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 amb 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, a 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

ÍNDEX DE CODIS

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 de la 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 d'implementació 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 de del programari interactiu extern.

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 [R]. Es tracta d'un model basat en llenguatge *DOT* [1] que proporciona una manera simple de descriure gràfiques i que les fa més entenedores per a l'usuari.

Tots els models basats en *DOT* d'aquest treball són creats des de zero i l'algoritme encarregat de crear aquests models generarà la disposició dels elements que facilita a l'usuari la tasca d'entendre el model 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:

- El conjunt dels nombres naturals és $N = \{1, 2, 3, \ldots\}$.
- El conjunt dels enters és $Z = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$.
- El conjunt dels nombres racionals és $Q = \{m/n : m \in Z, n \in Z, n \neq 0\}$.

Si A i B són conjunts, llavors:

A és un subconjunt de B ($A \subseteq B$) si cada element de A és també un element de B. Per exemple, el conjunt de nombres parells naturals és un subconjunt del conjunt de tots els nombres naturals. Cada conjunt A és un subconjunt de si mateix, és a dir, $A \subseteq A$. El conjunt buit és un subconjunt de cada conjunt A, $0 \subseteq A$. 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: 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].

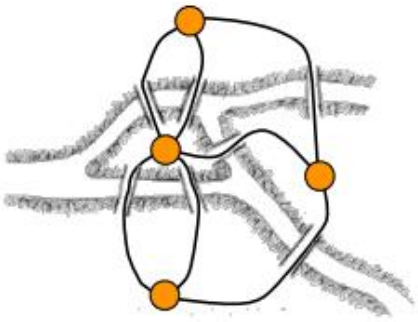


Figura 1. Solució d'Euler.

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 vèrtex 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 vèrtex diferent.

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

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

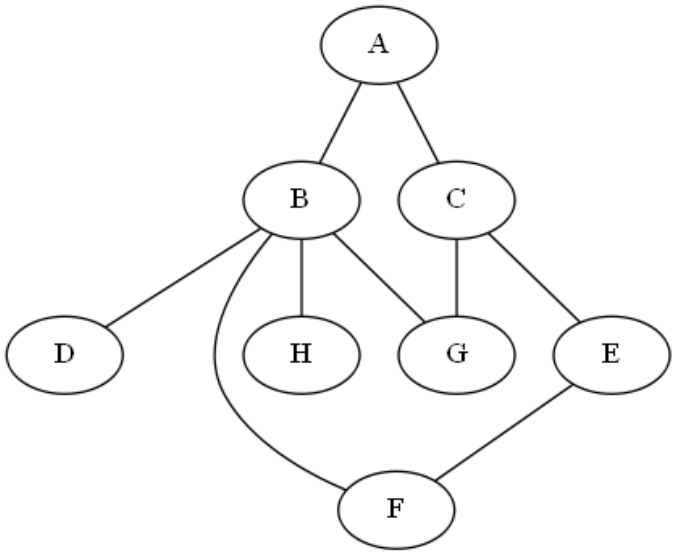


Figura 2. Graf no dirigit.

Exemple 2. A la figura 3 veiem la representació gràfica de l'exemple 1 però en un graf 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.

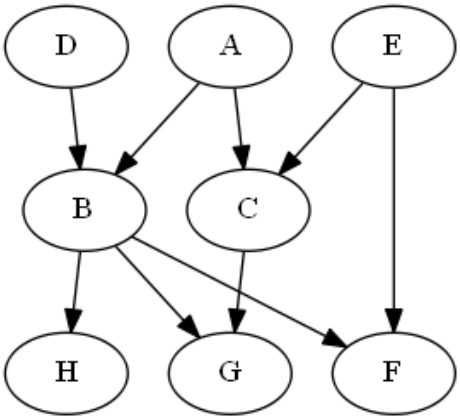


Figura 3. Graf dirigit.

Els grafs (dirigits o no) 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 veiem la representació gràfica de l'Exemple 1 però en un graf 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.

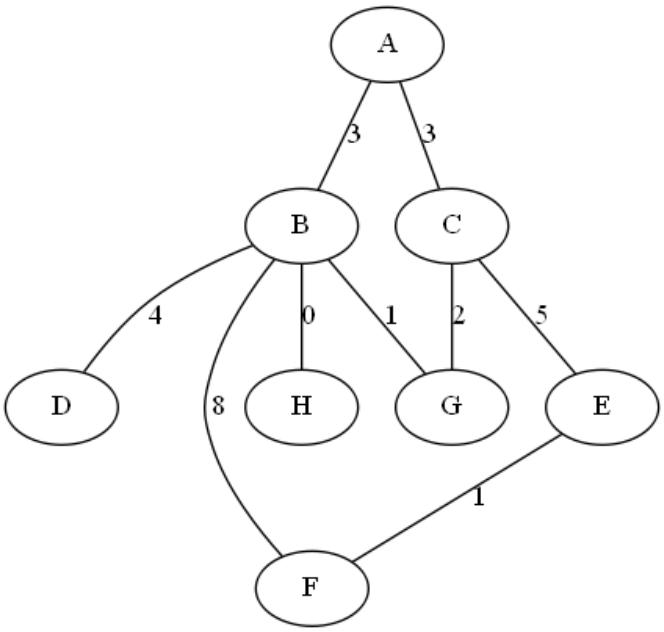


Figura 4. Graf ponderat.

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 vèrtex connectats per les arestes. Un camí és un cicle, si comença i acaba al mateix vèrtex. Un camí simple és un camí sense cap vèrtex repetit. Un graf està connectat si hi ha 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 múltiples 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 *"estàtica"* de 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 a les arestes. Les 2-estructures no etiquetades representen l'estructura principal del paquet de software desenvolupat.

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

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, 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 cada node intern que es troba en la 2-estructura representa un element del graf o bé un subconjunt d'aquest. Més endavant expliquem el procés amb el qual s'obté aquesta representació.

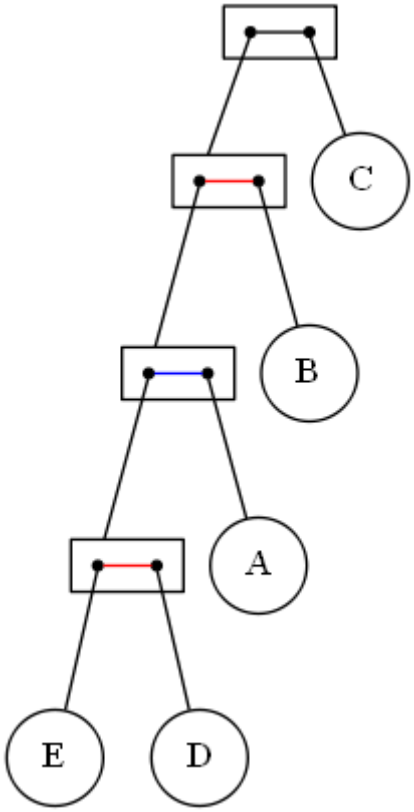


Figura 5. 2-estructura de l'exemple 4.

