Es podria realitzar un treball similar en molt menys temps o amb molts menys recursos, i per tant menor cost?*

Es segur que amb més recursos, com per exemple amb més programadors i els seus propis ordinadors es podria realitzar aquest treball més ràpidament. Però és difícil reduir el cost en general, ja que, com s'ha comentat anteriorment els costos del treball són mínims.

- *El temps dedicat a cada tasca és proporcional a la seva importància (s'ha dedicat molt de temps a desenvolupar parts del treball que podien haver estat reutilitzades amb tecnologies / projectes / coneixements existents)?*

Totes les tasques desenvolupades en aquests treball no tenen una base ja existent, s'ha portat a terme un treball d'investigació. I s'ha dedicat més temps del normal en general.

4.3 ASPECTES SOCIALS

- *Hi ha una necessitat real del producte / servei?*

L'anàlisi de les dades relacionals per part de les empreses "no és un treball futur", sinó que és ja essencial per a la seva estratègia de negoci i cap empresa, independentment de la seva mida, n'hauria de quedar fora, donat que s'ha demostrat que és un element clau que permet prendre millors decisions, tant estratègiques com en el dia a dia, fins al punt de anticipar-se al mercat. Amb aquest treball es vol millorar l'anàlisi d'aquestes dades i l'adaptació a la situació actual de forma molt més eficient.

- *Satisfer aquesta necessitat, millora la qualitat de vida dels consumidors?*

Una major quantitat de dades implica més vulnerabilitat per als consumidors ja que n'augmenta el perill d'ús maliciós. I protegir el consumidor d'això és fonamental.

A més, l'ús de l'anàlisi de dades en camps com en la fixació de preus, una tendència a l'alça, pot servir per equilibrar el preu de venda en temps real en funció de l'oferta i la demanda. Com a conseqüència tindríem preus més adients pels consumidors.

4.4 ASPECTES AMBIENTALS

- Quins recursos es poden reaprofitar d'altres treballs?

La majoria del maquinari que s'utilitza en aquest projecte no s'ha adquirit per a la realització del mateix.

- *S'ha tingut en compte el desmantellament del procés/producte al medi ambient? Si és un producte, s'han tingut en compte en el seu disseny criteris de facilitació de les seves posterior reciclatge?*

Una vegada acabi la vida útil dels ordinadors utilitzats durant el treball es poden donar a l'associació per a la reutilització dels ordinadors de la UPC anomenada TxT (Tecnologia per a Tothom) que readaptaran els ordinadors per a poder distribuir-los a persones que no en tenen.

PART II

IMPLEMENTACIÓ

5. ANÀLISI I DISSENY DE LES 2-ESTRUCUTRES

L'anàlisi i el disseny realitzat per a la implementació es comenta breument a continuació. Concretament, el model de dades de les 2-estructures i les decisions de disseny adoptades.

A la figura 15 es mostra el diagrama de classes amb les representacions de les principals estructures i les seves relacions[1].

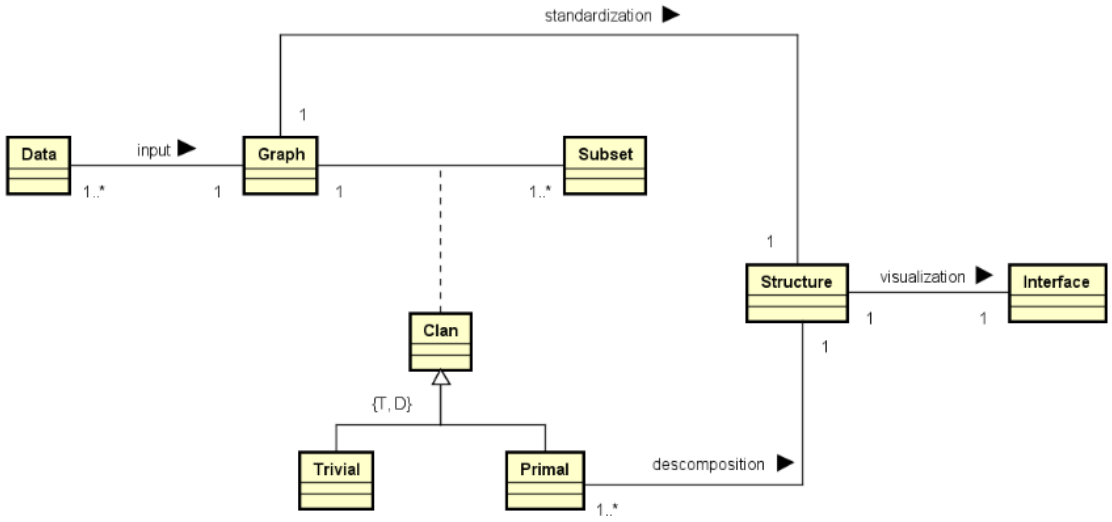


Figura 15. Diagrama de classes amb el modelat i el disseny de les 2-estructures.

La classe Data gestiona i adequa les dades relacionals d'entrada per a crear el Graph.

La classe Graph crea el graf:Graph amb les dades relacionals ajustades per la classe Data. El graf resultant s'utilitza per a la construcció de la 2-estructura:Structure.

La creació de la 2-estrcutura:Structure es pot dividir en tres parts:

- Inicialització del graf:Graph.
- Representació de les classes d'equivalències del graf:Graph.
- Exportació del graf:Graph.

Des del primer moment es va pensar en representar les classes d'equivalències amb les arestes del graf:Graph, pintant-les de diferents colors. Perquè el resultat final que volem sigui el més visual i intuïtiu possible. El graf:Graph pot ser de tipus pla, pla amb llindar, lineal o exponencial. El tipus de graf: Graph amb el que es crea la 2-estructur:Structure indica el tipus de la 2-estructura:Structure. Per tant, el graf:Graph i la 2-estructura:Structure tenen la mateixa tipificació.

La classe Subset genera tots els subconjunts possibles que del graf:Graph a partir d'ell. I per a cada parella de valors <Graph, Subset> existeix un únic clan:Clan, el qual constitueix la descomposició del graf:Graph i representa la classe Clan. La descomposició del graf:Graph està formada per un número finit de subconjunts d'aquests, segons si tenen la mateixa classe d'equivalència o no.

L'ús dels clans millora l'eficiència de les consultes sobre les propietats de les 2-estructures perquè es divideix el graf:Graph en seccions més petites.

Com que un clan:Clan pot ser trivial o primer, els tipus de clans que s'expliquen més endavant, es generalitza amb aquestes dues subclasses a la figura 15 anterior, Trivial i Primal. Es va prendre aquesta decisió perquè una 2-estructura:Structure està formada de clans trivials i primers.

La classe Trivial representa els clans de longitud u i el clan que conté tots els elements del graf:Graph. A més, un clan:Clan trivial sempre es considera que també és un clan:Clan primer. I la classe Primal representa els clans que no experimenten superposició. Propietat que compleixen tots els clans primers i que s'explica més detalladament en el capítol 8 d'aquest document.

La classe Structure crea la 2-estructura:Structure que està basada en la descomposició del graf:Graph en clans primers. Com s'ha comentat anteriorment, el graf:Graph i la 2-estructura:Structure tenen la mateixa tipificació, i per tant la 2-estructura:Structure també pot ser de tipus pla, pla amb llindar, lineal o exponencial.

Finalment la classe Interface genera la visualització del graph:Graph i la de la 2-estructura:Structure amb l'ajuda del programari interactiu extern anomenat Graphviz.

A continuació es mostra el diagrama de classes complet amb les representacions de les principals estructures, les seves relacions i operacions:

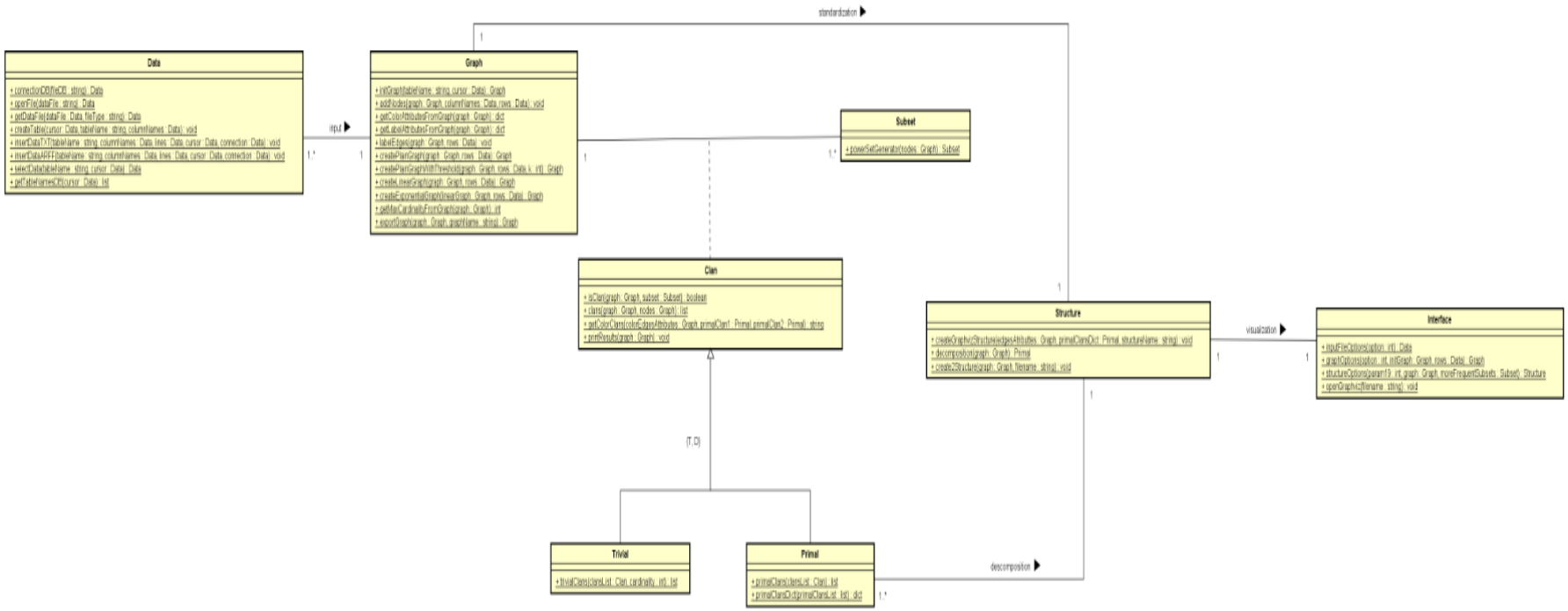


Figura 16. Diagrama de classes complet amb el modelat i el disseny de les 2-estructures.

Durant la fase d'optimització i millores es va comprovar que la unitat central de processament, durant l'execució del paquet de software amb grans grups de dades, anava molt lenta. Per aquest motiu es va modificar el disseny anteriorment esmentat.

La classe Subset enlloc de generar tots els subconjunts possibles del graf:Graph, generarà els subconjunts més freqüents possibles del graf:Graph.

A la figura 17 es mostra el nou diagrama de classes amb les representacions de les principals estructures i les seves relacions[1].

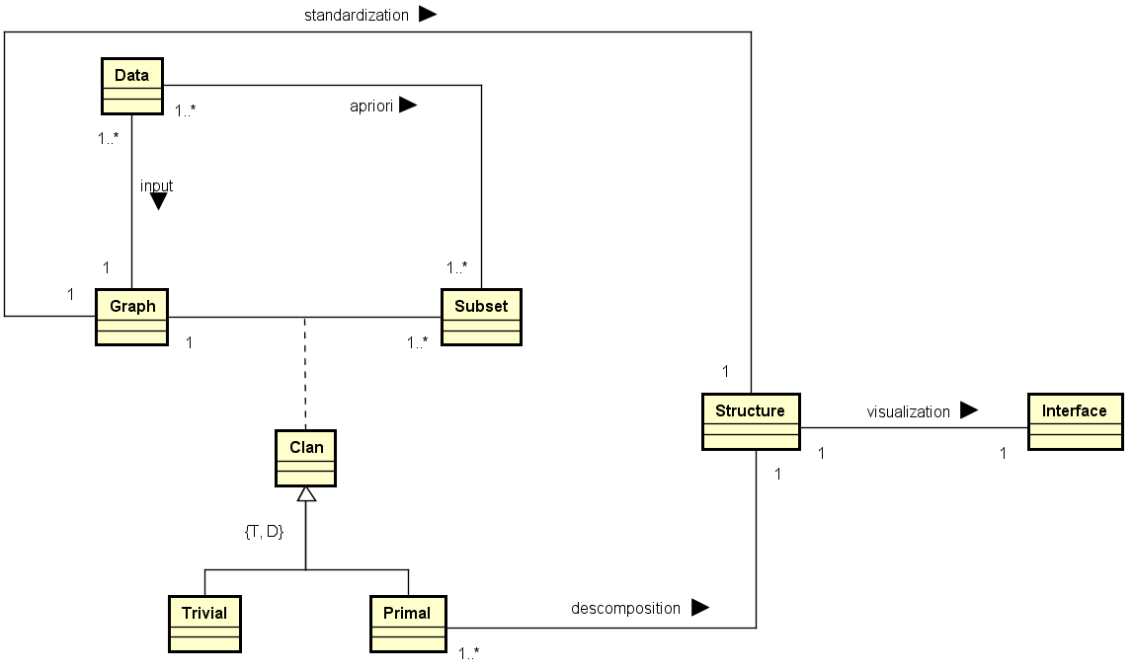


Figura 17. Nou diagrama de classes amb el modelat i el disseny de les 2-estructures.

En aquest nou model de disseny, la classe Data gestiona i adequa les dades relacionals d'entrada per a la classe Subset que generarà els subconjunts més freqüents possibles del graf:Graph.

[1] S'han omès els atributs i les operacions de les classes per claredat.

A continuació es mostra el nou diagrama de classes complet amb les representacions de les principals estructures, les seves relacions i operacions:

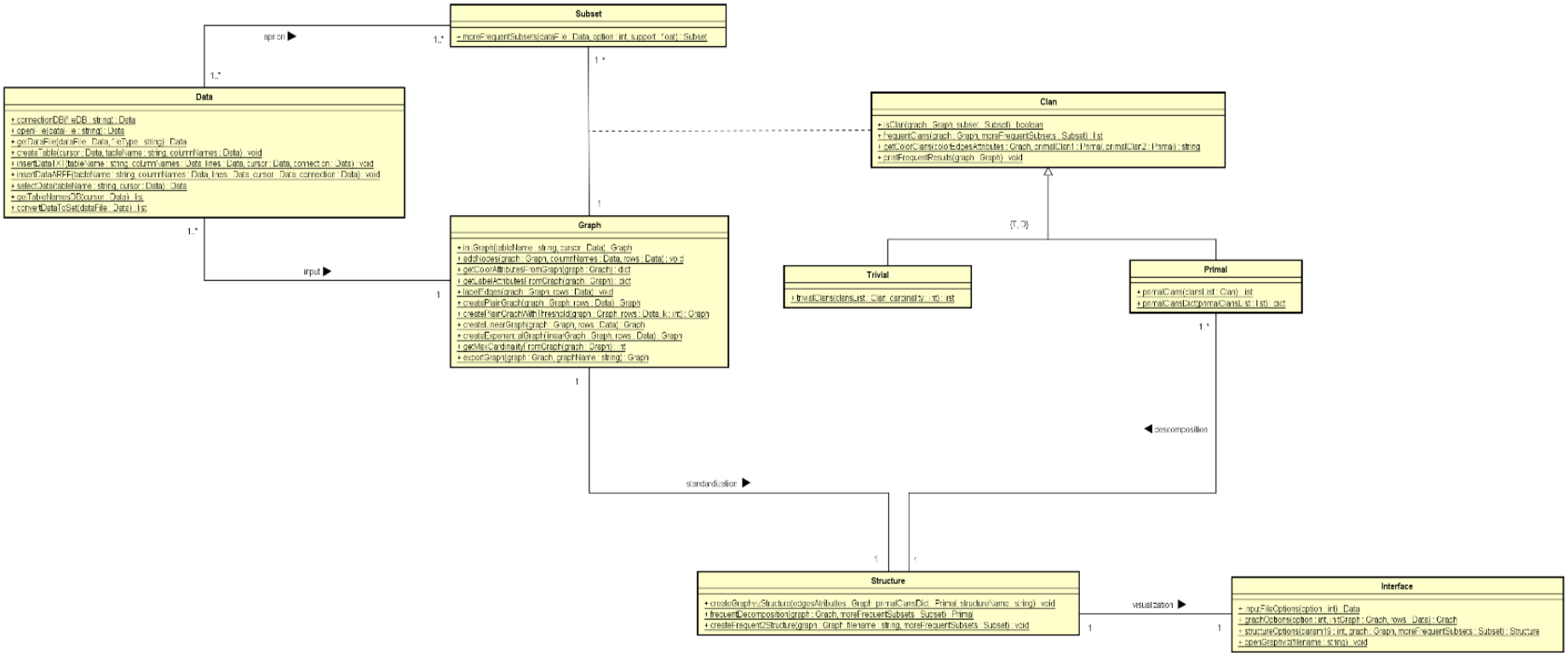


Figura 18. Nou diagrama de classes complet amb el modelat i el disseny de les 2-estructures.

6. DADES D'ENTRADA

Alhora de treballar amb les dades relacionals, que necessiten les bases de dades relacionals amb les que s'implementa el paquet de software, les podem dividir en tres tipus de fitxers. O sigui que només es permet la càrrega de dades relacionals pel paquet de software desenvolupat dels tipus de fitxers següents:

- Fitxers de tipus ARFF
- Fitxers de tipus TXT
- Fitxers de tipus DB

Els fitxers de dades contenen les dades relacionals necessàries per a la creació de les taules SQLite, que s'utilitzen per a la creació dels grafs i també per a la generació de tots o els subconjunts més freqüents de cada graf. I conseqüentment per a cada 2-estructura. Cada fitxer de dades és equivalent a una taula SQLite.

Es van triar els fitxers de tipus ARFF i BD perquè són dos formats molt utilitzats en el camp de la mineria de dades. Els fitxers de tipus TXT també s'utilitzen però la seva elecció va lligada a la simplicitat i els pocs errors que aquest tipus de fitxers provoquen.