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 projecte 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 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 per al 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: 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 també es pot trobar per a dispositius iOS i Android.

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

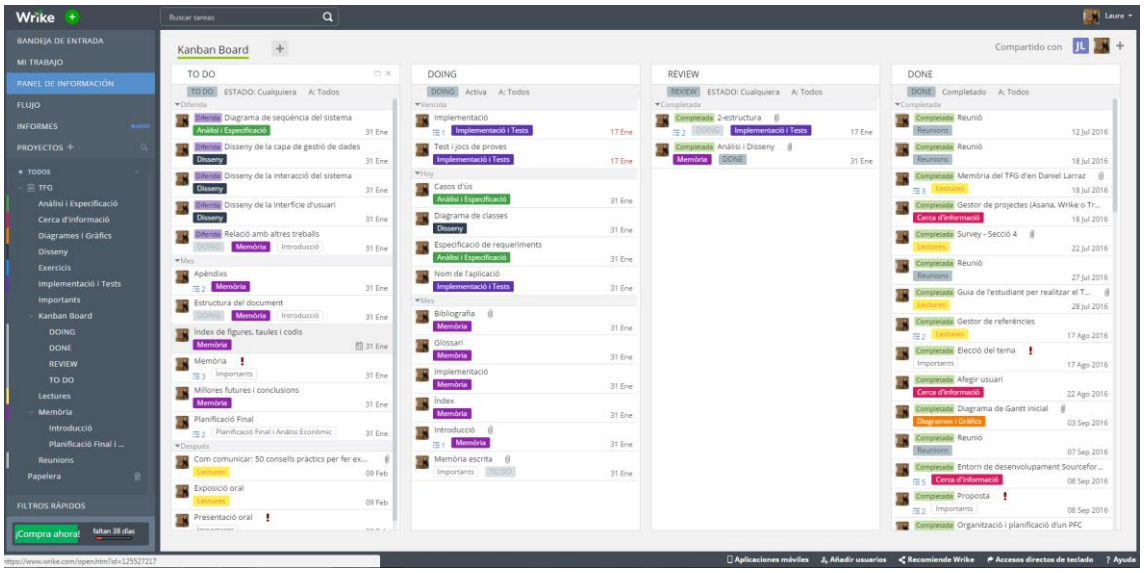


Figura 6. Taulell *Kanban* a Wrike.

Wrike també incorpora la generació automàtica del diagrama de Gantt. Però per a la simplificació del diagrama es va optar per a importar les tasques 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. De la mateixa manera, 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 posa a prova la implementació mitjançant un conjunt de jocs de prova que es composen de casos senzills i aleatoris per a 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 consulta 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 PostgreSQL o Oracle.

Per a la creació i la manipulació dels grafs es va utilitzar NetworkX [10].

NetworkX és un paquet de programari del llenguatge 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 (una estructura de dades que associa claus o claus amb valors), una cadena de text, una imatge, un objecte
XML , un altre graf, etc.

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 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.

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 l'objectiu s'han utilitzat les diferents llibreries següents: os, sys, subprocess i six, amb les seves funcionalitats respectives.

- Mòdul os: llibreria que permet l'ús de funcionalitats pròpies d'un sistema operatiu.
- Mòdul sys: llibreria que permet l'accés a algunes variables que introdueixen els usuaris utilitzades per un intèrpret d'ordres, que té la capacitat de traduir-les, i les funcions que interactuen amb aquest intèrpret d'ordres.
- Mòdul subprocess: llibreria que té 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.

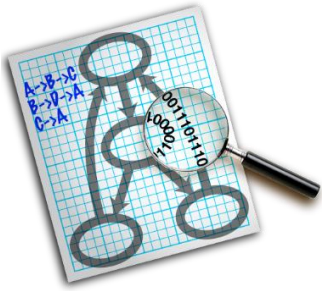


Figura 9. Logotip del programari Graphviz.

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) es crea la visualització corresponent (a la dreta de la figura).

```
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];
}
```

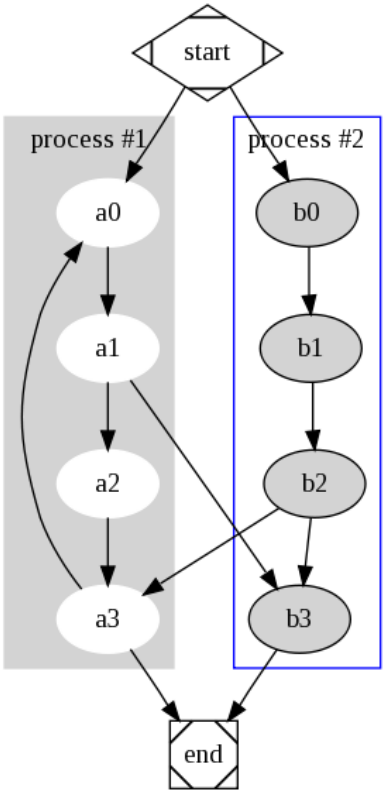


Figura 10. Conversió del programari Graphviz de l'exemple 5.

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 amb 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.

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 dades mateixes.

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 visualització del paquet de software desenvolupat.

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 del treball. En el capítol 5 es mostra l'anàlisi i el disseny del paquet de software implementat.

En els capítols 6, 7, 8, 9 i 10 es documenta la implementació del paquet de software desenvolupat. Concretament, en el capítol 6 es descriuen les dades d'entrada que es necessiten. 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 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.

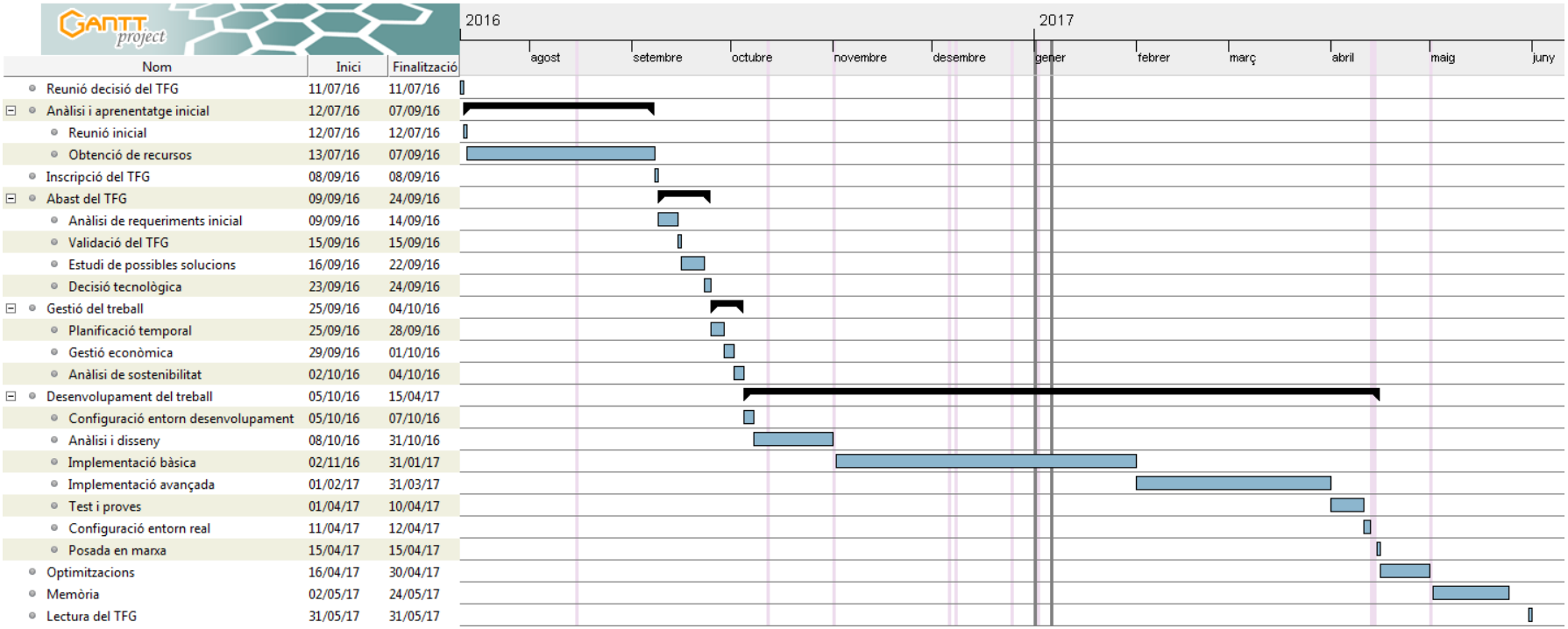
Fase 3 (Fase d'implementació avançada): des del 1/02/2017 al 31/03/2017 al diagrama de Gantt. Fase on, una vegada tenim la base, podem realitzar el procediment per acabar d'implementar els algoritmes definitius del treball i la integració d'aquests amb les bases de dades relacionals.

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].

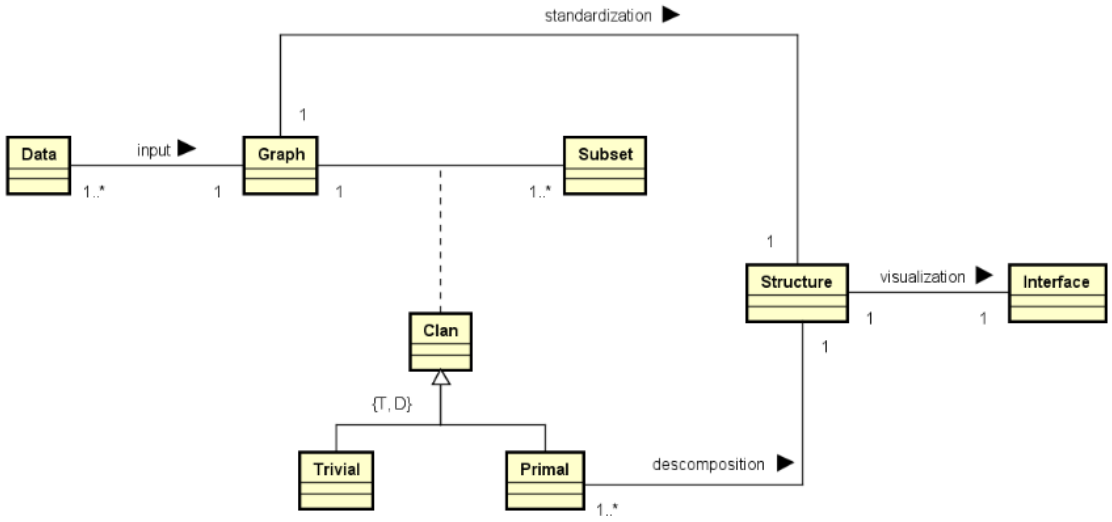


Figura 15. Diagrama de classes amb el modelat i el disseny de les 2-estructures.

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 classe d'equivalència 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 els clans que contenen 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:

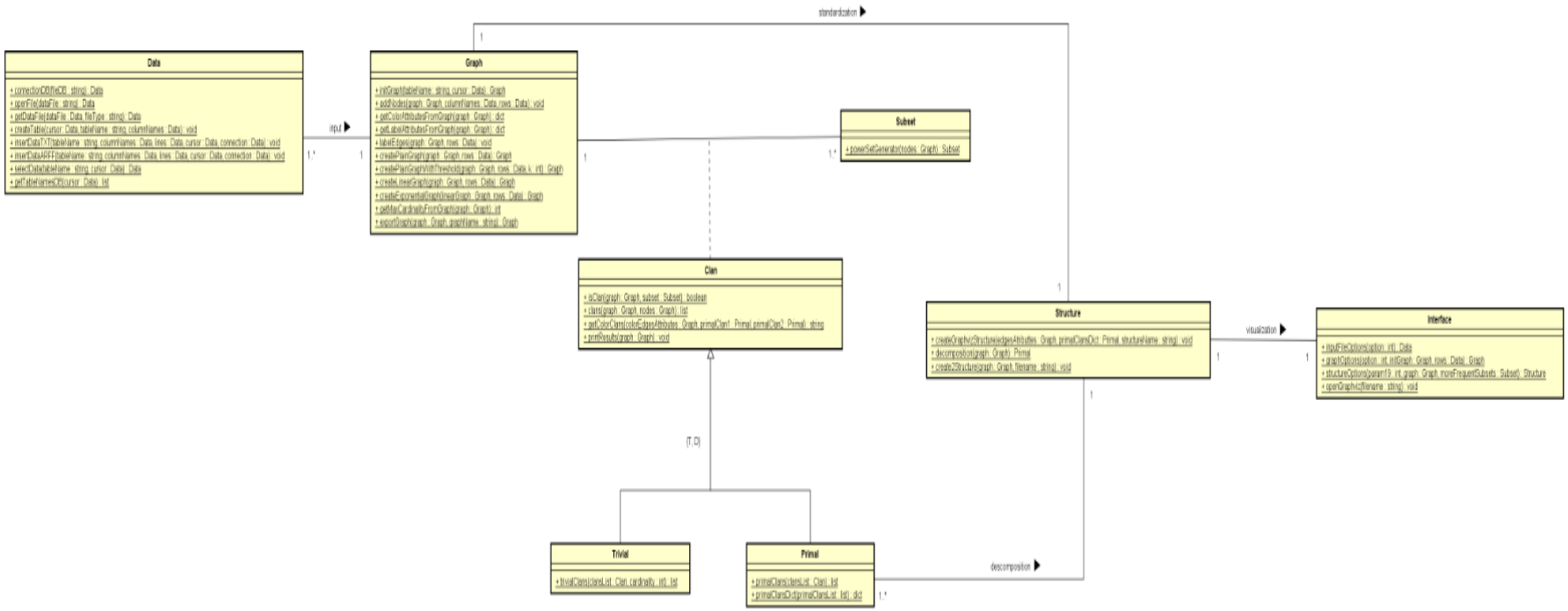


Figura 16. Diagrama de classes complet amb el modelat i el disseny de les 2-estructures.

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 conjunts d'elements 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].

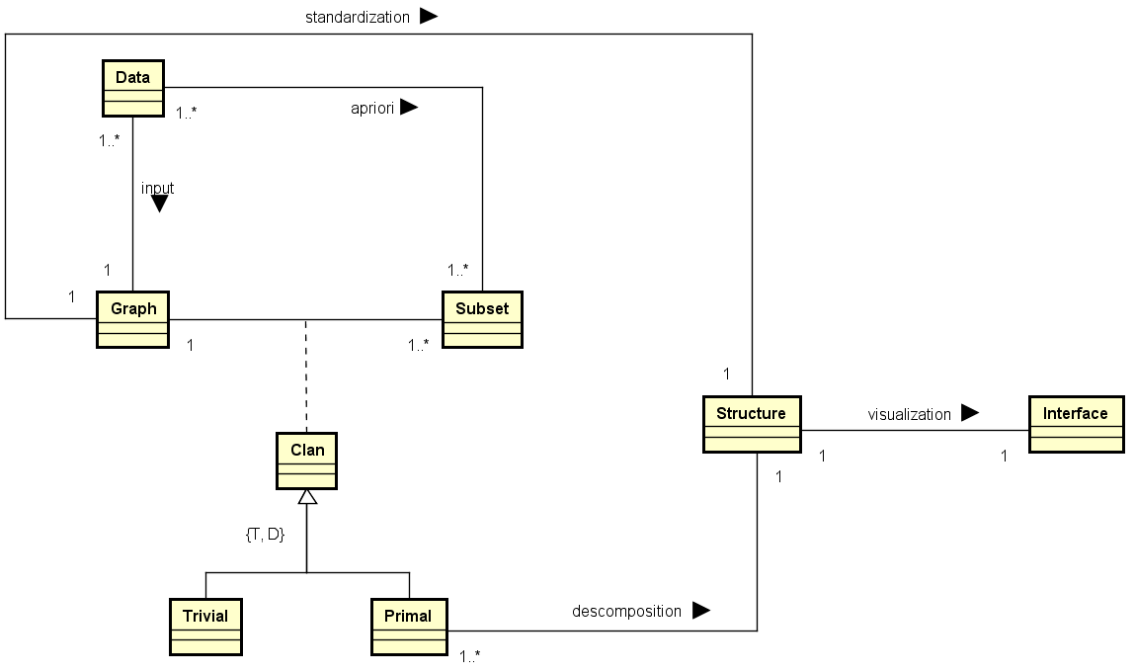


Figura 17. Nou diagrama de classes amb el modelat i el disseny de les 2-estructures.

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:

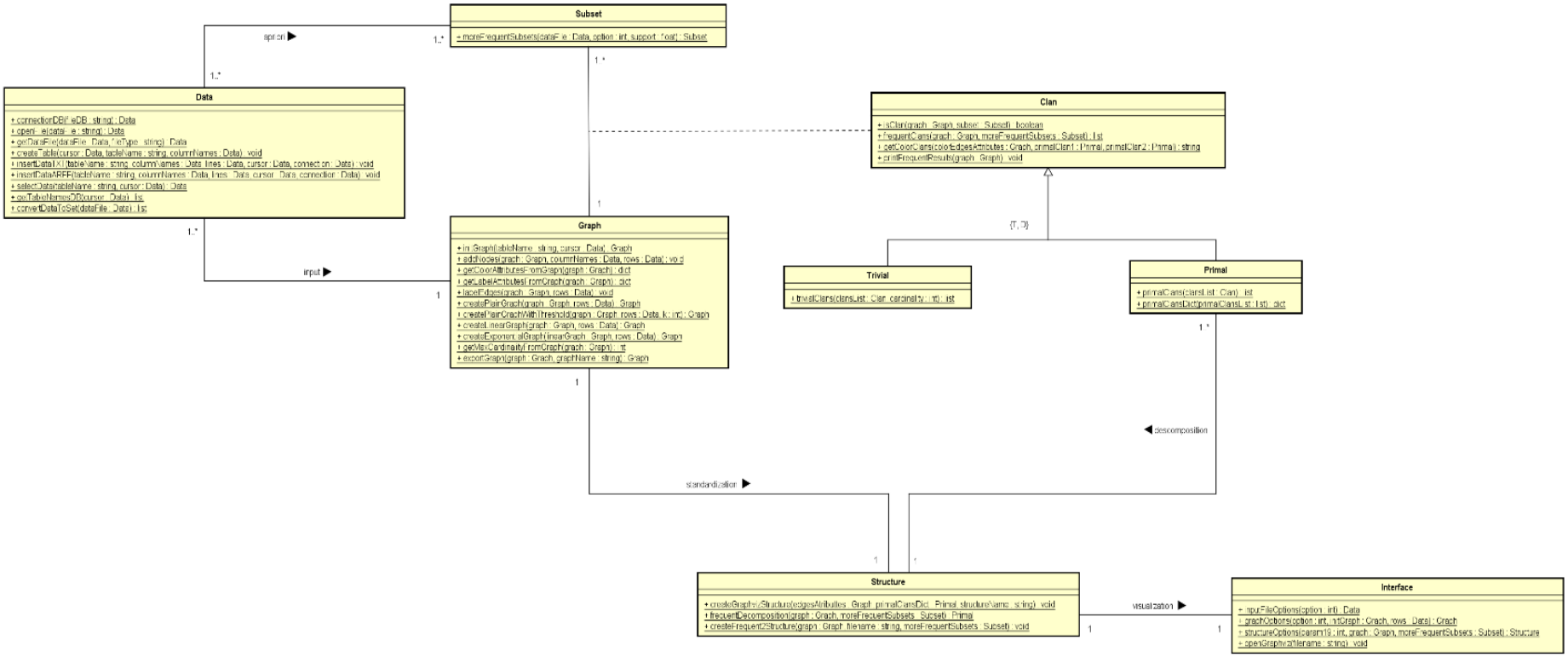


Figura 18. Nou diagrama de classes complet amb el modelat i el disseny de les 2-estructures.

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

CSV 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 $(5 \cdot (5-1) / 2)$.

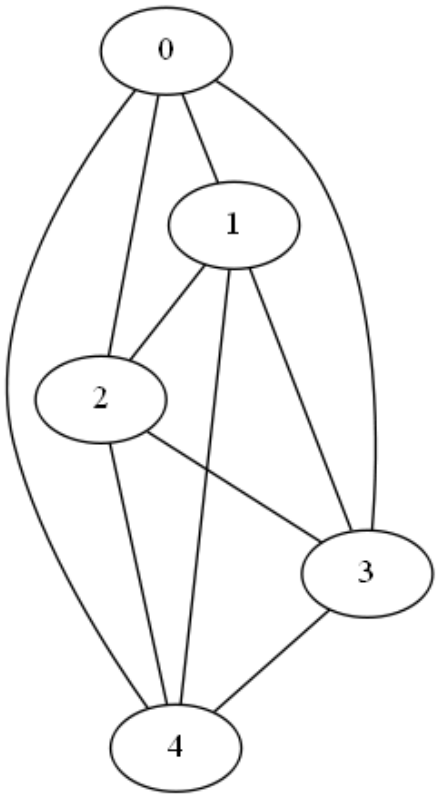


Figura 21. Graf complet.

A més, tots els grafs desenvolupats en el paquet de software també compleixen la localitat de Gaifman [27]. 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:

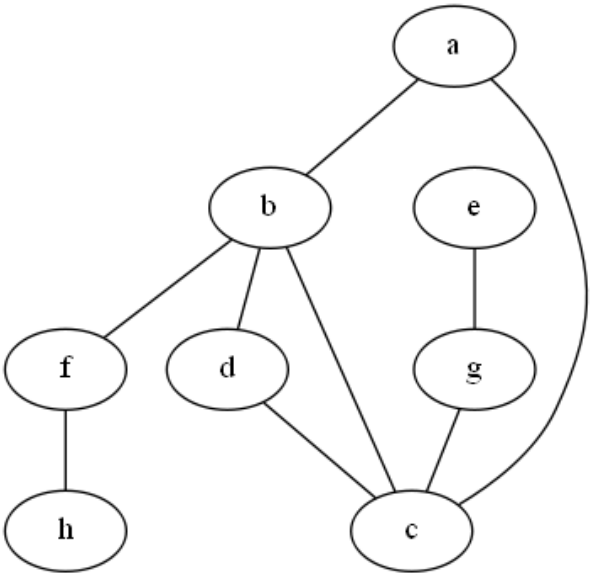


Figura 22. **Graf de Gaifman de l'exemple 9.**

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 junts 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 de nodes 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.

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 columnes i les files per a afegir-les com a nodes i a arestes del graf. Cada element d'una fila (sense repeticions) representa un node del graf i per a cada parella de valors <atribut:valor> de la taula es crea una aresta entre ella.

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 de les files (obviant la repetició dels valors) representa un node del graf. Per tant, els nodes del graf seran: a, c, b, f, d i e. Nombre total de nodes del graf igual a sis. I per a cada parella de nodes de cada fila es crearan les arestes.

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.

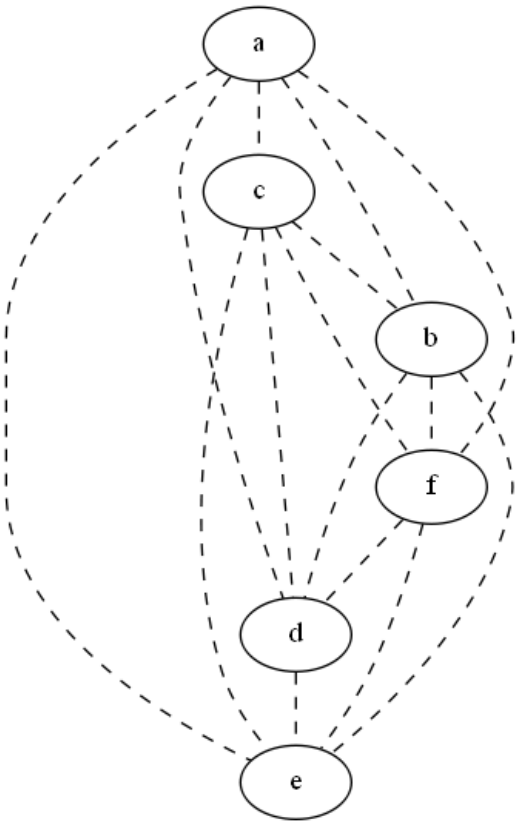


Figura 23. Inicialització del graf de l'exemple 10.

7.1.2 Classes d'equivalència de les arestes dels grafs

Una classe d'equivalència és el nombre de parelles de valors, de les dades relacionals que conté una taula SQLite, repetides. Cada classe d'equivalència es representa mitjançant les arestes dels grafs amb un color diferent. O sigui que, cada classe d'equivalència estarà representada per un únic color. 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, una 2-estructura és simètrica si totes les seves arestes són simètriques. I una 2-estructura és antisimètrica si totes les seves arestes són antisimètriques.