6.2 FITXERS ARFF

Els fitxers ARFF van ser desenvolupats pel Projecte d'aprenentatge automàtic, en el Departament de Ciències de la Computació de la Universitat de Waikato (Nova Zelanda), per a utilitzar-los amb el programari d'aprenentatge de màquina Weka [26].

Un fitxer ARFF (Attribute Relation File Format) és un arxiu de text ASCII que descriu una llista d'instàncies que comparteixen un conjunt d'atributs. L'estructura del fitxer és molt senzilla i es divideix en 3 seccions: @relation, @attribute i @data.

- *@relation <relation-name>*

Tots els fitxer ARFF han de començar amb aquesta declaració a la primera línia ja que no es poden deixar línies en blanc al inici del fitxer. *<relation-name>* és una cadena de caràcters que conté el nom que se li vol donar a *@relation*.

- *@attribute <attribute-name> <datatype>*

En aquesta secció s'inclou una línia per a cada atribut (o columna equivalent a la taula SQLite) que es vulgui incloure en el conjunt de dades, indicant el nom i el tipus de dada. Amb <*attribute-name*> es proporciona el nom de l'atribut, que ha de començar per una lletra. I amb <*datatype*> es representa el tipus de dada per a aquest atribut (o columna equivalent a la taula SQLite).

El tipus de dada de <*datatype*> pot ser:

- Numèric. Nombres reals o enters.
- String (text).
- Date [<*date-format>]* (data). Format de la data, que és del tipus "yyyy-MM-dd'T'HH: mm: ss".
- <*nominal-specification*>. Llista de possibles valors: *{<nominal-name1>, <nominal-name2>, <nominal-name3>, ...}.*

Els tipus de dades numèric, text i data són sensibles a les majúscules. I el tipus de dada text és molt útil en aplicacions de mineria de dades, ja que permet crear conjunts de dades a partir de diferents cadenes d'atributs.

- *@data*

En l'ultima secció s'inclouen les dades pròpiament dites. Les dades de cada columna de la taula SQLite es troben separades per comes i totes les files (cada línia del fitxer a partir del nom de secció @data és equivalent a una fila de la taula SQLite) han de tenir el mateix nombre de columnes, el número total de declaracions *@attribute* de la secció anterior.

Si no es disposa d'alguna dada, es col·loca un signe d'interrogació (?) en el seu lloc. I el separador de decimals dels valors numèrics ha de ser obligatòriament un punt.

Exemple 6. A la figura 19 es pot veure el format d'un fitxer ARFF apte per a l'entorn de proves del paquet de software desenvolupat.

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,hot,high,FALSE,no
sunny,hot,high,TRUE,no
overcast,hot,high,FALSE,yes
rainy,mild,high,FALSE,yes
rainy,cool,normal,FALSE,yes
rainy,cool,normal,TRUE,no
overcast,cool,normal,TRUE,yes
sunny,mild,high,FALSE,no
sunny,cool,normal,FALSE,yes
rainy,mild,normal,FALSE,yes
sunny,mild,normal,TRUE,yes
overcast,mild,high,TRUE,yes
overcast,hot,normal,FALSE,yes
rainy,mild,high,TRUE,no
```

Figura 19. Fitxer ARFF.

El els fitxers ARFF també es poden afegir línies que comencin amb el caràcter % que representen comentaris. Totes les declaracions (@relation, @attribute i @data) són sensibles a majúscules.

6.2 FITXERS TXT

Els fitxers TXT són documents de text estàndard que contenen text sense format. Això fa que siguin reconeguts per a qualsevol programa de processament d'edició de text.

A causa de la seva simplicitat, els fitxers de text s'utilitzen molt sovint per a l'emmagatzematge d'informació. Ja que eviten alguns dels problemes trobats en altres formats de fitxer, com ara l'ordre de bytes (*endianness*), els bytes de farciment, o diferències en el nombre de bytes en una paraula màquina.

Un desavantatge dels fitxers de text és que en general tenen una entropia baixa. Això vol dir que la informació ocupa més espai d'emmagatzematge del necessari.

Exemple 7. A la figura 20 es pot veure el fitxer ARFF de l'exemple 6 però en format TXT apte per a l'entorn de proves del paquet de software.

```
outlook temperature humidity windy play
outlook:sunny temperature:hot humidity:high windy:FALSE play:no
outlook:sunny temperature:hot humidity:high windy:TRUE play:no
outlook:overcast temperature:hot humidity:high windy:FALSE play:yes
outlook:rainy temperature:mild humidity:high windy:FALSE play:yes
outlook:rainy temperature:cool humidity:normal windy:FALSE play:yes
outlook:rainy temperature:cool humidity:normal windy:TRUE play:no
outlook:overcast temperature:cool humidity:normal windy:TRUE play:yes
outlook:sunny temperature:mild humidity:high windy:FALSE play:no
outlook:sunny temperature:cool humidity:normal windy:FALSE play:yes
outlook:rainy temperature:mild humidity:normal windy:FALSE play:yes
outlook:sunny temperature:mild humidity:normal windy:TRUE play:yes
outlook:overcast temperature:mild humidity:high windy:TRUE play:yes
outlook:overcast temperature:hot humidity:normal windy:FALSE play:yes
outlook:rainy temperature:mild humidity:high windy:TRUE play:no
```

Figura 20. Fitxer TXT.

A la primera línia del fitxer, cada cadena de caràcters representa una columna de la taula SQLite. I la resta de línies contenen les dades per a cada fila de la taula SQlite. Cada fila està formada per parelles de valors <columna:dada (columna, fila)>.

6.3 FITXERS DB

Els fitxers DB tenen un format de fitxers de bases de dades genèriques. La informació que emmagatzema un fitxer d'aquest tipus normalment es composa d'una sèrie de taules, els camps d'aquestes taules i dades per a aquests camps. Una vegada s'han emmagatzemat les dades, la informació és organitzada d'acord amb el model de dades corresponent. El model d'estructura del treball és el model de dades relacional.

Una característica molt rellevant és que es poden exportar a un altre format. Al format CSV. El format CVS és molt senzill i els fitxers amb aquest format bàsicament s'utilitzen per a representar dades en forma de taula. Els camps de la taula de dades es troben separats per comes. I igual que amb els fitxers TXT, són reconeguts per a qualsevol programa de processament d'edició de text.

7. GRAFS

Com s'ha comentat anteriorment, una 2-estructura es pot considerar un graf dirigit complet amb les arestes acolorides. Per tant, tot graf desenvolupat en el paquet de software és complet.

Un graf és complet quan entre totes les parelles de vèrtexs del graf existeix una aresta.

Suposant que |E| és el nombre total d'arestes d'un graf (o cardinalitat) podem afirmar que:

- $|E| = n \cdot (n-1)$ si el graf és dirigit.
- $|E| = n \cdot (n-1) / 2$ si el graf no és dirigit.

Exemple 8. A la figura 21 es pot observar un graf complet i no dirigit de 5 nodes. L'etiquetatge dels nodes és de 0 a n-1. I el nombre total d'arestes és 10 ($5 \cdot (5-1) / 2$). Per a cada parella de nodes existeix una aresta.

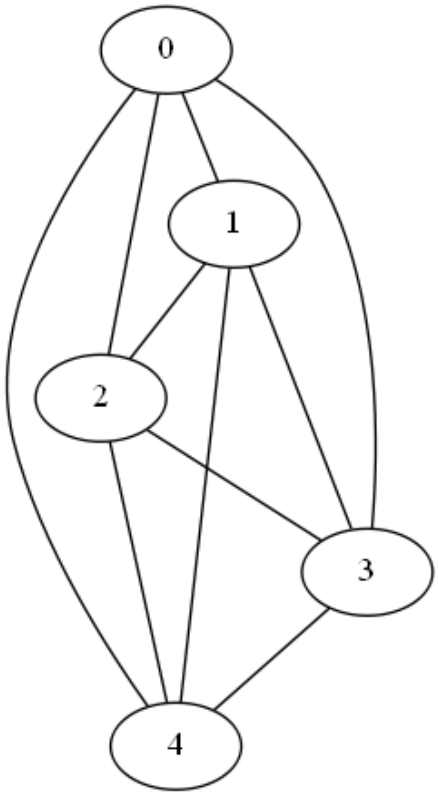


Figura 21. Graf complet.

A més, tots els grafs desenvolupats en el paquet de software també se'ls pot anomenar grafs de Gaifman [27]. Un graf de Gaifman d'una estructura relacional és un graf subjacent de l'hipergraf que representa l'estructura. A continuació en podem veure un exemple.

Exemple 9. Considerant una base de dades que conté la taula T1 següent:

col_1	col_2
a	b
b	c
d	b
b	c
a	c
d	c
f	b
e	g
f	h
g	c

Taula 7. Taula T1 de l'exemple 9.

El graf de Gaifman o també anomenat graf primer per a aquesta taula és el graf no dirigit:

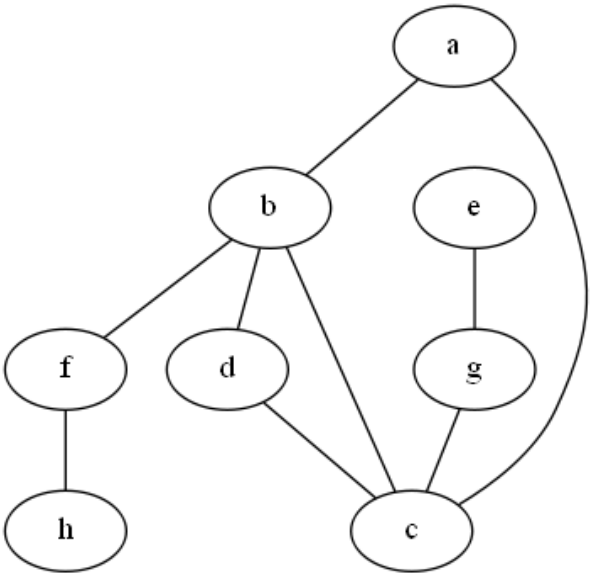


Figura 22. Graf de Gaifman de l'exemple 9.

En el graf de la figura 22, per a cada parella de valors hi ha una aresta entre ella si els valors d'aquesta es troben relacionats en alguna de les files de la taula T1. Per exemple, els valors a i b es troben junts a la primera fila i per tant existeix una aresta entre ells en el graf. D'altra banda, no hi ha cap fila que contingui la relació entre els elements a i g, i per tant no hi ha cap aresta que la representi en el graf.

7.1 CREACIÓ DELS GRAFS

7.1.1 Inicialització dels grafs

Inicialment a partir de la llibreria NetworkX es crea un graf buit (sense nodes ni arestes). Tots els grafs seran acíclics, no suporten cicles.

Després, donada una taula SQLite d'una base de dades relacional, ja anteriorment creada a partir dels tipus de fitxers que conté les dades d'entrada, es seleccionen les per a afegir-les com a nodes i a arestes del graf. Cada element d'una fila (sense comptar els elements repetits) representa un node del graf i per a cada parella de valors de cada fila de la taula es crea una aresta amb aquest valors. També obviarem les repeticions de cada parella de d'elements relacionats de cada fila.

Exemple 10. Considerant la taula T2 següent:

A	B	C	D
a	b	d	e
a	f	d	e
c	c	c	c

Taula 8. Taula T2 de l'exemple 10.

Es seleccionen totes les files de la taula T2 per a la inicialització del graf. Cada element no repetit d'una fila representa un node en el graf. Per tant, els nodes del graf seran: a, c, b, f, d i e. El nombre total de nodes del graf serà igual a sis. I per a cada parella d'elements no repetida en cada fila es crearan les arestes entre els dos elements que la representen. Per exemple, a la primera fila de la taula existeix una relació entre els nodes a-b, a-d i a-e. Com a conseqüència a la figura 23 podem comprovar que el node a té una aresta amb els nodes b, d i e.

A la figura 23 s'observa la inicialització del graf amb els valors de la taula T2. Les arestes de cada parella de nodes inicialment són discontínues.

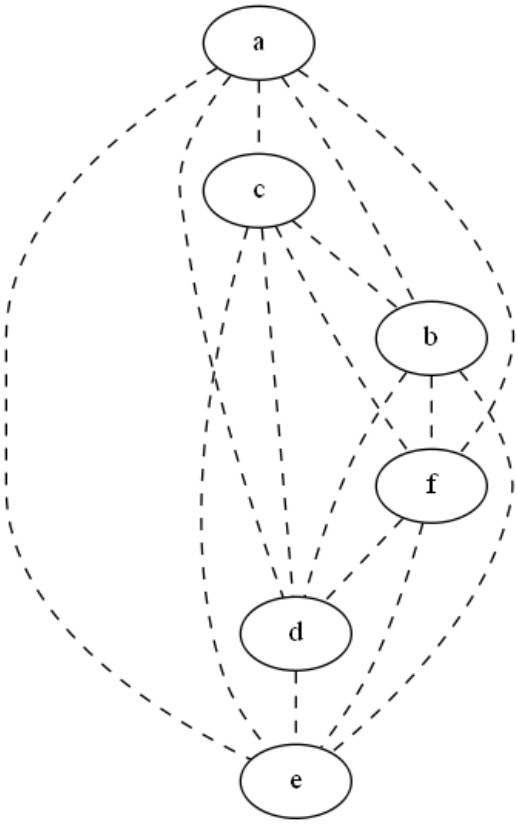


Figura 23. Inicialització del graf de l'exemple 10.

L'algoritme que inicialitza els grafs s'anomena initGraph [init_graph].

7.1.2 Classes d'equivalència de les arestes dels grafs

Una classe d'equivalència és el nombre de parelles de valors repetides de les dades relacionals que conté una taula SQLite. O sigui que cada parella d'elements relacionada en més d'una fila de la taula representarà una classe d'equivalència.

Cada classe d'equivalència s'indica amb les arestes acolorides dels grafs. Dues arestes del graf tindran el mateix color si pertanyen a la mateixa classe d'equivalència.

Abans de passar a la visualització d'un exemple donarem una última definició en relació de les arestes dels grafs que és necessari que coneguem.

Donada una aresta (u, v) es diu que és simètrica si i només sí l'aresta (v, u) pertany a la mateixa classe d'equivalència. Sinó es diu que és antisimètrica. Així doncs, estructura és simètrica si totes les seves arestes són simètriques. I una estructura és antisimètrica si totes les seves arestes són antisimètriques.

Exemple 11. A la figura 24 es mostra el graf equivalent a una 2-estructura. La estructura és simètrica perquè totes les arestes són simètriques: (u, v) = (v, u). En aquest graf existeixen tres classes d'equivalències: la negra, la blava i la vermella.

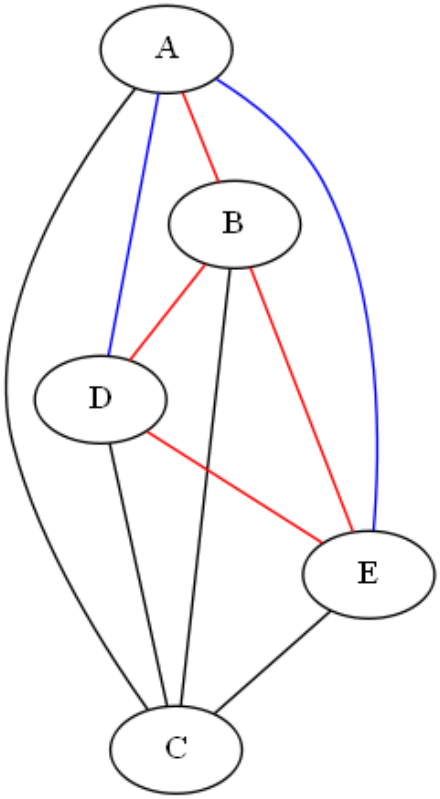


Figura 24. Graf equivalent a una 2-estructura de l'exemple 11.

Notar que a l'exemple 11, l'element C es relaciona de la mateixa manera amb els nodes A, B, D i E (arestes de color negre). Això vol dir que, aquests nodes pertanyen a la mateixa classe d'equivalència. També ho podem observar amb el node B que es relaciona de la mateixa manera amb els nodes A, D i E (arestes de color vermell). I el node A que ho fa amb els nodes E i D (arestes de color blau).

7.1.2.2 Nombre d'equivalències

Després de la inicialització dels graf, es tornen a recórrer totes les files de la taula SQLite amb la que l'hem inicialitzat. Ara però, per a cada fila de la taula es comprova si cada parella d'elements existeix com a mínim una vegada en tota la taula (parelles de valors repetits a la taula) i es compten el nombre de repeticions de cada parella d'elements. Després s'etiqueta l'aresta equivalent amb aquest nombre.

El procediment el duu a terme l'algoritme labeledEdges [labeled_edges]. L'algoritme compte el número de repeticions de cada parella de valors relacionada a la taula. I per a cada aresta del graf que les representa, les dibuixa i les etiqueta amb aquest nombre.

Després de l'execució de l'algoritme, el graf de la figura 23 resultant:

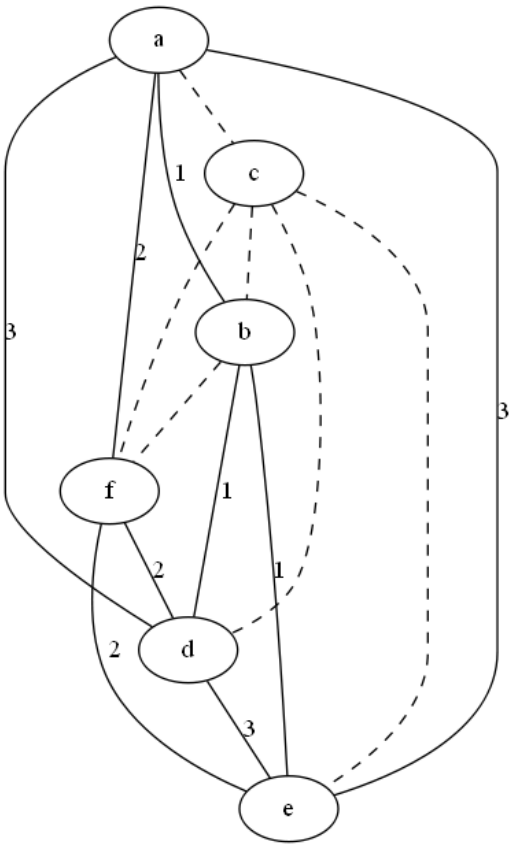


Figura 25. Graf de la figura 23 després de l'execució de l'algoritme labeledEdges.

El graf de la figura 25 és un graf de tipus pla. Més endavant es documenta aquests tipus de grafs.

7.1.3 Exportació dels grafs

Una vegada s'han creat els grafs s'exporten al llenguatge DOT del que en fa ús el programari interactiu extern Graphviz. L'exportació es fa mitjançant una crida conjunta a la llibreria NetworkX i al seu submòdul PyDot.

La funció nx.nx_pydot.write_dot(graf:Graph, string:nom_del_graf_amb_extensió_dot), dibuixa els grafs en el format DOT de Graphviz.

7.1.3.1 Fitxers DOT

Els fitxers DOT es composen d'un llenguatge de descripció de grafs en text pla. Normalment compten amb les extensions de fitxer gv o dot. I per defecte, assumeixen la codificació de caràcters UTF-8.

El llenguatge DOT pot descriure grafs dirigits i grafs no dirigits. Els grafs dirigits normalment s'utilitzen per mostrar diagrames de flux i arbres de dependències. El llenguatge és compatible amb comentaris d'estil: / * * / i //.

Exemple 12. A la figura 26 es poden observar dos fitxers en format DOT que representen un graf no dirigit (a la part esquerra) i un graf dirigit (a la part dreta) de la figura 24.

```
strict graph "" {
A;
B;
C;
D;
E;
F;
H;
G;
A -- B;
A -- C;
B -- D;
B -- H;
B -- G;
B -- F;
C -- E;
C -- G;
E -- F;
}
```

```
strict digraph "" {
A;
B;
C;
D;
E;
F;
H;
G;
A -> B;
A -> C;
B -> H;
B -> G;
B -> F;
C -> G;
D -> B;
E -> C;
E -> F;
}
```

Figura 26. Graf no dirigit i dirigit en format DOT de la figura 24.

Pels grafs no dirigits s'utilitza el doble guió (-) per a mostrar les arestes entre els nodes. I pels grafs dirigits s'utilitza la fletxa (->).

També es poden aplicar diferents atributs en els grafs, nodes i arestes. Aquests atributs poden controlar aspectes com ara el color, la forma i els estils de les línies de les arestes. Els atributs es col·loquen entre claudàtors després d'una declaració i abans del punt i coma (que és opcional).

La figura 27 mostra la gramàtica abstracta que defineix el llenguatge DOT. Els caràcters literals es donen entre cometes simples. Els parèntesis indiquen una agrupació. Els claudàtors tanquen elements opcionals i les barres verticals separen les alternatives que podem utilitzar.

```
graph : [ strict ] (graph | digraph) [ ID ] '{' stmt_list '}'
stmt_list : [ stmt [ ';' ] stmt_list ]
stmt : node_stmt
     | edge_stmt
     | attr_stmt
     | ID '=' ID
     | subgraph
attr_stmt : (graph | node | edge) attr_list
attr_list : '[' [ a_list ] ']' [ attr_list ]
a_list : ID '=' ID [ (';' | ',') ] [ a_list ]
edge_stmt : (node_id | subgraph) edgeRHS [ attr_list ]
edgeRHS : edgeop (node_id | subgraph) [ edgeRHS ]
node_stmt : node_id [ attr_list ]
node_id : ID [ port ]
port : ':' ID [ ':' compass_pt ]
     | ':' compass_pt
subgraph : [ subgraph [ ID ] ] '{' stmt_list '}'
compass_pt : (n | ne | e | se | s | sw | w | nw | c | _)
```

Figura 27. Gramàtica del llenguatge DOT.

Les paraules node, edge, graph, digraph, subgraph i strict no són sensibles a les majúscules. Les cadenes de caràcters entre cometes dobles es poden concatenar utilitzant l'operador '+'.

El nom del graf o també anomenat ID, pot estar buit ("") o incloure:

- Qualsevol cadena de caràcters [a-zA-Z\200-\377], caràcters connectats per ('_') o dígits [0-9]. Un dígit però no pot estar mai a l'inici d'una cadena de caràcters.
- Un nombre tal que $[-]^{?}(.[0-9]^{+} \mid [0-9]^{+}(.[0-9]^{*})^{?})$.
- Qualsevol cadena de caràcters entre cometes dobles.
- Una cadena de caràcters en format HTML.

La propietat strict prohibeix la creació de múltiples arestes, és a dir, no pot haver-hi una aresta amb el mateix node cua i node cap. En els grafs no dirigits, no pot haver-hi més d'una aresta connectada entre els mateixos dos nodes.

Els subgrafs i els clústers es processen d'una manera especial en els fitxers DOT. Per exemple, els subgrafs juguen tres papers. En un primer paper, un subgraf pot ser emprat per a representar l'estructura d'un graf, que indica quins nodes i arestes han d'estar agrupats junts. Aquest és el paper habitual dels subgrafs i en general s'especifica la informació semàntica sobre els components del graf. En un segon paper, un subgraf pot proporcionar un context per a l'establiment d'atributs. Per exemple, un subgraf podria especificar que el blau és el color per defecte per a tots els nodes definits en ell mateix. I el tercer paper dels subgrafs involucra directament com serà exposat un graf a certs motors de disseny.

Si el nom d'un subgraf comença amb clúster, Graphviz assenyala el subgraf com subgraf especial (o clúster). Si és compatible amb el format DOT, el motor de la disposició farà el disseny de manera que els nodes que pertanyen a l'agrupació es dibuixin junts, amb tota l'agrupació dins d'un rectangle delimitador.

Però hi ha certes restriccions amb els subgrafs i els clústers. En primer lloc, el noms d'un graf i el dels subgrafs comparteixen el mateix espai de noms. Per tant, cada subgraf ha de tenir un nom únic. I en segon lloc, tot i que els nodes poden pertànyer a qualsevol dels subgrafs, se suposa que segueixen una jerarquia estricta en forma d'arbre.

Exemple 13. A de la figura 28 es pot veure l'script que descriu l'estructura del graf de la figura 25 en format DOT. El graf té els atributs, color i estil, per cada aresta. No hi ha cicles per la propietat strict.

De les línies 2 a 7 es representen els nodes del graf. I la resta de línies representen les arestes entre els nodes del graf.