Exemple 11. A la figura 24 es mostra el graf que defineix una 2-estructura. La 2-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.

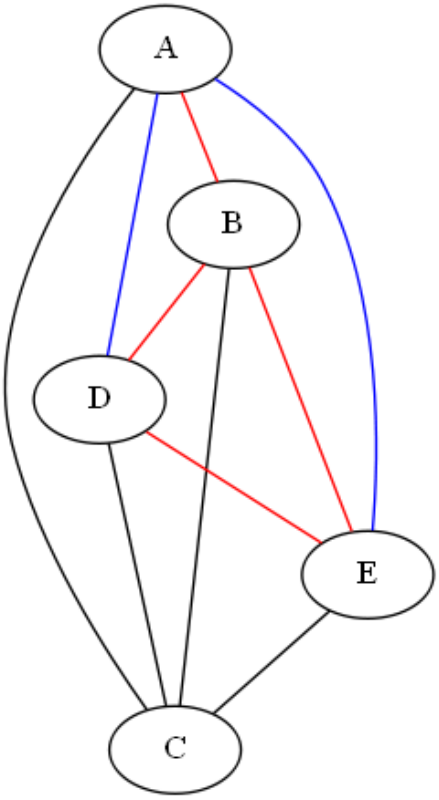


Figura 24. Graf que defineix una 2-estructura de l'exemple 11.

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 Algoritme labeledEdges

L'algoritme que acoloreix les arestes dels grafs segons la classe d'equivalència a la qual pertanyen s'anomena *"labeledEdges"*. Aquest algoritme compte el número d'equivalències de cada parella de valors de la taula SQLite. I per a cada aresta del graf que les representa, les etiqueta amb el nombre total d'equivalències trobades.

```python
@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 itertools.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)
```

Codi 1. Algoritme *labeledEdges*.

7.1.3 Exportació dels grafs

Una vegada s'han creat els grafs i marcat les classes d'equivalències d'aquest, s'exporten al llenguatge *DOT* del que fa ús el programari interactiu extern Graphviz, mitjançant una crida a la llibreria NetworkX i al seu submòdul PyDot conjuntament.

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 els comentaris d'estil: / * * / i //.

Exemple 12. A la figura 25 es poden observar dos fitxers en format *DOT* que representen el graf no dirigit (esquerra) i el graf dirigit (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 25. 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 relacions entre els nodes. I pels grafs dirigits s'utilitza la fletxa (->).

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

La figura 26 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.

```
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 26. Gramàtica del llenguatge *DOT* .

Les paraules node, edge, graph, digraph, subgraph i strict no són sensibles a les majúscules. I 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 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 primer lloc, 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 el 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 tot el dibuix de l'agrupació dins d'un rectangle delimitador.

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 l'esquerra de la figura 27 es pot veure un exemple senzill d'un un script que descriu l'estructura d'un graf en format DOT i el graf resultant que produeix el programari interactiu extern Graphviz a la dreta. El graf resultant és un graf no dirigit i conté 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 relacions o arestes entre els nodes d'aquest 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];
}
```

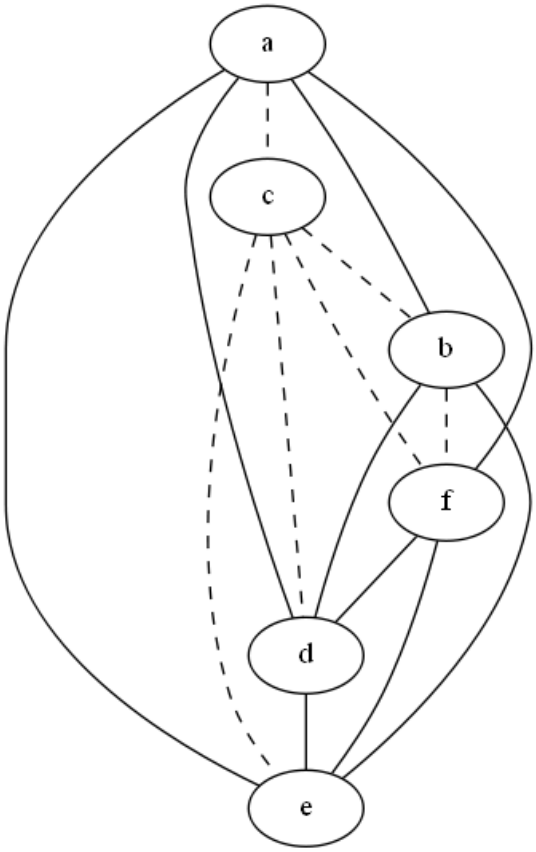


Figura 27. Script senzill en format *DOT* i graf resultant.

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 <atribut:valor> de cada fila de la taula SQLite, amb les que es representen els nodes i les arestes del graf, s'acoloreixen del mateix color si les arestes que formen apareixen en aquesta taula. O sigui que s'acoloreixen totes les arestes del graf on el nombre total d'equivalències de cadascuna sigui igual o superior a u.

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

A partir de l'algoritme següent es pot construir 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
```

Codi 2. Algoritme pla.

L'algoritme pla dibuixa les arestes entre els nodes del graf que representen una o més relacions a la taula corresponent. Concretament, les arestes es pinten de color negre amb la línia contínua. Si no existeix una relació entre els nodes del graf a la taula, les arestes es dibuixen de color negre i amb la línia discontínua.

L'algoritme també etiqueta les arestes amb el nombre total de relacions que existeixen a la taula per a cada parella de nodes. El nombre total de relacions ha de ser igual o superior a u. En aquest tipus de graf, l'etiquetatge de les arestes amb el nombre total de relacions no és molt rellevant però més endavant ho serà.

Exemple 14. A la figura 28 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 i indica que no s'ha trobat cap relació entre les parelles de nodes de la taula apropiada. La segona classe d'equivalència està representada per les arestes no discontínues i indica la o les relacions que s'han trobat entre les parelles de nodes de la taula apropiada. Cada aresta ponderada també ens indica el nombre total de les relacions trobades.

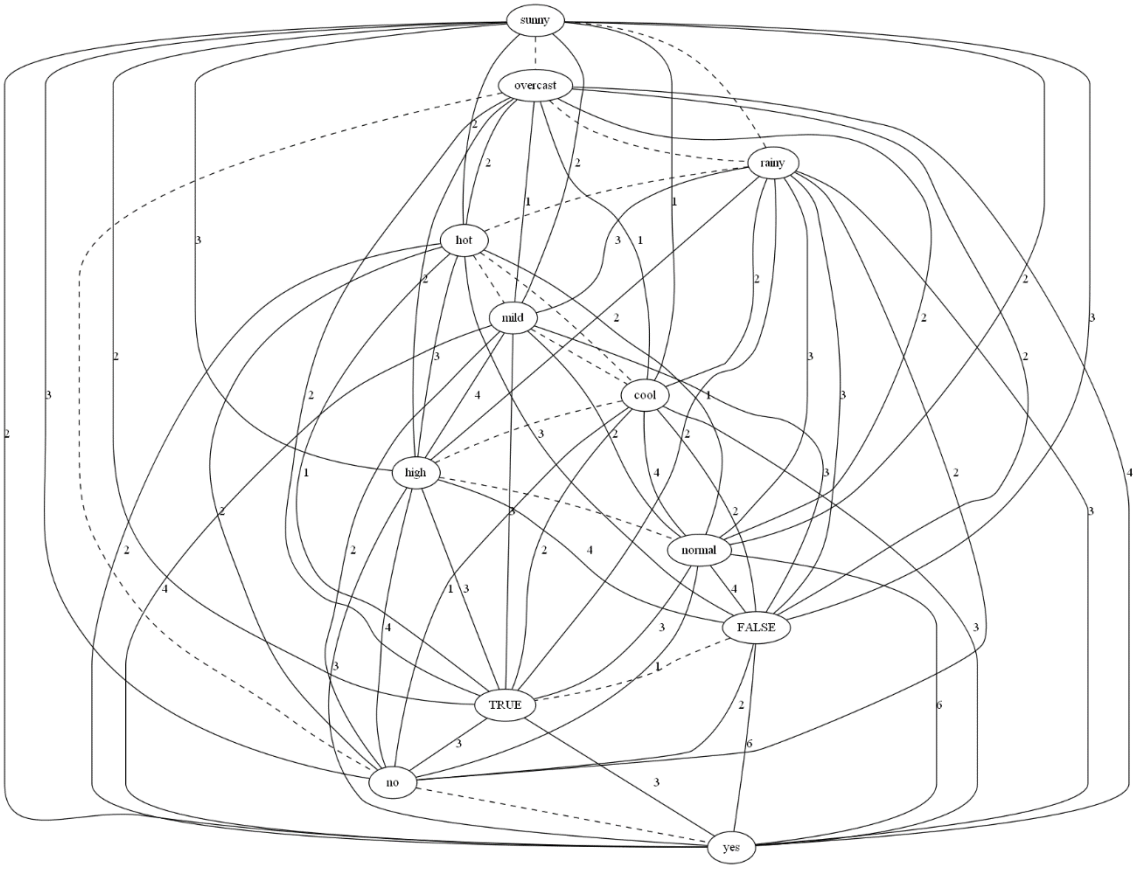


Figura 28. **Graf pla.**

7.2.2 Graf pla amb llindar

Un graf pla amb llindar és un graf pla on el nombre total de relacions de cada aresta entre els nodes del graf es dibuixa amb una línia contínua segons si aquest valor és més gran o igual, o més petit que un llindar (nombre enter introduït per l'usuari).

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

A partir de l'algoritme següent es pot construir 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
```

Codi 3. Algoritme pla amb llindar.

L'algoritme pla amb llindar també etiqueta les arestes amb el nombre total de relacions que existeixen a la taula SQLite corresponent per a cada parella de nodes. Així, si el nombre total de relacions d'una aresta entre una parella de nodes del graf és igual o més gran que el valor del llindar anotat per l'usuari, l'algoritme pla amb llindar dibuixa l'aresta de color negre i amb la línia contínua. En canvi, si el nombre total de relacions d'una aresta entre una parella de nodes del graf és més petit que el valor del llindar anotat per l'usuari, l'algoritme pla amb llindar distingeix la línia de l'aresta discontínuament.

Exemple 15. A la figura 29 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.