```
strict graph "" {
a;
c;
b;
f;
d;
e;
a -- c  [color=black, style=dashed];
a -- b  [color=black, style=solid];
a -- f  [color=black, style=solid];
a -- d  [color=black, style=solid];
a -- e  [color=black, style=solid];
c -- b  [color=black, style=dashed];
c -- f  [color=black, style=dashed];
c -- d  [color=black, style=dashed];
c -- e  [color=black, style=dashed];
b -- f  [color=black, style=dashed];
b -- d  [color=black, style=solid];
b -- e  [color=black, style=solid];
f -- d  [color=black, style=solid];
f -- e  [color=black, style=solid];
d -- e  [color=black, style=solid];
}
```

Figura 28. Script senzill en format DOT.

7.2 TIPUS DE GRAFS

Divisió dels grafs en 4 casos. Un graf pot ser pla, pla amb llindar, lineal o exponencial.

7.2.1 Graf pla

Un graf pla és equivalent a un graf de Gaifman o graf primer simple. Les parelles de valors de cada fila d'una taula SQLite, amb les que es representen els nodes i les arestes del graf, s'acoloreixen del mateix color si les parelles de valors de cada fila de la taula apareixen una o més vegades. O sigui que s'acoloreixen totes les arestes del graf on el nombre total d'equivalències sigui igual o superior a u.

En aquest tipus de grafs, hi ha dues classes d'equivalències: existeix l'aresta o no. Que existeixi una aresta en el graf vol dir que com a mínim hi ha una relació repetida entre una parella de valors a la taula i que uneix els nodes i l'aresta equivalents.

L'algoritme que s'encarrega de crear un graf pla s'anomena createPlainGraph [create_plain_graph].

L'algoritme dibuixa les arestes entre els nodes del graf que tenen el nombre d'equivalències igual o superior a u (arestes ponderades). Concretament, les arestes les pinta de color negre amb la línia contínua. Si el nombre d'equivalències és zero, les arestes es dibuixen de color negre i amb la línia discontínua (arestes no ponderades).

Exemple 14. A la figura 29 es pot observar un graf pla. Com que és un graf pla compte amb dues classes d'equivalències. La primera classe d'equivalència està representada per les arestes discontínues, que indiquen que el nombre d'equivalències entre els seus nodes és zero (arestes no ponderades). La segona classe d'equivalència està representada per les arestes no discontínues, que indiquen que el nombre d'equivalències entre els seus nodes és igual o superior a u (arestes ponderades). L'etiquetatge de les arestes no és rellevant, ja que podem diferenciar les dues classes d'equivalències segons l'estil de les línies de les arestes.

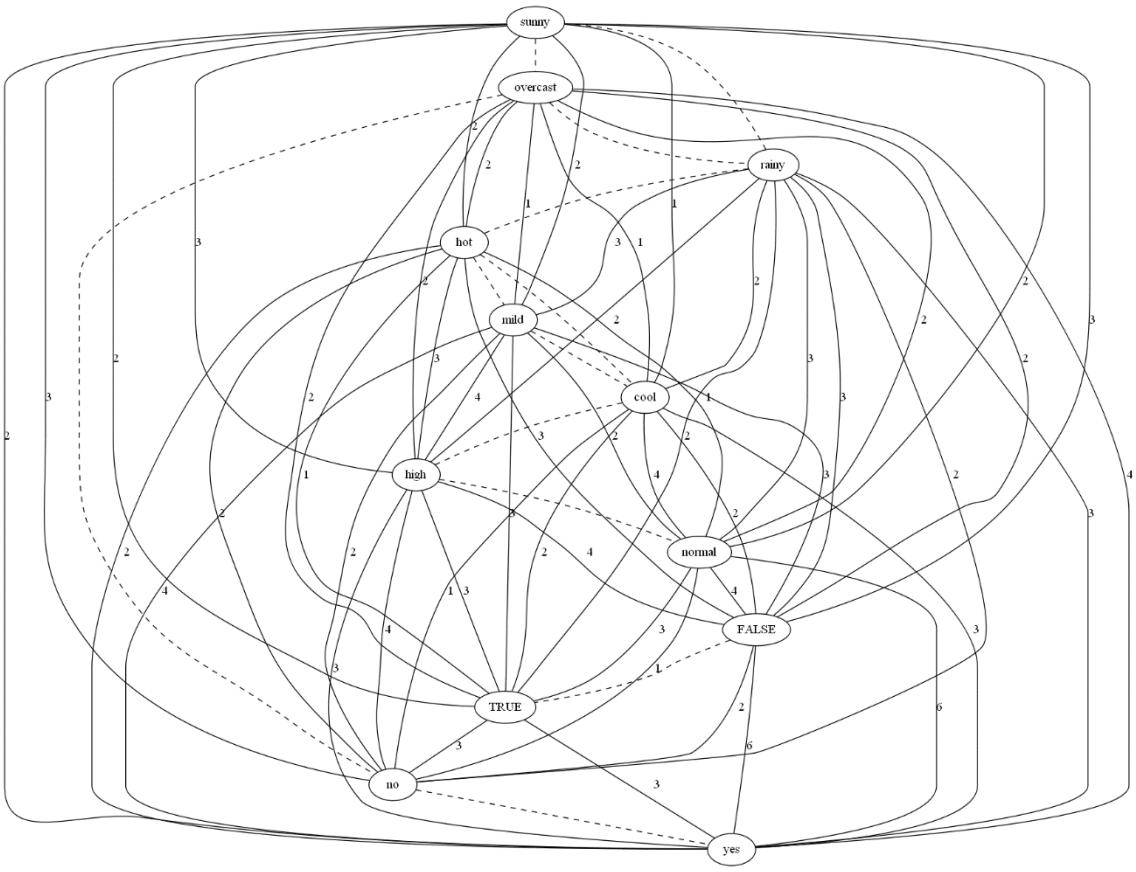


Figura 29. Graf pla.

A partir d'ara l'etiquetatge de les arestes sí que serà rellevant, ja que no només diferenciarem les classes d'equivalències segons l'estil de les línies de les arestes.

7.2.2 Graf pla amb llindar

Un graf pla amb llindar és un graf pla on cada aresta entre els nodes del graf es dibuixa amb una línia contínua si el nombre d'equivalències de l'aresta és més gran o igual a un llindar (nombre enter introduït per l'usuari). Si el llindar introduït per l'usuari és més petit que el nombre d'equivalències de l'aresta, aquesta no s'etiqueta i se li dibuixa la línia discontínuament.

En aquest tipus de grafs, també existeixen dues classes d'equivalències. La classe d'equivalència que representa les arestes etiquetades amb un valor per sota el valor del llindar. I la classe d'equivalència que representa les arestes etiquetades que tenen el mateix o un valor superior al llindar.

L'algoritme que s'encarrega de crear un graf pla tenint en compte un llindar s'anomena createPlainGraph WithThreshold [create_plain_graph_with_threshold].

L'algoritme dibuixa les arestes entre els nodes del graf que tenen el nombre d'equivalències igual o superior al llindar (arestes ponderades). Concretament, les arestes les pinta de color negre amb la línia contínua. Si el nombre d'equivalències està per sota del llindar, les arestes es dibuixen de color negre i amb la línia discontínua (arestes no ponderades).

Exemple 15. A la figura 30 es pot observar un graf pla amb el valor del llindar igual a 3. Es pot comprovar que si el nombre total de relacions d'una aresta ponderada és igual o superior a 3, la línia de l'aresta és contínua. En canvi, si el nombre total de relacions d'una aresta ponderada és inferior a 3, la línia de l'aresta és discontínua. Per tant, existeixen dues classes d'equivalències.

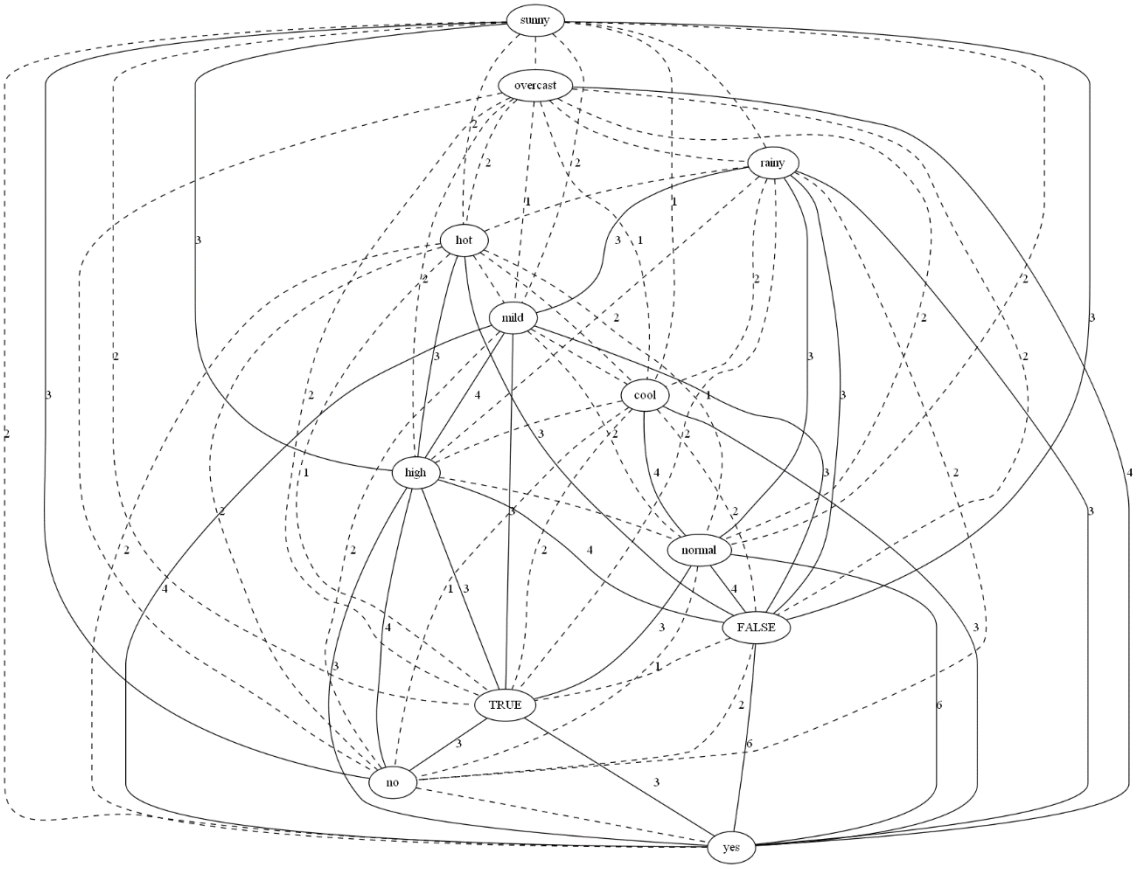


Figura 30. Graf pla amb llindar.

7.2.3 Graf lineal

Un graf lineal és un graf pla on cada aresta entre els nodes del graf es dibuixa amb una línia contínua si el nombre d'equivalències de l'aresta és més gran o igual a u. A cada nombre total d'equivalències d'una aresta diferent li correspon un únic color. Concretament, en el treball es processen 10 colors bàsics [28] que més endavant s'anomenen en una taula.

En aquest tipus de grafs, el número de classes d'equivalències és igual al número de colors que es processen. Per tant, existeixen tantes classes d'equivalències com colors hi hagi en el graf.

L'algoritme que s'encarrega de crear un graf lineal s'anomena createLinearGraph [create_linear_graph].

Inicialment, l'algoritme lineal també etiqueta les arestes amb el nombre total de relacions que existeixen a la taula SQLite adient per a cada parella de nodes. Després acoloreix cada aresta ponderada d'un color diferent segons el seu nombre total de relacions equivalents a la taula. Cada aresta amb un valor de ponderació diferent representa una única classe d'equivalència. Finalment pinta les arestes discontínues que indiquen que el nombre total de relacions d'una aresta és igual a 0.

L'algoritme dibuixa les arestes segons la ponderació de cada aresta. Cada aresta amb un nombre d'equivalències diferent es pinta d'un color diferent seguint la nomenclatura de la taula següent:

Nombre d'equivalències	Classe d'equivalència
0	Negre/Línia discontínua
1	Negre/Línia contínua
2	Turquesa
3	Verd
4	Magenta
5	Taronja
6	Blau
7	Vermell
8	Groc
9	Marró
>= 10	Gris

Taula 9. Taula d'equivalències.

Exemple 16. A la figura 31 es pot observar un graf lineal. Cada classe d'equivalència està diferentment representada per un color. El nombre total de classes d'equivalències d'aquest graf és deu. Les classes que representen: 0, 1, 2, 3, 4, 5, 6, 7, 8 i 12. Per tant, segons la taula d'equivalències anterior, els colors que s'han d'utilitzat per a acolorir les arestes són nou colors diferents.

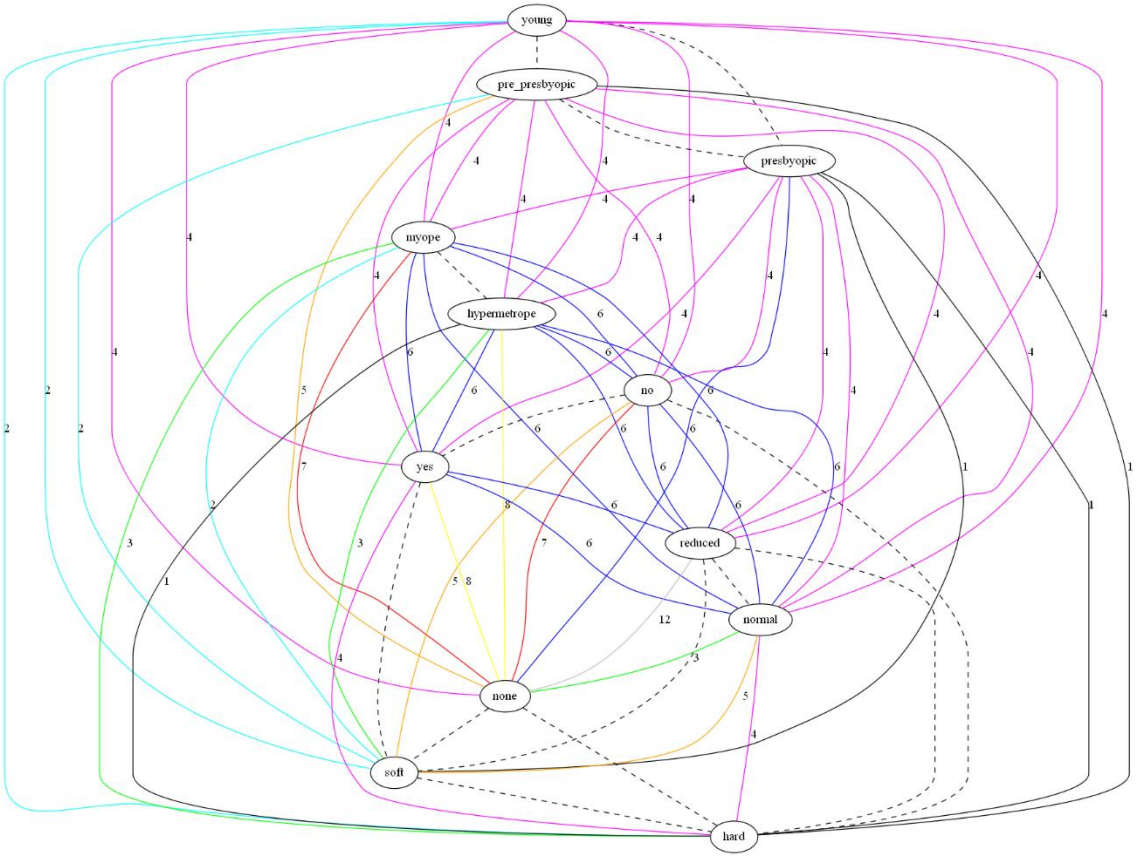


Figura 31. Graf lineal.

7.2.3 Graf exponencial

Un graf exponencial és un graf lineal on diferents classes d'equivalències s'acoloreixen del mateix color. Cada color pot representar a més d'una classe d'equivalència i comprèn un interval exponencial de classes d'equivalències. El rang de l'interval exponencial va de 2^0 fins a 2^9. Per tant, en aquest tipus de grafs, les classes d'equivalències s'agrupen segons la sèrie de potències del dos anterior.

També es processen 10 colors bàsics i cada aresta té el mateix número total d'equivalències que en el graf lineal. Ja que el graf exponencial es crear a partir d'un graf lineal. Més endavant s'anomena la composició dels grups de colors o classes d'equivalències per a cada nombre d'equivalències de les arestes.

L'algoritme que s'encarrega de crear un graf exponencial s'anomena createExponentialGraph [create_exponential_graph].

A l'inici, l'algoritme exponencial crea un graf lineal. Aquest graf lineal ja conté les arestes etiquetades amb el nombre total de repeticions de cada parella de valors o nodes de la taula SQLite apropiada.

A continuació a partir del graf lineal creat, l'algoritme acoloreix les arestes existents amb el color de l'interval exponencial que comprenen, i que podem veure a la taula següent:

Classes d'equivalències	Color
(0, 1)	Negre i línia discontínua/línia contínua
(2, 3)	Turquesa
(4, 7)	Verd
(8, 15)	Magenta
(16, 31)	Taronja
(32, 63)	Blau
(64, 127)	Vermell
(128, 255)	Groc
(256, 511)	Marró
>= 512	Gris

Taula 10. Taula dels grups d'equivalències.

A cada interval exponencial li correspon una classe d'equivalència diferent i el que compren les classes d'equivalències 0 i 1 simbolitza els grafs plans.

Exemple 17. A la figura 32 es pot observar un graf exponencial. El nombre de classes d'equivalències del graf és vuit. Els nombres 45, 52 i 57 pertanyen a un mateix interval exponencial, segons la taula dels grups d'equivalències anterior. I per això les arestes que contenen aquests valors estan acolorides del mateix color (blau). Observem el mateix comportament amb els elements que es troben dins de l'interval exponencial (128, 255) que els hi correspon una mateixa classe d'equivalència (groc).

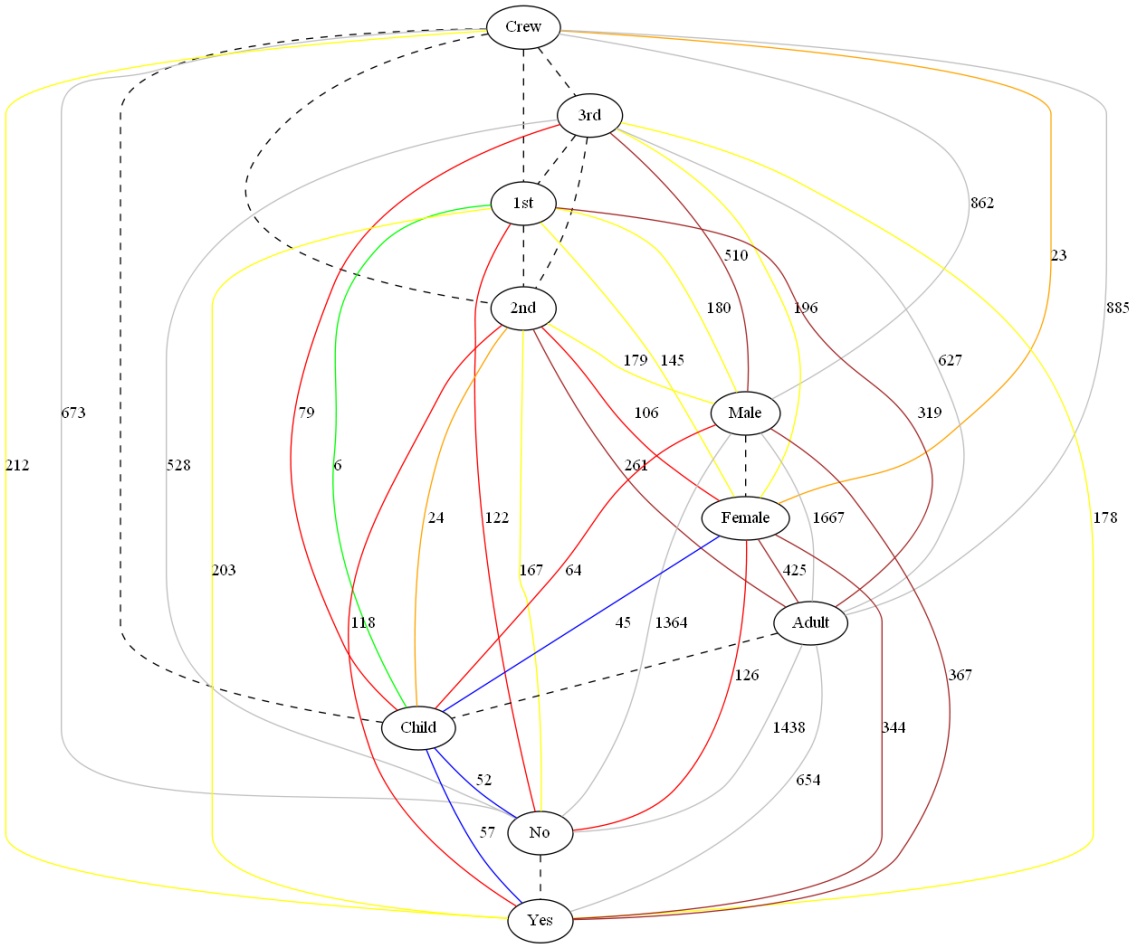


Figura 32. Graf exponencial.

Al suposar que els colors de cada aresta dels grafs formen part de la definició de les 2-estructures, podem suposar que cada graf defineix una 2-estructura.

8. DESCOMPOSICIÓ DEL GRAFS

Les descomposicions són d'especial interès dins de les matemàtiques i les seves aplicacions des de que proporcionen eines per a la divisió de problemes complexos en problemes més petits que són més fàcils de solucionar. Les descomposicions són un cas especial del principi *"divideix i venceràs"* [29].

La idea amb la que es basa la descomposició dels grafs consisteix en trobar subconjunts en cada graf, anomenats clans, en els quals els elements continguts en cada clan es relacionen de la mateixa manera amb tots aquells elements fora del clan. En aquest tipus de descomposició s'extreuen els clans primers dels grafs, un tipus especial de clans. A partir dels clans primers es generen les 2-estructures. En la construcció de les 2-estructures s'indiquen les classes d'equivalències entre els clans primers.

L'algoritme que processa la descomposició dels grafs en clans s'anomena decomposition [decomposition]. Aquest algoritme primer cerca tots els clans d'un tipus de graf i després a partir d'ells els clans trivials i els clans primers (generalització de clans). En els propers apartats s'explica detalladament tots els tipus de clans que s'utilitzen en el treball.

La descomposició dels grafs en clans, procés que també és conegut com la descomposició modular o la descomposició de substitució, és un exemple d'una descomposició estretament relacionada amb la descomposició per a quocients d'àlgebra.

8.2 CLANS

La noció tècnica més important en aquest treball és la definició de clan i es defineix formalment a continuació.

Un subconjunt $X \subseteq D_g$ d'un graf g és un clan si cada node $y \notin X$ "veu" els nodes de X de la mateixa manera i cada dos nodes $x_1, x_2 \in X$ "veuen" cada node $y \notin X$ de la mateixa manera per a tot $x_1, x_2 \in X$ i $y \notin X$:

$$(y, x_1)\ R_g\ (y, x_2) \text{ i } (x_1, y)\ R_g\ (x_2, y)$$

Exemple 18. A la figura 33 s'han encerclat els clans X = {A, B} i Y = {C, D} del graf G. X i Y són clans perquè la resta de nodes del graf no els distingeixen. Ja que cada node extern als clans està relacionat amb ells amb la mateixa classe d'equivalència. O sigui que, que cada node extern a un clan està unit a aquest amb la mateixa classe d'equivalència.

A la figura 33, el conjunt de nodes {A, C} no és un clan perquè els nodes F, E i D distingeixen A i C: R (F, A) ≠ R (F, C), R (E, A) ≠ R (E, C) i R (D, A) ≠ R (D, C), no les uneix la mateixa classe d'equivalència. El conjunt de nodes {A, B} és un clan perquè els nodes C, D, E i F no distingeixen A i B. R (C, A) ≠ R (C, B) les uneix la mateixa classe d'equivalència. Amb la resta de nodes (D, E i F) succeeix el mateix.

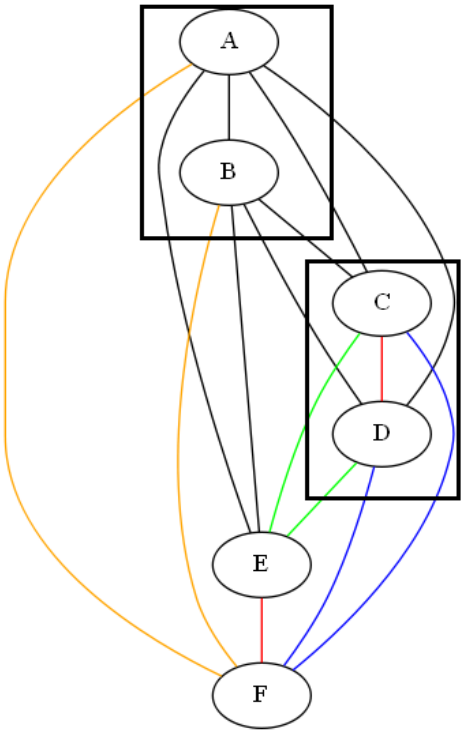


Figura 33. **Clans X i Y de l'exemple 18.**

Les classes d'equivalències que existeixen a la figura 33 són cinc: negra, taronja, verda, vermella i blava. S'han omès els nombres d'equivalència de les arestes per claredat.

L'algoritme que conclou que un clan o sigui o no s'anomena isClan [is_clan].

8.3 CLANS TRIVIALS

Per a qualsevol graf, els subconjunts únics amb el format {x}, $x \in D$ i els que contenen tots els nodes del graf són clans. Concretament s'anomenen clans trivials. Un clan no trivial és un clan X de tal manera que $|X| \geq 2$ (el nombre d'elements del clan és igual o superior a dos). Els clans trivials sempre són considerats clans primers. Per tant, tots els clans trivials també seran clans primers. Els clans primers es defineixen en el proper apartat.

Exemple 19. Considerant la figura 33 de l'exemple 18, els clans X i Y no són clans trivials del graf. Perquè són clans on |X| i |Y| són iguals a 2. En canvi, els subconjunts {'A'}, {'B'}, {'C'}, {'D'}, {'E'}, {'F'} i {'C', 'A', 'F', 'B', 'D', 'E'}, segons la definició de clan trivial, sí que ho són. Ja que representen tots els subconjunts formats per un element i el subconjunt que inclou tots els elements del graf (sense elements repetits).

Recopilant els dos apartats anteriors, tots els subconjunts que formen un clan a la figura 33 són: {'A'}, {'B'}, {'C'}, {'D'}, {'E'}, {'F'}, {'A', 'B'}, {'C', 'D'} i {'C', 'A', 'F', 'B', 'D', 'E'}.

8.4 CLANS PRIMERS

Els clans primers són molt importants en aquest treball ja que les 2-estructures es formen a partir d'ells. I un clan és primer si no compleix la propietat *d'overlapping.* Aquesta propietat defineix els clans que tenen alguns elements en comú per superposició.

Si A i B són clans d'un graf llavors:

A i B estan superposats si $A \cap B \neq 0$, $(A \cap B) \subset A$ i $(A \cap B) \subset B$ (propietat *d'overlapping*). Per tant, els clans A i B no seran clans primers. Si no compleixen la propietat *d'overlapping*, els clans A i B seran clans primers. A i B formen part d'un altre clan primer si els elements que els formen són clans primers i ells mateixos són clans primers d'un altre clan primer.

Seguint amb la figura 33 de l'exemple 18, els clans {'A'}, {'B'}, {'C'}, {'D'}, {'E'}, {'F'} i {'C', 'A', 'F', 'B', 'D', 'E'} són clans primers perquè també són clans trivials. {'A', 'B'} és un clan primer ja que {'A', 'B'} ∩ {'C', 'A', 'F', 'B', 'D', 'E'} ≠ 0, {'A', 'B'} ∩ {'C', 'A', 'F', 'B', 'D', 'E'} ⊂ {'A', 'B'} i {'A', 'B'} ∩ {'C', 'A', 'F', 'B', 'D', 'E'} ⊂ {'C', 'A', 'F', 'B', 'D', 'E'}. A més {'A',

'B'} està format per dos clans primers {'A'} i {'B'} i és subconjunt d'un altre clan primer, {'C', 'A', 'F', 'B', 'D', 'E'}.

L'algoritme que conclou que un clan sigui primer o no s'anomena primalClans [primal_clans].

8.5 CLANS MÉS FREQÜENTS

Un cas especial del treball són els clans més freqüents.

L'algoritme Apriori, que s'executa mitjançant el programari Apriori, és l'encarregat de trobar els clans més freqüents dels grafs i reduir una mica l'espai de cerca en el procés de descomposició d'aquests.

Exemple 20. Considerant el graf de la figura 34, els clans més freqüents són: {'young'}, {'pre_presbyopic'}, {'presbyopic'}, {'none', 'presbyopic'}, {'myope'}, {'no'}, {'no', 'myope'}, {'reduced'}, {'reduced', 'myope'}, {'none', 'reduced'}, {'reduced', 'myope', 'none'}, {'normal'}, {'myope', 'normal'}, {'yes'}, {'myope', 'yes'}, {'none'}, {'none', 'myope'}, {'reduced', 'no'}, {'reduced', 'no', 'none'}, {'no', 'normal'}, {'hypermetrope', 'no'}, {'none', 'no'}, {'reduced', 'yes'}, {'none', 'reduced', 'yes'}, {'hypermetrope'}, {'hypermetrope', 'reduced'}, {'none', 'hypermetrope', 'reduced'}, {'yes', 'normal'}, {'hypermetrope', 'normal'}, {'hypermetrope', 'yes'}, {'none', 'yes'} i {'none', 'hypermetrope'}, {'none', 'reduced', 'presbyopic', 'normal', 'myope', 'no', 'young', 'yes', 'hypermetrope', 'pre_presbyopic'}. El programari Apriori els genera automàticament i amb el paquet de software desenvolupat es cerquen els clans trivials i primers.

Així doncs, els clans trivials més freqüents són els clans anteriors amb una longitud igual a u: {'young'}, {'pre_presbyopic'}, {'presbyopic'}, {'myope'}, {'no'}, {'reduced'}, {'normal'}, {'yes'}, {'none'}, {'hypermetrope'}; i el clan que conté tots els elements o nodes del graf: {'none', 'reduced', 'presbyopic', 'normal', 'myope', 'no', 'young', 'yes', 'hypermetrope', 'pre_presbyopic'}. Tots ells també seran clans primers.

Finalment però, a la figura 34 no trobem més clans primers. Els clans primers trobats en aquest graf només contenen els clans trivials.

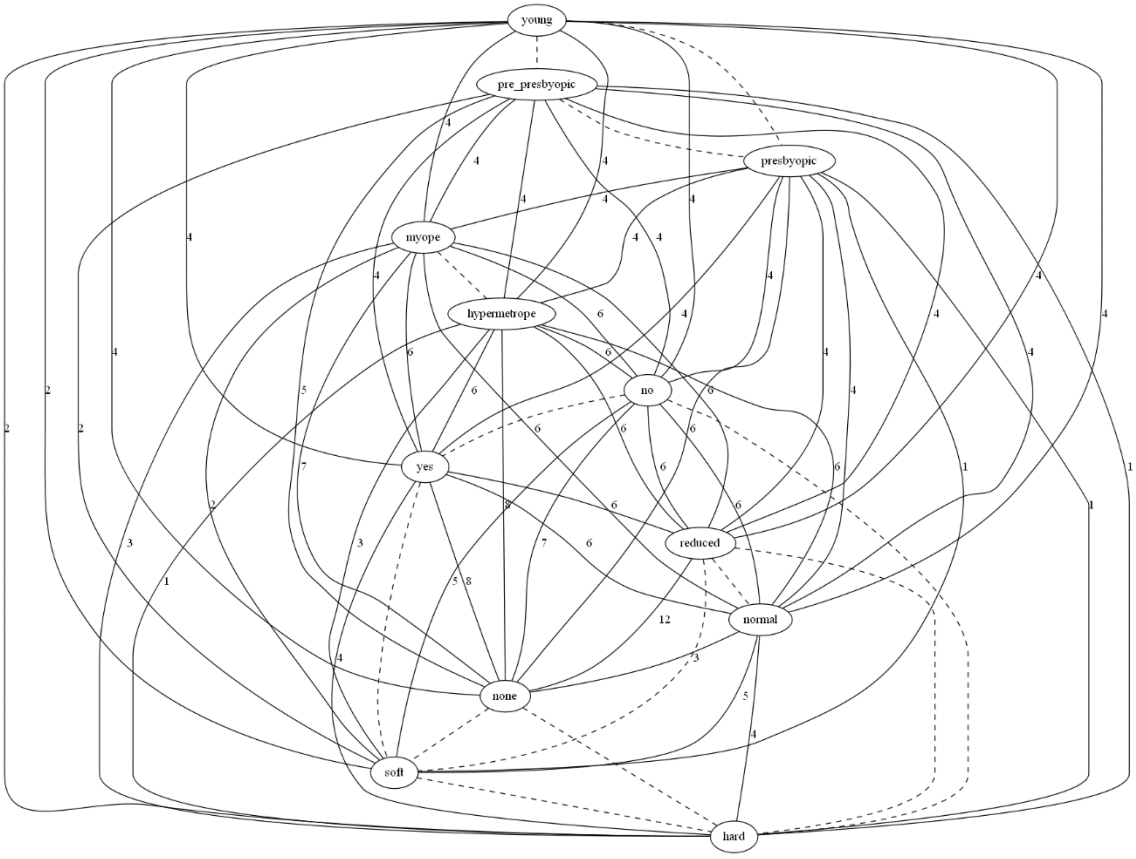


Figura 34. Graf pla de l'exemple 20.

Quan es vol descomposar un graf amb una grandària considerable, és recomanable utilitzar els clans més freqüents per a realitzar el procés de descomposició en un temps més raonable.

9. 2-ESTRUCTURES

La finalitat d'aquest treball es desenvolupar la teoria de la 2-estructures, i en particular es vol demostrar que cada 2-estructura pot ser construïda a partir de la descomposició d'un graf en clans primers. En el procés de descomposició resultant s'obté una 2-estructura a través d'una representació jeràrquica en forma d'arbre. En aquest arbre cada estructura reprodueix un clan primer o un element d'aquest. Òbviament, si tots els grafs són complets, les 2-estructures també ho seran.

Concretament, una 2-estructura és una seqüència de valors agrupats, formada per un subconjunt finit D, anomenat domini, i una relació d'equivalència $R \subseteq E(D) \times E(D)$ en el conjunt $E(D) = \{(u, v) \mid u \in D, v \in D, u \neq v\}$ de les seves arestes. D'aquesta manera una 2-estructura es pot definir com un graf G dirigit i complet, on el domini D representaria els nodes, i les classes d'equivalències R les arestes acolorides del graf G. Formalment:

$$G = (D, R)$$

Cada tipus de graf és equivalent a un tipus de 2-estructura. Per exemple, una 2-estructura lineal serà el resultat de la descomposició d'un graf lineal. Com en el proper exemple que és la 2-estructura resultant de la descomposició del graf lineal de la figura 33.

Exemple 21. A la figura 35 es pot veure la representació gràfica d'una 2-estructura lineal. En ella, cada estructura rectangular o clúster representa un clan primer no trivial del graf lineal. En cada clúster també podem observar les classes d'equivalència entre els elements del clan primer corresponent al clúster. Les arestes internes dels clústers serveixen per a diferenciar les classes d'equivalències que pertanyen a cada element del clan primer corresponent. Les arestes externes dels clústers serveixen per indicar les connexions entre els clústers mateixos. Les connexions entre els clústers indiquen els clans primers que formen part d'un altre clúster (o clan primer). Els clans trivials de longitud igual a u es troben a les fulles del subarbre al que queda connectat cada clan primer representat.

Més detalladament, a la 2-estructura de la figura 35 hi ha cinc classes d'equivalències diferents: negra, blava, vermella, groga i verda. Hi ha tres clústers definits pels clans primers: {'A', 'B'}, {'C', 'D'} i {'C', 'A', 'F', 'B', 'D', 'E'}. I els clans trivials: {'A'}, {'B'}, {'C'},

{'D'}, {'E'} i {'F'}, que es troben a les fulles del subarbre. Les arestes externes dels clústers ens confirmen que el clan primer {'C', 'A', 'F', 'B', 'D', 'E'}, es composa dels dos clans primers {'A', 'B'} i {'C', 'D'}.

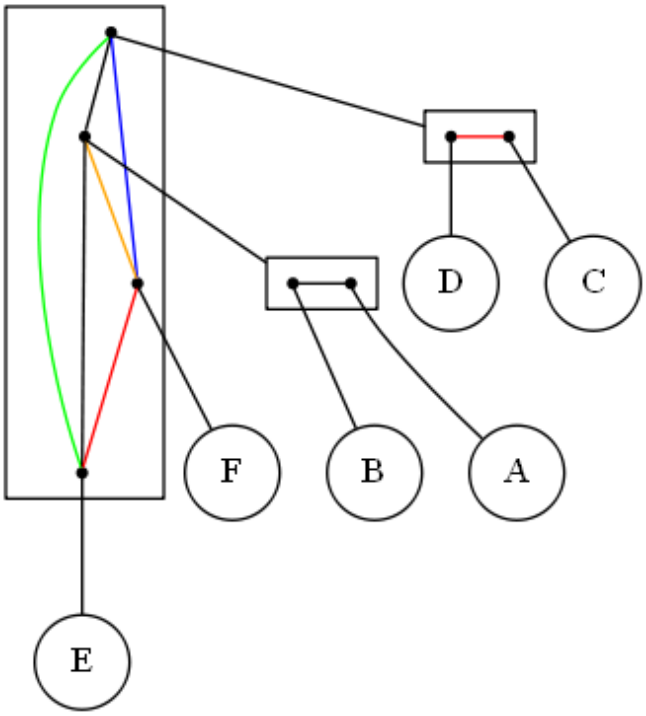


Figura 35. 2-estructura lineal de l'exemple 21.

Les 2-estructures també es poden descomposar en altres 2-estructures, com s'ha comentat anteriorment. La divisió d'una 2-estructura en una o més d'una 2-estructura ha de contenir com a mínim alguna de les tres subclasses especials de 2-estructures bàsiques que existeixen: completes, lineals i primitives.

- Una 2-estructura és completa quan només té definida una classe d'equivalència. Cada 2-estructura completa pot ser primitiva si $|D| \leq 2$ (el nombre de nodes que conté és igual o inferior a dos)
- Una 2-estructura és lineal quan té definides més d'una classe d'equivalència i les arestes que representen les classes d'equivalències estableixen un ordre total sobre els elements del domini D (nodes de la 2-estructura).

- Una 2-estructura és primitiva quan no pot descomposar-se en altres 2-estructures. També diem que una 2-estructura és primitiva si només conté clans trivials i que és veritablement primitiva si almenys conté 2 nodes.

A la figura 35 tenim un exemple de la descomposició d'una 2-estructura en altres 2-estructures. El clúster més gran es descomposa en dues 2-estructures primitives.

9.1 CREACIÓ DE LES 2-ESTRUCTURES

Com s'ha anomenat en la creació dels grafs, el format DOT defineix un graf. Però no disposa d'eines per a la visualització gràfica i per això s'utilitza el programari interactiu extern Graphviz. En la creació de les 2-estructures es segueix el mateix procediment i s'utilitza el format DOT per definir les 2-estructures. Ja que com hem comentat moltes vegades en aquest document, un graf complet i dirigit amb les arestes acolorides es equivalent a una 2-estructura. Per la construcció de les 2-estructures en beneficiem d'aquesta definició per a crear-les.

L'algoritme que construeix les 2-estructures mitjançant el llenguatge DOT s'anomena createGraphvizStructure [create_Graphviz_Structure].

Exemple 22. A la figura 36 es pot veure un exemple senzill de l'script que descriu la 2-estructura de la figura 37.

```
strict digraph "linear_2-structure" {
compound=true;
fontname=Verdana;
fontsize=12;
newrank=true;
node [shape=circle];
d;
a;
c;
b;
subgraph cluster_dcab {
node [shape=point];
s_cb -> s_d  [arrowhead=none, color=black];
s_cb -> s_a  [arrowhead=none, color=black];
s_d -> s_a  [arrowhead=none, color=black];
}

subgraph cluster_cb {
rank=same;
node [shape=point];
s_c -> s_b  [arrowhead=none, color=cyan];
}

s_cb -> s_c  [arrowhead=none, lhead=cluster_cb];
s_d -> d  [arrowhead=none];
s_a -> a  [arrowhead=none];
s_c -> c  [arrowhead=none];
s_b -> b  [arrowhead=none];
```

Figura 36. Script senzill en format DOT de l'exemple 22.

La primera línia de l'script indica el tipus de graf i el seu nom entre cometes dobles. A més de la propietat strict, que assegura que no es repeteixin les arestes amb el mateix node cap i node cua.

De les línies 2 a 6 es declaren les característiques per defecte de la 2-estructura. En ordre les característiques són:

- Permetre arestes entre els clústers.
- El tipus de lletra.
- La mida del tipus de lletra.
- La direcció de la posició dels clústers.
- L'estil dels nodes. A l'exemple els nodes externs dels clústers es representen amb nodes i els nodes interns amb punts.

De les línies 7 a 10 s'observen els diferents clústers de la 2-estructura, els nodes i les arestes que els formen. A més de les arestes entre els clústers i les arestes de les fulles del subarbre per a cada clan trivial.

Finalment la 2-estructura equivalent a l'script de la figura 36 és:

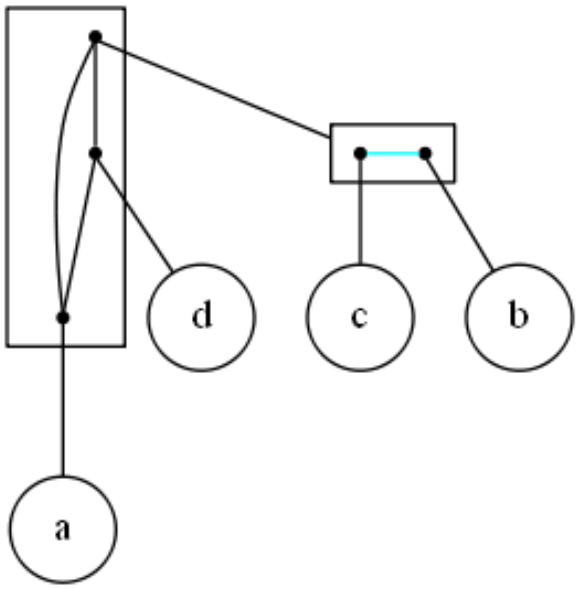


Figura 37. 2-estructura equivalent a l'script de la figura 36.

9.2.1 Limitacions

És possible que no es pugui especificar algun detall de disseny en el format DOT. Ja que en funció de les eines utilitzades durant el disseny, els usuaris han de confiar en alguns algoritmes automatitzats de traçat (potencialment amb un resultat inesperat) o que provoquen desajustaments (estructures mal col·locades).

En aquest treball hem trobat alguna dificultat en definir la direcció de la posició d'alguns clústers. A la figura 38 podem observar la limitació amb la que ens hem trobat.

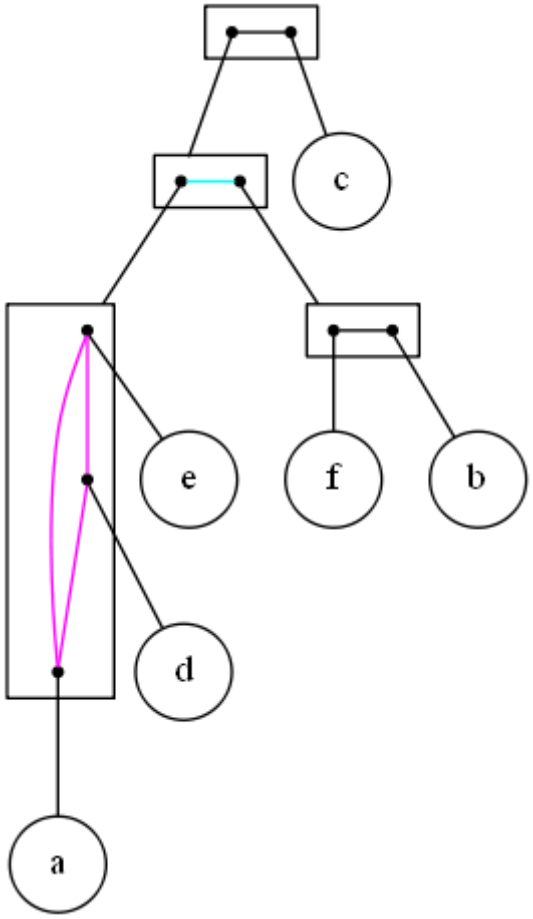


Figura 38. Limitació top-down del llenguatge DOT.

Com es pot comprovar a la figura 38, els clústers amb més de dos nodes, no estan ben alineats seguint la estratègia del tractament de la informació i el coneixement de comandes que incorpora el llenguatge DOT per defecte: *top-down* [40].

Això es deu a que els clústers no formen part del llenguatge DOT, sinó a una convenció sintàctica adherida per alguns dels motors de disseny del programari Graphviz. Seguint la alineació *top-down*, les arestes dels clústers amb més de dos nodes surten d'aquests. A la figura 39 podem veure la mateixa 2-estructura que a la figura 38 però amb tots els clústers alineats amb *top-down*. I comprovem que en el clúster de tres nodes, hi ha un aresta que sobresurt.

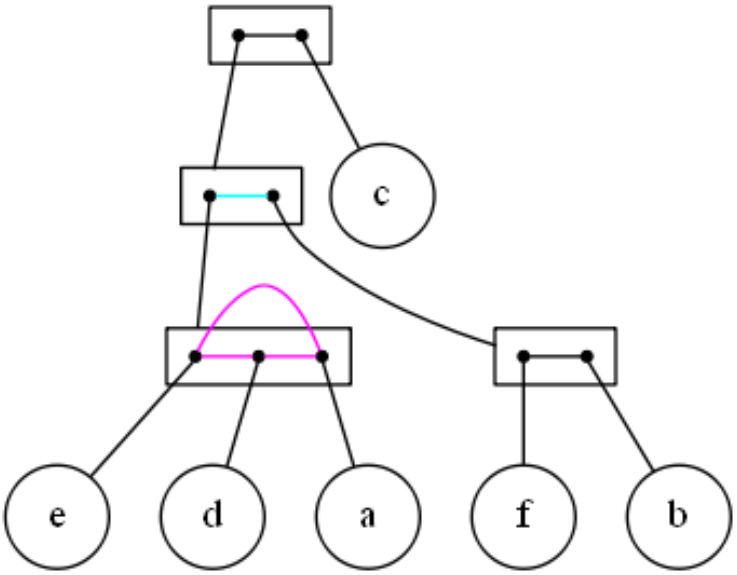


Figura 39. Sense limitació top-down del llenguatge DOT.

10. EXEMPLE: TITANIC

Aquest és un exemple en què es mostra un procés que modela el comportament d'un usuari. El cicle del procés que es duu a terme és el següent: fitxer amb les dades relacionals d'entrada -> graf -> descomposició del graf -> 2-estructura. Quan es crea el graf i la 2-estructura es mostra la seva visualització a l'usuari.

Desglossarem el procés més detalladament en els apartats següents.

10.1 DADES D'ENTRADA

Les dades relacionals que conté el fitxer TXT que utilitzarem per aquest l'exemple es converteixen en una taula SQLite de 2201 files per quatre columnes (2201 dels 2224 passatgers que anaven a bord del Titanic). Mostrem les primeres vint files de la taula a continuació per a tenir una visió de la estructuració i el tipus de les dades relacionals amb les que treballarem l'exemple.

	Class	Sex	Age	Survived
1	Crew	Male	Adult	No
2	Crew	Male	Adult	No
3	Crew	Male	Adult	Yes
4	3rd	Female	Adult	Yes
5	3rd	Male	Adult	No
6	Crew	Male	Adult	Yes
7	Crew	Male	Adult	No
8	Crew	Male	Adult	Yes
9	Crew	Male	Adult	No
10	3rd	Male	Adult	No
11	Crew	Male	Adult	No
12	1st	Male	Adult	Yes
13	2nd	Female	Adult	Yes
14	3rd	Male	Adult	No
15	1st	Female	Adult	Yes
16	Crew	Male	Adult	No
17	Crew	Male	Adult	No
18	Crew	Male	Adult	No
19	3rd	Male	Adult	No
20	3rd	Female	Adult	Yes

Taula 10. Taula de l'exemple Titanic.

10.2 TIPUS DE GRAF

El tipus de graf que s'ha volgut generar a partir de les dades de la taula de l'apartat anterior és un graf pla (figura 42). Hem de recordar que en aquests tipus de grafs existeixen dues classes d'equivalències.

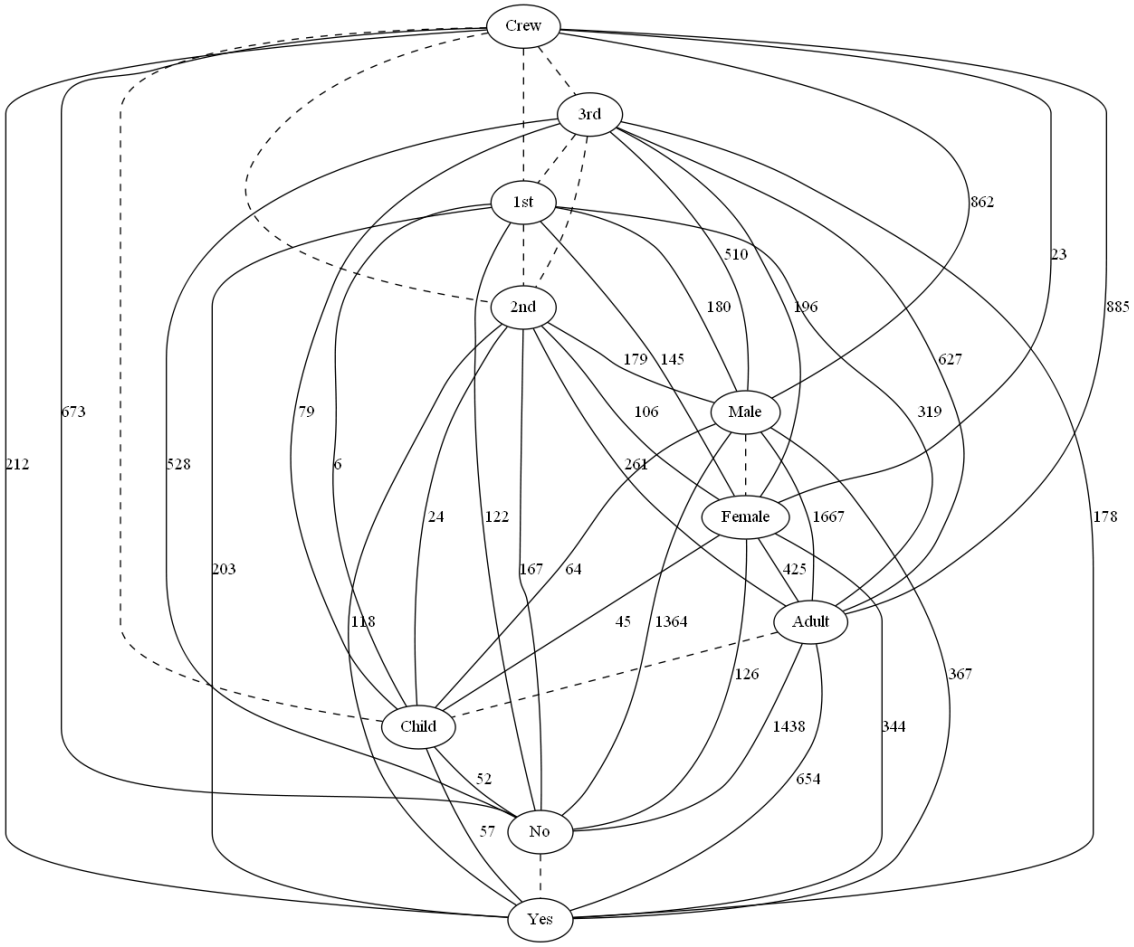


Figura 42. Graf pla de l'exemple Titanic.

Observem doncs, les dues classes d'equivalències del graf amb dos exemples. Per un costat, l'aresta (línia contínua) entre la parella de nodes 3rd i Male ens indica que hi havia 510 passatgers que eren homes i viatjaven en 3a classe. Per l'altre costat, l'aresta (línia discontínua) entre la parella de nodes 3rd i Crew ens indica que la tripulació no viatjava en 3a classe.

Visualment, la disposició de les dades és més intuïtiva perquè només en mirar el graf ja podem veure dades relacionades o no entre sí. També ens permet veure el nombre de vegades que es repeteixen cada parella de valors dins d'aquesta estructura de dades molt fàcilment. A primera vista podem saber que el nombre de supervivents adults després de la col·lisió amb l'iceberg va ser de 654. O que hi havia 23 dones a la tripulació.

10.3 DESCOMPOSICIÓ DEL GRAF

En aquest apartat es duu a terme la descomposició del graf pla de la figura 42 en clans primers per a la construcció dels nodes, de les arestes i dels clústers de la 2-estructura.

Primer es troben tots els clans del graf, que són:

{'Crew'}, {'3rd'}, {'1st'}, {'2nd'}, {'Male'}, {'Female'}, {'Adult'}, {'Child'}, {'No'}, {'Yes'}, {'3rd', '1st'}, {'2nd', '3rd'}, {'2nd', '1st'}, {'Female', 'Male'}, {'Yes', 'No'}, {'2nd', '3rd', '1st'}, {'Female', 'No', 'Male', 'Yes'}, {'Adult', 'Crew', '2nd', 'Child', '3rd', '1st'}, {'Adult', 'Crew', 'Female', '2nd', 'Child', 'Male', '3rd', '1st'}, {'Adult', 'Crew', 'Yes', '2nd', 'Child', 'No', '3rd', '1st'} i {'Adult', 'Crew', 'Female', 'Yes', '2nd', 'Child', 'No', 'Male', '3rd', '1st'}

Per exemple si ens fixem en la figura 42 anterior, el subconjunt {'Female', 'Male'} és un clan del graf ja que la resta de nodes no el distingeixen (les arestes cap als elements 'Female' i 'Male' de la resta de nodes són del mateix color i estil de línia). I els elements {'Child', 'Second'} no formen un clan perquè {'Adult'} el distingiria: R ('Adult', 'Child') ≠ R ('Adult', 'Second').

Després es classifiquen els clans en clans trivials i clans primers. Els clans que tenen com a longitud u i el clan que conté tots els elements de la figura 42, els clans trivials d'aquest graf, són: {'Crew'}, {'3rd'}, {'1st'}, {'2nd'}, {'Male'}, {'Female'}, {'Adult'}, {'Child'}, {'No'}, {'Yes'} i {'Adult', 'Crew', 'Female', 'Yes', '2nd', 'Child', 'No', 'Male', '3rd', '1st'}. Cada clan trivial també és un clan primer del graf. Per tant, els clans trivials anteriorment anometat són clans primers.

Els clans únicament primers del graf són: {'Female', 'Male'}, {'Yes', 'No'}, {'2nd', '3rd', '1st'} i {'Adult', 'Crew', '2nd', 'Child', '3rd', '1st'}. Si ens tornem a fixar amb la figura 42, per exemple el subconjunt {'Yes', 'No'} és un clan primer ja que {'Yes', 'No'} ∩ {'Female', 'No', 'Male', 'Yes'} ≠ 0, {'Yes', 'No'} ∩ {'Female', 'No', 'Male', 'Yes'} ⊂ {'Yes', 'No'} i {'Yes', 'No'} ∩ {'Female', 'No', 'Male', 'Yes'} ⊂ {'Female', 'No', 'Male', 'Yes'}. A més {'Yes, 'No'} està format per dos clans primers {'Yes} i {'No} i és subconjunt d'un altre clan primer, {'Female', 'No', 'Male', 'Yes'}.

Resumint, a la figura 42 hi ha 15 clans primers: 11 clans trivials ({'Crew'}, {'3rd'}, {'1st'}, {'2nd'}, {'Male'}, {'Female'}, {'Adult'}, {'Child'}, {'No'}, {'Yes'} i {'Adult', 'Crew', 'Female', 'Yes', '2nd', 'Child', 'No', 'Male', '3rd', '1st'}) i 4 clans totalment primers ({'Female', 'Male'}, {'Yes', 'No'}, {'2nd', '3rd', '1st'} i {'Adult', 'Crew', '2nd', 'Child', '3rd', '1st'}). Aquests clans s'hauran de reflectir en la 2-estructura.

Exemple 23. A la figura 43 podem observar el graf de la figura 42 amb alguns dels exemples de clans diferenciats i anomenats en els paràgrafs anteriors.

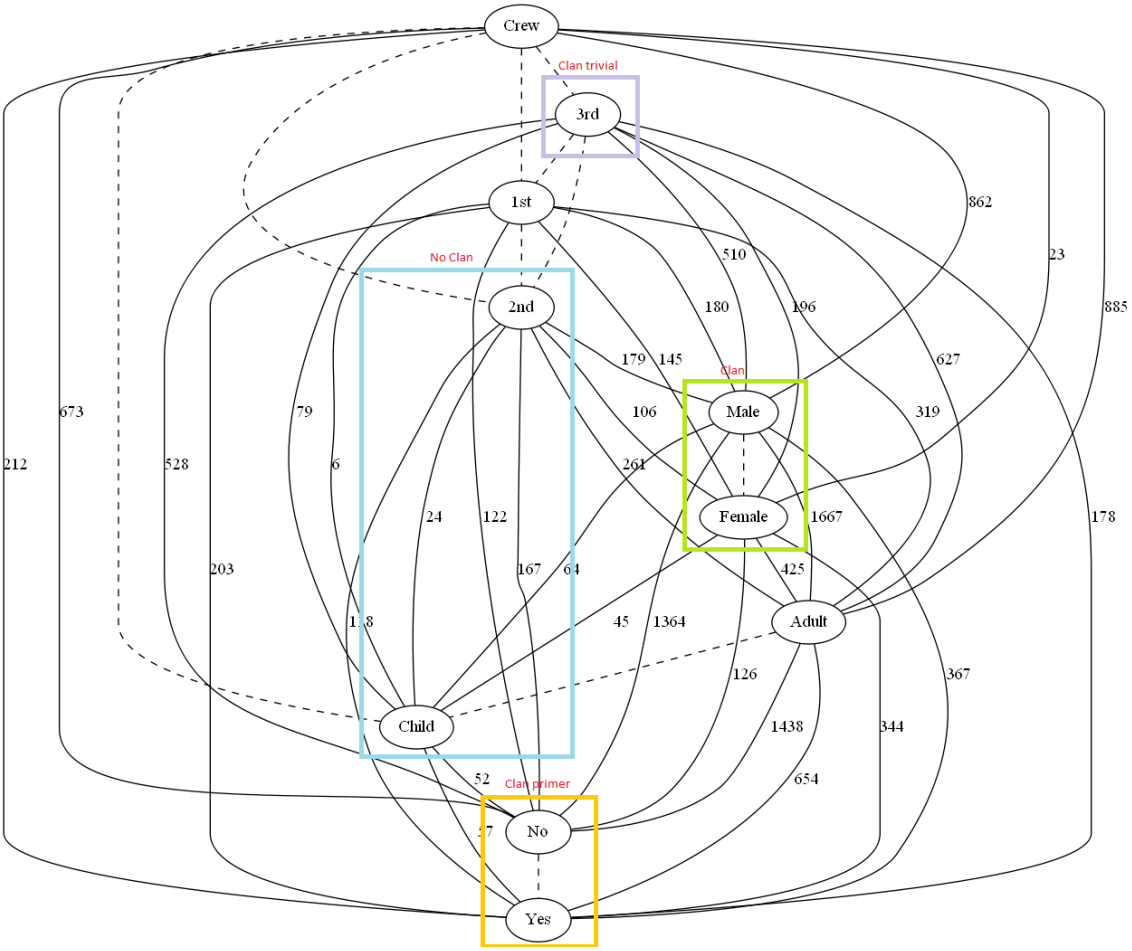


Figura 43. Representació d'alguns clans del graf de la figura 42.

10.4 2-ESTRUCTURA

Una vegada trobats tots els clans primers del graf es crea la 2-estructura amb ells. Hem de recordar que el tipus de 2-estructura serà pla, ja que el graf també ho és.

Cada clan primer amb el nombre d'elements o subconjunts igual o superior a dos formarà un clúster. Per tant, els clústers de la 2-estructura plana seran: {'Adult', 'Crew', 'Female', 'Yes', '2nd', 'Child', 'No', 'Male', '3rd', '1st'}, {'Female', 'Male'}, {'Yes', 'No'}, {'2nd', '3rd', '1st'} i {'Adult', 'Crew', '2nd', 'Child', '3rd', '1st'}. Cinc clústers per cinc clans primers. I cada element o subconjunt d'un dels clans primers anteriors amb longitud igual a u penjarà del clúster que els contingui. Del clúster que representa el clan primer {'2nd', '3rd', '1st'} penjaran els nodes {'2nd'}, {'3rd'} i {'1st'} (clans trivials), que són els tipus de classes del vaixell.

La 2-estructura de l'exemple Titanic és la figura següent:

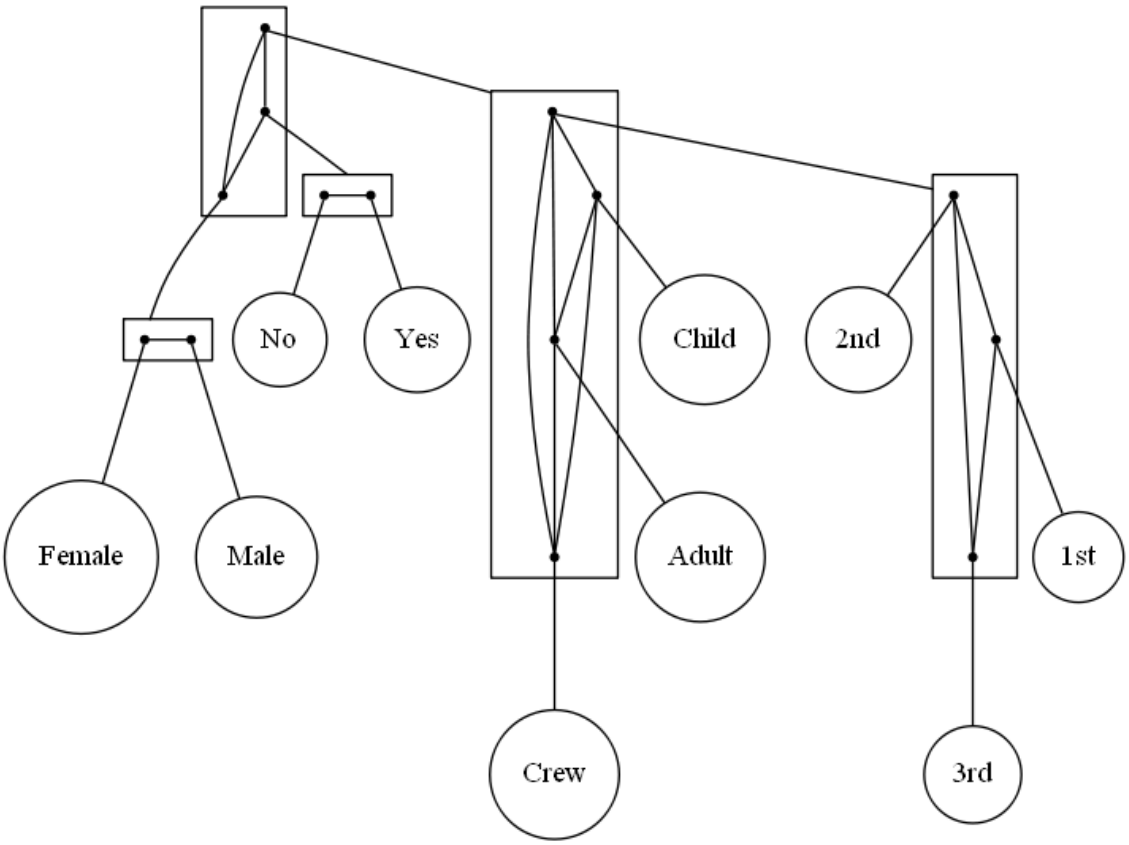


Figura 44. 2-estructura plana de l'exemple Titanic.

A la figura 44 també podem observar la propietat de factorització de les 2-estructures en altres 2-estructures més simples. Una de les 2-estructures completes (clúster amb més de dos nodes i amb una sola classe d'equivalència) es divideix en dues 2-estructures primitives.

En aquest últim pas, les dades s'han tornat a organitzar per a una millor visualització. Es podran relacionar, classificar i analitzar millor.

Com a demostració, en mirar la figura 44 sabem que existien tres tipus de classes de passatgers en el Titanic: primera, segona i tercera classe. També sabem que hi havia tres tipus de passatgers (tripulació, adults i nens) que podien estar allotjats en alguna de les tres classes. Cada tipus de passatger es pot desglossar per gènere (home o dona) i si va arribar a sobreviure al xoc amb l'iceberg d'aquell fatídic 14 d'abril de 1912.

Aquest tipus d'anàlisi de dades no es pot aconseguir d'un fitxer o d'una taula SQlite tant fàcilment.

Com a curiositat d'aquest exemple es mostra el graf pla i la 2-estructura plana de la figura 44 amb llindar igual a 1000:

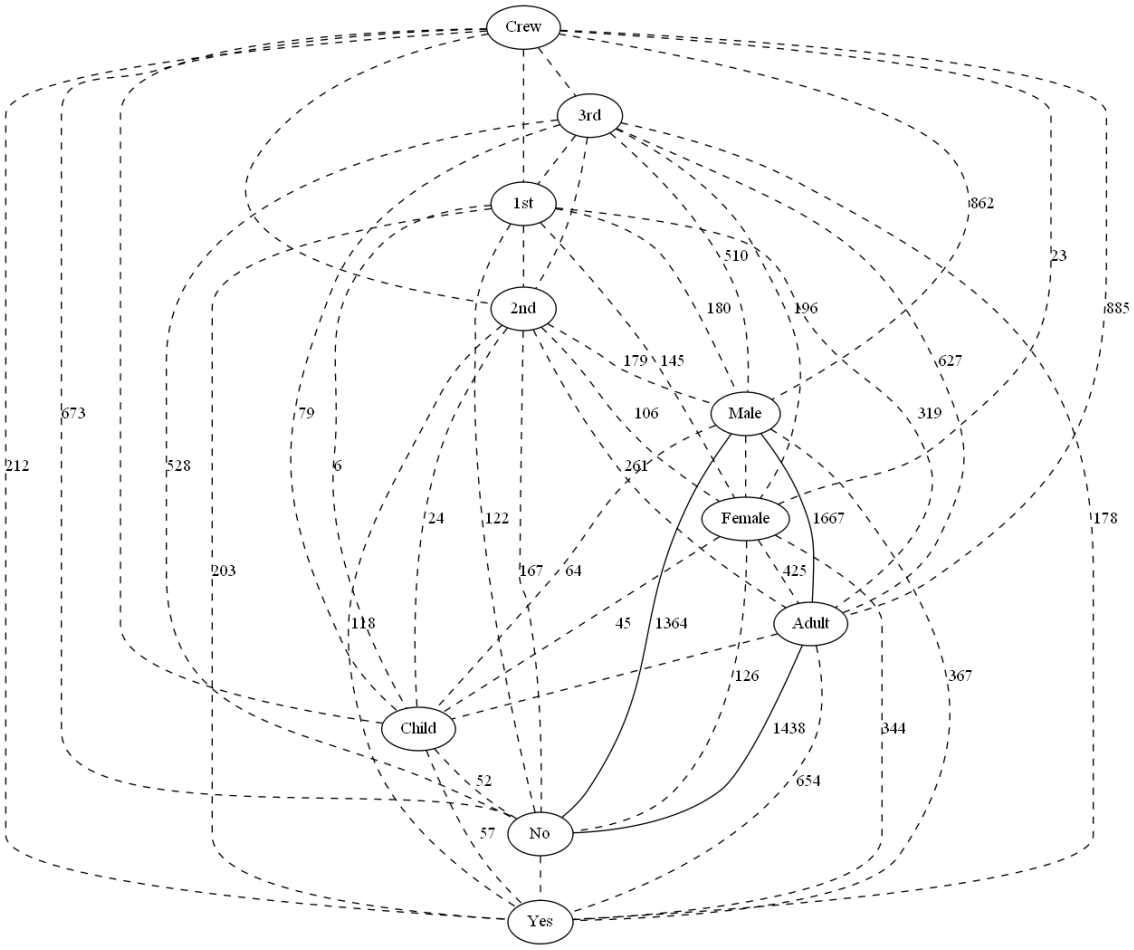


Figura 45. Graf pla amb llindar de l'exemple Titanic.

Es pot evidenciar la vella consigna marítima de *"les dones i els nens primer"* a simple vista. La majoria de passatgers que no van sobreviure van ser homes i adults. A la 2-estructrua resultant de la descomposició d'aquest graf encara és més evident.

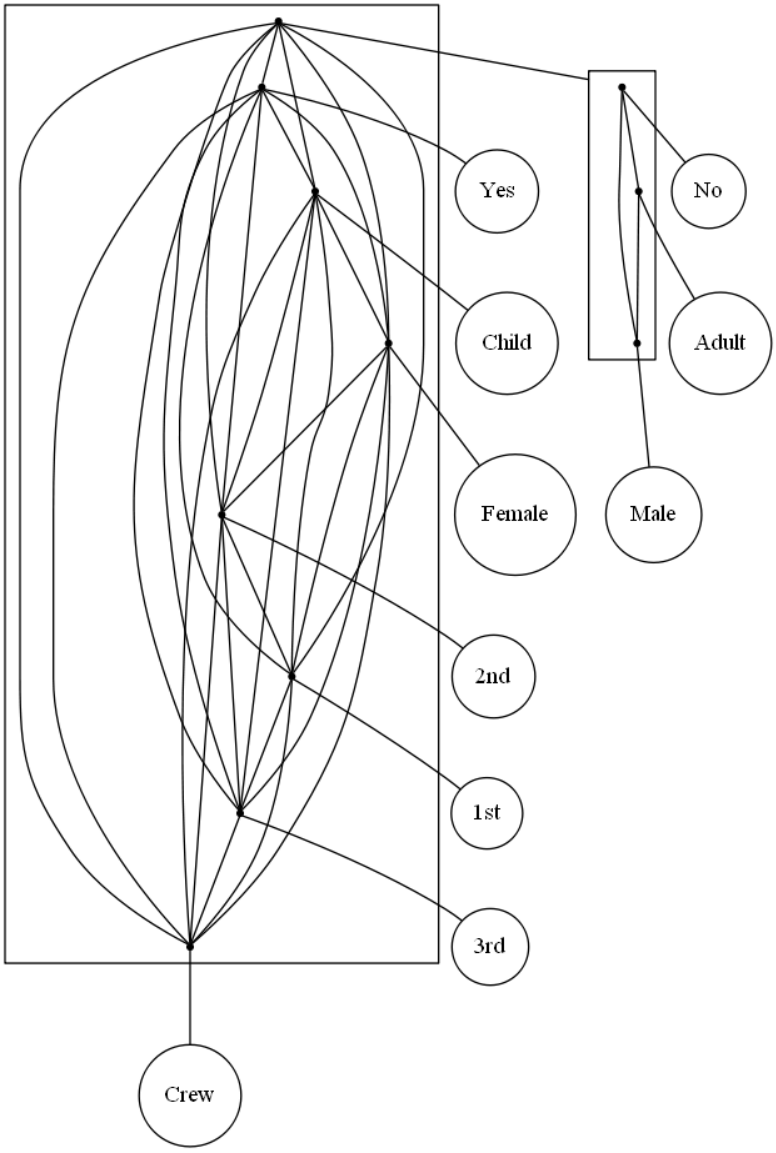


Figura 46. 2-estructura plana amb llindar de l'exemple Titanic.

11. CONCLUSIONS

En aquest capítol es presenta un resum del conjunt del treball realitzat i es descriuen les possibilitats de continuació. També s'ofereix una valoració personal sobre el desenvolupament del treball.

11.1 RESULTATS DEL TREBALL

Aquest treball va néixer amb una doble finalitat: crear una eina per a l'estudi de les 2-estructures amb Python i establir resultats prometedors que relacionin les 2-estructures amb les dades relacionals; objectius que s'han complert.

El treball realitzat es pot resumir en els següents punts:

- El disseny i la implementació d'un paquet de software que conté una biblioteca per a programes futurs que vulguin analitzar i visualitzar les estructures principals involucrades en la teoria de les 2-estructures per a usuaris finals que investiguin sobre el tema.
- La formulació d'una extensió de la teoria dels grafs que es fonamenta amb al teoria de les 2-estructures.
- Diferents algoritmes capaços de generar una gramàtica DOT a partir d'una taula SQLite, d'un fitxer de tipus ARFF o d'un fitxer de tipus TXT.

Com a resultat addicional es preveu la elaboració d'un article d'investigació.

11.2 TREBALL FUTUR

Part de l'objectiu amb el que s'ha construït el paquet de software és la seva utilització com a eina base per a investigacions futures en relació amb la teoria de les 2-estructures. Tot i així, és impossible avançar-se i enumerar totes les possibles utilitats que podria tenir. Per això, serà necessari seguir completant i adaptant el treball realitzat per ajustar-se a les necessitats dels treballs futurs.

En relació als colors que s'han utilitzat per a les classes d'equivalències es proposa la ampliació del nombre de colors per poder diferenciar moltes més classes d'equivalències.

Un altre punt interesant seria l'anàlisi en profunditat de la creació de 2-estructures a partir de diferents taules SQLite. I el seu anàlisi corresponent.

Per últim, es proposa com a següent pas natural, la millora dels algoritmes per augmentar la capacitat del tractament de les bases de dades relacionals amb major nombre de dades.

11.3 VALORACIÓ PERSONAL

La realització del treball final de grau ha estat una experiència gratificant i enriquidora.

Sobretot pel repte que suposa realitzar un treball d'investigació. I per la doble naturalesa del treball, que es composa d'una part més teòrica, una mica matemàtica, i una part més pràctica implementada en un llenguatge de programació que era bastant nou per a mi.

A la part pràctica s'ha desenvolupat el paquet de software, on he pogut aplicar molts dels coneixements de programació adquirits durant la carrera. I a la part teòrica es troba l'estudi de les 2-estructures, tema totalment desconegut, que gràcies a la seva exploració m'ha resultat molt interessant. A la part teòrica també es troba la teoria de grafs. En aquest cas però no era un tema desconegut per a mi. Que pugés treballar amb ella i ampliar-la va ser molt important ja que és una de les teories que més m'han agradat durant la meva formació acadèmica.

Aquest treball també s'ha convertit en la oportunitat d'estudiar alguns dels temes inclosos en la mineria de dades, que per problemes d'horaris no vaig poder cursar a l'assignatura de Mineria de Dades (MIDA).

Un altre dels motius a valorar molt positivament és la llibertat que he tingut, el suport i l'ajuda que he rebut per a desenvolupar el treball per part del director del treball.

BIBLIOGRAFIA

[1] The DOT Language.

http://www.graphviz.org/content/dot-language (Últim accés: 30/04/2017).

[2] Lali Barrière. *"Teoria de grafs"*. Apunts de teoria del Departament de Matemàtica Aplicada IV, 2006.

[3] A. Ehrenfeucht i G.Rozenberg. *"The Theory of 2-Structures: A Framework for Decomposition and Transformation of Graphs".* World Scientific, 1999.

[4] Wrike. https://www.wrike.com/ (Últim accés: 30/05/2017).

[5] GanttProject. http://www.ganttproject.biz/ (Últim accés: 30/04/2017).

[6] WebDAV. http://www.webdav.org/ (Últim accés: 30/04/2017).

[7] Python. https://www.python.org/ (Últim accés: 30/04/2017).

[8] sqlite3. https://docs.python.org/3/library/sqlite3.html (Últim accés: 30/04/2017).

[9] PEP 249. https://www.python.org/dev/peps/pep-0249/ (Últim accés: 30/04/2017).

[10] NetworkX. https://networkx.github.io/ (Últim accés: 30/04/2017).

[11] Llicència BSD.

https://ca.wikipedia.org/wiki/Llic%C3%A8ncia_BSD (Últim accés: 30/04/2017).

[12] itertools. https://docs.python.org/3/library/itertools.html (Últim accés: 30/04/2017).

[13] APL.

https://en.wikipedia.org/wiki/APL_(programming_language) (Últim accés: 30/04/2017).

[14] Haskell. https://www.haskell.org/ (Últim accés: 30/04/2017).

[15] SML.

http://scholarpedia.org/article/Standard_ML_language (Últim accés: 30/04/2017).

[16] PyDot.

https://networkx.github.io/documentation/networkx-1.10/reference/drawing.html (Últim accés: 30/04/2017).

[17] Graphviz. http://www.graphviz.org/ (Últim accés: 30/04/2017).

[18] PyCharm. https://www.jetbrains.com/pycharm/ (Últim accés: 30/04/2017).

[19] GitHub. https://github.com/ (Últim accés: 30/04/2017).

[20] PEP-8. https://www.python.org/dev/peps/pep-0008/ (Últim accés: 30/04/2017).

[21] CVS. http://cvs.nongnu.org/ (Últim accés: 30/04/2017).

[22] Git. https://git-scm.com/ (Últim accés: 30/04/2017).

[23] Programari Apriori. http://borgelt.net/apriori.html (Últim accés: 30/04/2017).

[24] Algoritme Apriori.

https://en.wikipedia.org/wiki/Apriori_algorithm (Últim accés: 30/04/2017).

[25] Tecnologia per a tothom. https://txt.upc.es/ (Últim accés: 30/04/2017).

[26] Weka. http://www.cs.waikato.ac.nz/~ml/ (Últim accés: 30/04/2017).

[27] Hodkinson, I. Otto, M. "*Finite conformal hypergraph covers and Gaifman cliques in finite structures"*. The Bulletin of Symbolic Logic, 2003.

[28] Graphviz color names.

http://www.graphviz.org/doc/info/colors.html (Últim accés: 30/04/2017).

[29] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. "Introduction to Algorithms". MIT Press, 2000.

[30] Màquina de Turing. https://ca.wikipedia.org/wiki/M%C3%A0quina_de_Turing (Últim accés: 30/04/2017).

[31] Python per a Windows.

https://www.python.org/downloads/windows/ (Últim accés: 30/04/2017).

[32] Python per a Mac OS X.

https://www.python.org/downloads/mac-osx/ (Últim accés: 30/04/2017).

[33] Paquet networkx.

https://pypi.python.org/pypi/networkx/1.11 (Últim accés: 30/04/2017).

[34] Paquet six. https://pypi.python.org/pypi/six/1.10.0 (Últim accés: 30/04/2017).

[35] Paquet PyDotPlus.

https://pypi.python.org/pypi/pydotplus/2.0.2 (Últim accés: 30/04/2017).

[36] Paquet PyDot. https://pypi.python.org/pypi/pydot/1.2.3 (Últim accés: 30/04/2017).

[37] Paquet Graphviz.

https://pypi.python.org/pypi/graphviz/0.7 (Últim accés: 30/04/2017).

[38] Programari Graphviz per Windows.

http://www.graphviz.org/Download_windows.php (Últim accés: 30/04/2017).

[39] Programari Graphviz per Mac OS X.

http://www.graphviz.org/Download_macos.php (Últim accés: 30/04/2017).

[40] Estratègia top-down. *"Teoria d'EDA"*. Apunts de teoria de l'assignatura d'Estructures de Dades i Algorismes, 2013.

GLOSSARI

Mineria de dades és el procés de càlcul de descobriment de patrons en grans conjunts de dades que impliquen mètodes en la intersecció de la intel·ligència artificial, aprenentatge automàtic, estadística i sistemes de bases de dades.

Model de dades relacional és el model més utilitzat en l'actualitat per a modelar problemes reals i administrar dades dinàmicament. Permeten establir interconnexions (relacions) entre les dades (que estan guardades en taules), i treballar amb elles conjuntament.

Hipergraf és una generalització d'un graf, les arestes es diuen hiperarestes, que pot relacionar qualsevol quantitat de vèrtexs.

DOT Graph Description Language

XML eXtensible Markup Language

KISS Keep It Simple, Stupid!

HTTP Hypertext Transfer Protocol

FTP File Transfer Protocol

SSH Secure Shell

CSV Comma-separated Values

APÈNDIXS

Els apèndixs contenen la descripció dels passos necessaris per a la instal·lació i l'execució del paquet de software destinat a la anàlisis i a la visualització de les 2-estructures. També incorpora la guia d'ús del programa interactiu extern Graphviz.

A MANUAL D'INSTAL·LACIÓ

A.1 REQUERIMENTS I INSTAL·LACIÓ

Per a poder compilar i executar el paquet de software desenvolupat necessitem tenir instal·lada una versió de Python i algunes de les seves llibreries externes que anomenarem a continuació. A més del programari interactiu extern Graphviz.

A.1.1 Instal·lació de Python

El paquet de software desenvolupat es pot compilar i executar amb les últimes versions de Python: 2.7.13 i 3.6.1.

- Versions per a Windows [31]
- Versions per a Mac OS X [32]

La instal·lació de les versions de Python per a Ubuntu s'ha de realitzar mitjançant la terminal, amb les comandes següents:

 # sudo apt-get install python
 # sudo apt-get install python3

A.1.2 Instal·lació de les llibreries de Python

Les llibreries necessàries que Python qur no aporta per defecte i també s'han d'instal·lar són: networkx [33], six [34], pydotplus [35] o pydot [36] i graphviz [37]. Aquesta última només és necessària en els sistemes Ubuntu.

- Windows

Exemple:

1. Descarregar el paquet six de https://pypi.python.org/pypi/six#downloads.

Figura 47. Lloc de descàrrega del paquet six.

2. Descomprimim el fitxer que hem descarregat i observem que a la carpeta resultant conté un executable setup.py. L'executem de la manera següent:

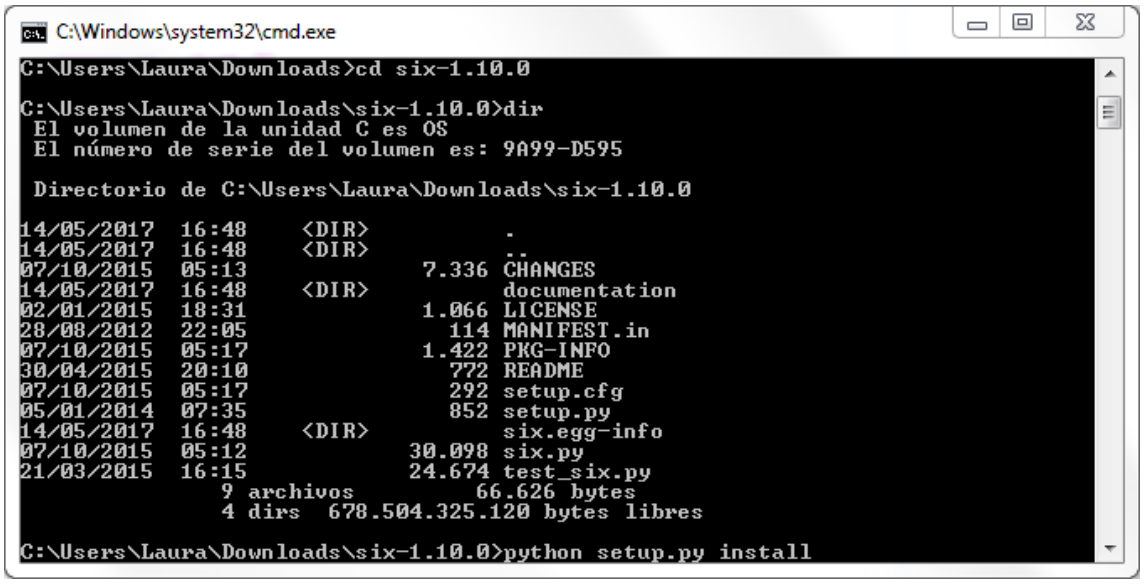


Figura 48. Terminal de Windows.

Utilitzarem la comanda python setup.py install per a la versió 2.7.13 de Python i la comanda python3 setup.py install per a la versió 3.6.1 de Python. Sempre com a administrador.

- Mac OS X i Ubuntu

Per a instal·lar els mòduls necessaris primer haurem de tenir el mòdul pip de Python 2 i 3. Per tenir-los executarem les comandes següents:

sudo apt-get install python-pip (Python 2)
sudo apt-get install python3-pip (Python 3)

Una vegada instal·lat el mòdul pip instal·larem la resta de mòduls segons la seva versió de Python:

sudo pip install <nom_del_mòdul> (Python 2)
sudo pip3 install <nom_del_mòdul> (Python 3)

A.1.2 Instal·lació del programari interactiu extern

L'aplicació és un executable i no necessita cap configuració prèvia. En un sistema Windows o Mac OS X només s'ha de fer doble clic sobre l'aplicació corresponent.

- Versions per a Windows [38]
- Versions per a Mac OS X [39]

La versió per a Ubuntu ha comportat alguna complicació i s'utilitza el programari XDot per a la visualització dels grafs i les 2-estructures. Per a la instal·lació d'aquest programari s'ha d'executar la comanda:

sudo apt-get install xdot.

B MANUAL D'USUARI DE L'ENTORN D'ANÀLISIS I VISUALITZACIÓ

Per a utilitzar el paquet de software ens l'hem de descarregar de https://github.com/lrodrin/TFG i guardar-lo en el nostre entorn de treball.

B.1 CREACIÓ I VISUALITZACIÓ D'UNA 2-ESTRCUTURA

Per a crear una nova 2-estructura s'ha d'obrir, del nostre entorn de treball, la carpeta /TFG/src/final/ que conté el paquet de software i seleccionar els fitxers changeImport.py (a Mac OS X i Ubuntu) o changeImportWindows.py (a Windows), segons la plataforma amb la que treballem; i main.py. Al executar el primer fitxer, l'entorn s'adequa per a l'execució fora de l'entorn de programació PyCharm. Al seleccionar el segon fitxer, s'obrirà un diàleg on es demanaran les dades necessàries per a crear la 2-estructura i el seu graf equivalent.

Primer ens demanarà quin tipus de fitxer d'entrada de dades relacionals proporcionarem. I el nom d'aquest fitxer. Recordem que els tipus de fitxers que processa el paquet de software són: ARFF, TXT i DB.

Es crearà la base de dades amb les dades relacionals que incorporava el tipus de fitxer proporcionat i es demanarà quin tipus de 2-estructura es vol crear.

A continuació, es crearà el graf corresponent a la nova 2-estructura. I finalment ens demanarà si volem utilitzar el clans més freqüents o no. En el cas que diguem que sí haurem d'entrar el suport (número de transaccions d'una taula SQLite que contenen tots els elements d'un conjunt d'elements). Per exemple amb un suport del 0.5 treballarem amb els conjunts d'elements que es trobin com a mínim en una transacció de cada 200. Amb un suport del 0.1, en una de cada 100.

Exemple 22. A la figura 49 podem observar el menú principal que crea i obre la visualització d'una 2-estructura i el seu graf equivalent.

```
Please enter the option for the type of file you provide:
 [1] = ARFF
 [2] = TXT
 [3] = DB
1
Please enter the name from ARFF file:
simpleGraph3
('Connection successful to', 'DB.db')
File simpleGraph3.arff opened
Please enter the option of 2-structure you want to create:
 [1] = plain
 [2] = plain with threshold
 [3] = linear
 [4] = exponential
3
A graph linearGraph.dot with 6 nodes was created
Want to work with the more frequent clans?
yes/no
no
A linear_2-structure.dot was created
```

Figura 49. **Menú principal.**

Una vegada s'han creat el graf i la 2-estructura del mateix tipus, apareixen dues finestres on es podrà visualitzar aquestes dues estructures.

C ALGORITMES

En aquest apèndix es mostren alguns algoritmes, sobretot els més importants, anomenats durant la documentació de la memòria per a tenir-ne una millor comprensió.

C.1 INIT_GRAPH

Crea i inicialitza un graf d'una taula de dades SQLite.

```
@staticmethod
def initGraph(tableName, cursor):
    """
    Create and initializes a graph from a table SQLite database source

    :param tableName: Table name
    :param cursor: Cursor object
    :type tableName: str
    :return: A graph
    :rtype: nx.Graph
    """
    graph = nx.Graph()  # Create an empty graph with no nodes and no edges
    columnNames, rows = Data.selectData(tableName, cursor)  # Select data from tableName
    Graph.addNodes(graph, columnNames, rows)  # Adding nodes to graph
    for (u, v) in combinations(graph.nodes(), 2):  # For the initialization all the edges from graph are
        # painted black and the edge style is dashed
        if u != v:
            graph.add_edge(u, v, color='black', style='dashed')

    return graph, rows
```

C.2 LABELED_EDGES

Comptar el nombre d'equivalències i etiqueta les arestes del graf amb aquest nombre

```
@staticmethod
def labeledEdges(graph, rows):
    """
    Count the number of equivalences and label the edges from graph with the total number of equivalences

    :param graph: Networkx's graph
    :param rows: Rows from a SQLite table
    :type graph: nx.Graph
    """
    numberOfEquivalences = dict()
    for row in rows:  # For each row in rows
        for (u, v) in combinations(row, 2):  # For each pair (u, v) in the row
            if u != v:  # If u and v have different values
                if (u, v) in numberOfEquivalences.keys():  # If exists edge (u, v) in a numberOfEquivalences
                    numberOfEquivalences[(u, v)] = numberOfEquivalences.get(
                        (u, v)) + 1  # Increases the number of equivalences
                    graph.edge[u][v]['label'] += 1  # Add the number of equivalences into label edge from graph

                else:  # If not exists edge (u, v) in numberOfEquivalences
                    numberOfEquivalences[(u, v)] = 1
                    graph.add_edge(u, v, label=1)
```

C.3 CREATE_PLAIN_GRAPH

Crea un graf pla.

```
@staticmethod
def createPlainGraph(graph, rows):
    """
    Create a plain graph

    :param graph: Networkx's graph
    :param rows: Rows from a SQLite table
    :type graph: nx.Graph
    :return: A plain graph
    :rtype: nx.Graph
    """
    Graph.labeledEdges(graph, rows)  # Labeling edges from graph
    labels = Graph.getLabelAttributesFromGraph(graph)  # Edge labels from graph

    for (u, v), label in labels.items():  # For each edge and label attribute in labels
        if graph.has_edge(u, v) and u != v:  # If exists edge (u, v) in graph and u and v have different values
            if label > 0:  # If label attribute from edge(u, v) is bigger than 0
                graph.add_edge(u, v, color='black', style='solid')  # Edge painted black and line style is not
                # dashed

    return graph
```

C.4 CREATE_PLAIN_GRAPH_WITH_THRESHOLD

Crea un graf pla amb llindar.

```
@staticmethod
def createPlainGraphWithThreshold(graph, rows, k):
    """
    Create a plain graph with threshold

    :param graph: Networkx's graph
    :param rows: Rows from a SQLite table
    :param k: Threshold
    :type graph: nx.Graph
    :type k: int
    :return: A plain graph with threshold
    :rtype: nx.Graph
    """
    Graph.labeledEdges(graph, rows)  # Labeling edges from graph
    labels = Graph.getLabelAttributesFromGraph(graph)  # Edge labels from graph

    for (u, v), label in labels.items():  # For each edge and label attribute in labels
        if graph.has_edge(u, v) and u != v:  # If exists edge (u, v) in graph and u and v have different values
            if label < k:  # If label attribute from edge(u, v) is smaller than k constant
                graph.add_edge(u, v, color='black', style='dashed')  # Edge painted black and line style is dashed
            elif label >= k:  # If label attribute from edge(u, v) is equal or greater than k constant
                graph.add_edge(u, v, color='black',
                               style='solid')  # Edge painted black and line style is not dashed

    return graph
```

C.5 CREATE_LINEAR_GRAPH

Crea un graf lineal.

```
@staticmethod
def createLinearGraph(graph, rows):
    """
    Create a linear graph

    :param graph: Networkx's graph
    :param rows: Rows from a SQLite table
    :type graph: nx.Graph
    :return: A linear graph
    :rtype: nx.Graph
    """
    Graph.labeledEdges(graph, rows)  # Labeling edges from graph
    labels = Graph.getLabelAttributesFromGraph(graph)  # Edge labels from graph

    for (u, v), label in labels.items():  # For each edge and label attribute in labels
        if graph.has_edge(u, v) and u != v:  # If exists edge (u, v) in graph and u and v have different values
            if label == 1:  # Equivalence class of 1
                graph.add_edge(u, v, color='black', style='solid')  # Edge painted black and style is dashed
            elif label == 2:  # Equivalence class of 2
                graph.add_edge(u, v, color='cyan', style='solid')  # Edge painted cyan and style is dashed
            elif label == 3:  # Equivalence class of 4
                graph.add_edge(u, v, color='green', style='solid')  # Edge painted green and style is dashed
            elif label == 4:  # Equivalence class of 4
                graph.add_edge(u, v, color='magenta', style='solid')  # Edge painted magenta and style is dashed
            elif label == 5:  # Equivalence class of 5
                graph.add_edge(u, v, color='orange', style='solid')  # Edge painted orange and style is dashed
            elif label == 6:  # Equivalence class of 6
                graph.add_edge(u, v, color='blue', style='solid')  # Edge painted blue and style is dashed
            elif label == 7:  # Equivalence class of 7
                graph.add_edge(u, v, color='red', style='solid')  # Edge painted red and style is dashed
            elif label == 8:  # Equivalence class of 8
                graph.add_edge(u, v, color='yellow', style='solid')  # Edge painted yellow and style is dashed
            elif label == 9:  # Equivalence class of 9
                graph.add_edge(u, v, color='brown', style='solid')  # Edge painted brown and style is dashed
            else:   # The others
                graph.add_edge(u, v, color='grey', style='solid')  # Edge painted grey and style is dashed

    return graph
```

C.6 CREATE_EXPONENTIAL_GRAPH

Crea un graf exponencial.

```
@staticmethod
def createExponentialGraph(linearGraph, rows):
    """
    Create a exponential graph

    :param linearGraph: Networkx's graph
    :param rows: Rows from a SQLite table
    :type linearGraph: nx.Graph
    :return: An exponential graph
    :rtype: nx.Graph
    """
    newGraph = Graph.createLinearGraph(linearGraph, rows)  # Create a new graph from linearGraph
    labels = Graph.getLabelAttributesFromGraph(newGraph)  # Edge labels from newGraph

    for (u, v), label in labels.items():  # For each edge and label attribute in labels
        if newGraph.has_edge(u, v) and u != v:  # If exists edge (u, v) in graph and u and v have different values
            if label == 1:  # Equivalence class of 1
                newGraph.add_edge(u, v, color='black', style='solid')  # Edge painted black and style is dashed
            elif 2 <= label < 4:  # Equivalence classes of (2-3)
                newGraph.add_edge(u, v, color='cyan', style='solid')  # Edge painted cyan and style is dashed
            elif 4 <= label < 8:  # Equivalence classes of (4-7)
                newGraph.add_edge(u, v, color='green', style='solid')  # Edge painted green and style is dashed
            elif 8 <= label < 16:  # Equivalence classes of (8-15)
                newGraph.add_edge(u, v, color='magenta', style='solid')  # Edge painted magenta and style is dashed
            elif 16 <= label < 32:  # Equivalence classes of (16-31)
                newGraph.add_edge(u, v, color='orange', style='solid')  # Edge painted orange and style is dashed
            elif 32 <= label < 64:  # Equivalence classes of (32-63)
                newGraph.add_edge(u, v, color='blue', style='solid')  # Edge painted blue and style is dashed
            elif 64 <= label < 128:  # Equivalence classes of (64-127)
                newGraph.add_edge(u, v, color='red', style='solid')  # Edge painted red and style is dashed
            elif 128 <= label < 256:  # Equivalence classes of (128-255)
                newGraph.add_edge(u, v, color='yellow', style='solid')  # Edge painted yellow and style is dashed
            elif 256 <= label < 512:  # Equivalence classes of (256-511)
                newGraph.add_edge(u, v, color='brown', style='solid')  # Edge painted brown and style is dashed
            else:  # The others
                newGraph.add_edge(u, v, color='grey', style='solid')  # Edge painted grey and style is dashed

    return newGraph
```

C.7 DECOMPOSITION

Descomposa un graf en clans primers.

```
@staticmethod
def decomposition(graph):
    """
    Decomposition of graph in primal clans

    :param graph: Networkx's graph
    :type graph: nx.Graph
    :return: The edges atributtes from graph and primal clans ordered
    :rtype: dict, dict
    """
    clansList = Clan.clans(graph, graph.nodes())  # List of clans
    primalClansList = Clan.primalClans(clansList)  # List of primal clans
    EdgesAtributtes = Graph.getColorAttributesFromGraph(graph)  # Edges atributtes from graph
    primalClansDict = OrderedDict(reversed(sorted(Clan.primalClansDict(primalClansList).items(),
                                                  key=lambda t: len(t[0]))))  # Dictionary of primal clans
    # sorted in reverse mode by primal clans length
    return EdgesAtributtes, primalClansDict
```

C.8 IS_CLAN

Comprova si un subconjunt del graf és un clan.

```
@staticmethod
def isClan(graph, subSet):
    """
    Checks if a subSet is a clan of graph

    :param graph: Networkx's graph
    :param subSet: Subset from graph
    :type graph: nx.Graph
    :type subSet: set
    :return: True if successful, False otherwise
    :rtype: bool
    """
    diff = set(graph.nodes()).difference(subSet)  # Subset formed by all nodes of graph less subSet passed as
    # parameter
    isClan = True
    for external in diff:  # For each subset in diff
        for (u, v) in combinations(subSet, 2):  # For each pair (u, v) in the subSet combinations
            if graph.has_edge(external, u) and graph.has_edge(external, v):  # If exists the edge (external,
                # u) and (external, v) in graph
                colorX = graph.edge[external][u]['color']
                colorY = graph.edge[external][v]['color']
                lineStyleX = graph.edge[external][u]['style']
                lineStyley = graph.edge[external][v]['style']
                if colorX != colorY or lineStyleX != lineStyley:  # If the pair (external, u) and (external,
                    # v) not have the same color edge
                    isClan = False
    return isClan
```

C.9 PRIMAL_CLANS

Retorna els clans primers d'una llista de clans.

```
@staticmethod
def primalClans(clansList):
    """
    Return a list of primal clans from a list of clans specified by clansList

    :param clansList: List of clans
    :type clansList: list
    :return: List of primal clans
    :rtype: list
    """
    noPrimalClans = defaultdict(bool)
    primalClansList = list()
    for i, key in enumerate(clansList):  # For each clan in clansList
        for j in range(i + 1, len(clansList)):
            intersection = clansList[i] & clansList[j]  # clansList[i] intersection clansList[j]
            if len(intersection) != 0 and intersection < clansList[i] and intersection < clansList[j]:
                # If exist an overlapping, the clan is not a primal clan
                noPrimalClans[frozenset(clansList[i])] = True
                noPrimalClans[frozenset(clansList[j])] = True

    for clan in clansList:
        if not noPrimalClans[frozenset(clan)]:  # If not exist overlapping the clan is a primal clan
            primalClansList.append(clan)

    return sorted(primalClansList, key=len)
```

C.10 CREATE_GRAPHVIZ_STRUCTURE

Crea una 2-estructura en llenguatge DOT.

```
@staticmethod
def createGraphvizStructure(edgesAtributtes, primalClansDict, structureName):
    """
    Create a 2-structure from a dictionary of primal clans specified by primalClansDict

    :param edgesAtributtes: Edges atributtes from a graph
    :param primalClansDict: Dictionary of primal clans
    :param structureName: Dot file name
    :type edgesAtributtes: dict
    :type primalClansDict: dict
    :type structureName: str
    :return: A 2-structure
    """
    structure = pydotplus.Dot(strict=True, graph_type="digraph", graph_name=structureName[:-4], compound="true",
                              fontname="Verdana",
                              fontsize=12, newrank="true")
    structure.set_node_defaults(shape="circle")

    # Creating external nodes
    for primals in primalClansDict.values():  # For each primal clan
        for primal in primals:  # For each sub primal clan
            if len(primal) == 1:  # If primal clan is a trivial clan
                # if not structure.get_node("".join(primal)):
                # Exclude the repetitive nodes
                structure.add_node(pydotplus.Node("".join(primal)))  # Add node to structure

    # Creating clusters
    for key, values in primalClansDict.items():  # For each primal clan and their sub primal clans
        if len(values) == 2:  # Primitive structure
            cluster = pydotplus.Cluster("".join(key), rank="same")  # Create a cluster
        else:
            cluster = pydotplus.Cluster("".join(key))  # Create a cluster

        cluster.set_node_defaults(shape="point")

        if len(values) <= 10:
            # Creating edges and nodes inside the cluster
            for primalClan1, primalClan2 in combinations(values, 2):
                u = "s_%s" % "".join(primalClan1)
                v = "s_%s" % "".join(primalClan2)
                if u != v:  # If node cycle not exists
                    edge = pydotplus.Edge(u, v, color=Clan.getColorClans(edgesAtributtes, primalClan1, primalClan2),
                                          arrowhead="none")

                    if not cluster.get_edge(edge):  # If edge not exists and node cycle also not exists
                        cluster.add_edge(edge)  # Add edge to cluster
        else:
            # Creating nodes inside the cluster
            for primals in primalClansDict.values():  # For each primal clan
                for primal in primals:  # For each sub primal clan
                    node = pydotplus.Node("s_%s" % "".join(primal))
                    if not cluster.get_node(node):  # If edge not exists and node cycle also not exists
                        cluster.add_node(node)  # Add edge to cluster

        structure.add_subgraph(cluster)  # Add cluster to structure

    # Creating external edges
    for values in primalClansDict.values():  # For each sub primal clans
        for primal in values:  # For each sub primal clan
            u = "s_%s" % "".join(primal)
            if primalClansDict.get(frozenset(primal)):  # If primal exists as a key in primalClansDict
                v = "s_%s" % "".join(primalClansDict.get(frozenset(primal))[0])
                edge = pydotplus.Edge(u, v, lhead="cluster%s" % "".join(u[1:]), arrowhead="none")

            else:  # If primal not exists as a key in primalClansDict
                v = "".join(primal)
                edge = pydotplus.Edge(u, v, arrowhead="none")

            structure.add_edge(edge)  # Add edge to structure

    structure.write(structureName)  # Write a Dot file with all previous information
    print("A %s was created" % structureName)
```