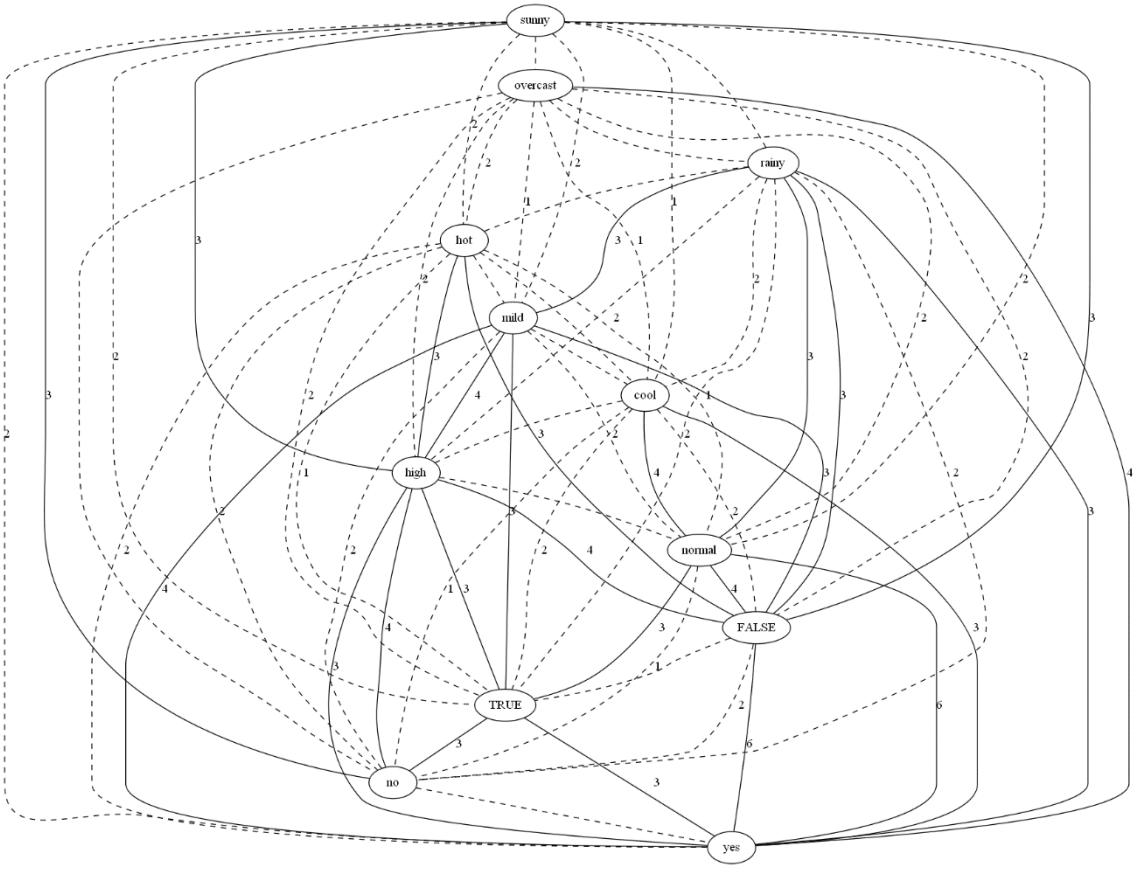


Figura 29. **Graf pla amb llindar.**

7.2.3 Graf lineal

Un graf lineal és un graf pla on el nombre total de relacions de cada aresta entre els nodes del graf s'acoloreix d'un color diferent. Cada color és únic per a la classe d'equivalència existent. Concretament, en el treball es processen 10 colors bàsics [28] que més endavant s'anomenen.

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.

A partir de l'algoritme següent es pot construir 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
    potentialColors = {1: 'black', 2: 'cyan', 3: 'green', 4: 'magenta', 5: 'orange', 6: 'blue',
                       7: 'red', 8: 'yellow', 9: 'brown', 10: 'grey'}  # Potential colours for equivalence classes

    for label, color in potentialColors.items():  # For each label and color edge attributes in potentialColors
        for (u, v) in graph.edges():  # For each edge from graph
            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' in graph[u][v] and label == graph[u][v]['label']:  # If edge have the label attribute and
                    # the number of equivalences in potentialColors are the same
                    if graph[u][v]['label'] > 0:  # If the number of equivalences is greater than 0
                        graph.add_edge(u, v, color=color,
                                       style='solid')  # Edge painted with a color from potentialColor and line
                        # style is not dashed

    return graph
```

Codi 4. Algoritme lineal.

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.

A continuació es mostra la taula de les equivalències entre les classes d'equivalències i el color que les representen:

Classe d'equivalència	Color
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 30 es pot observar un graf lineal. Cada classe d'equivalència està diferentment representada per un únic color. El nombre total de classes d'equivalències d'aquest graf és sis. Les classes 0, 1, 2, 3, 4 i 6. Per tant, segons la taula d'equivalències anterior, els colors que s'han d'utilitzar per a acolorir les arestes són cinc.

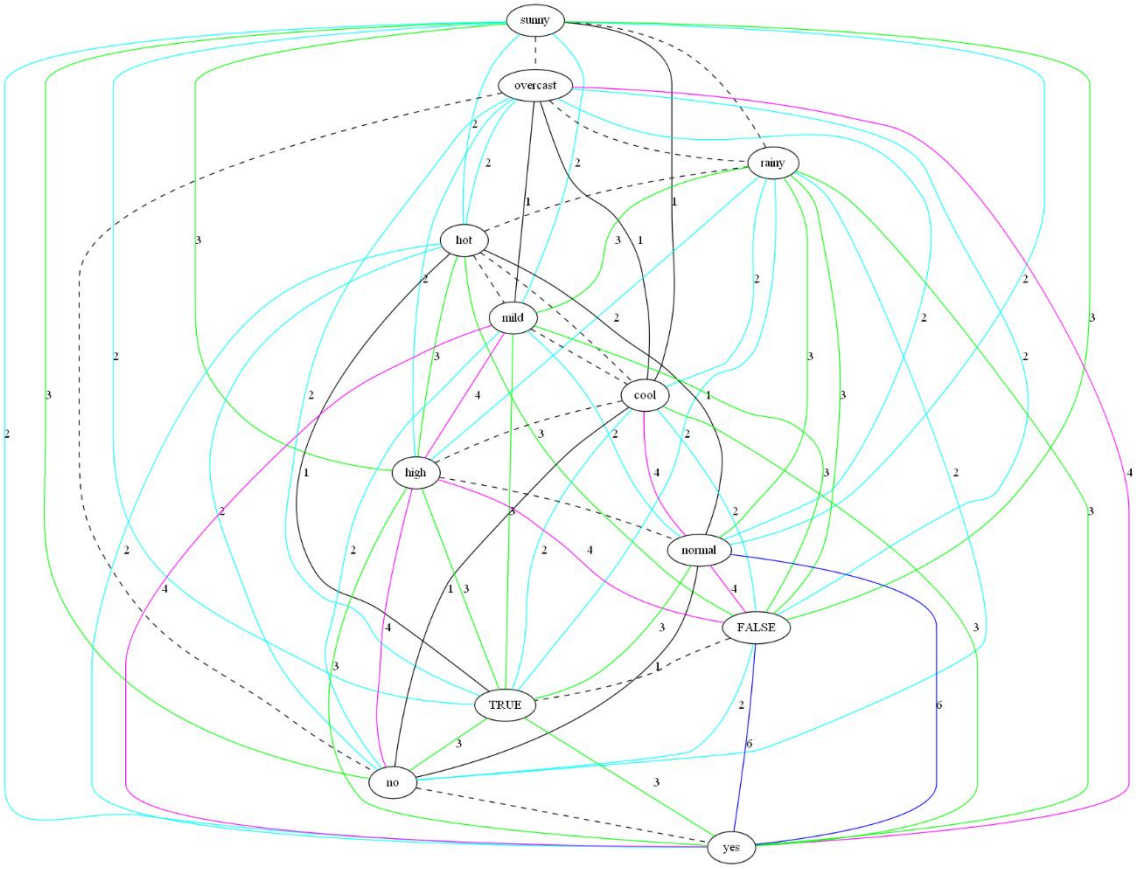


Figura 30. **Graf lineal.**

7.2.3 Graf exponencial

Un graf exponencial és un graf pla on diferents classes d'equivalències s'acoloreixen del mateix color. Cada color ja no és únic per a cada classe d'equivalència, sinó que un mateix color comprèn un interval exponencial de classes d'equivalències. El rang de l'interval exponencial va de 2^0 fins a 2^{10}. Per tant, en aquest tipus de grafs, les classes d'equivalències s'agrupen segons la sèrie de potències del dos anterior.

Es processen 10 colors bàsics i cada aresta té el mateix número total d'equivalències, com en els grafs lineals. Més endavant s'anomena la composició dels grups amb les classes d'equivalències i els seus colors pertinents.

A partir de l'algoritme següent es pot construir 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
            elif 512 <= label < 1024:  # Equivalence classes of (512-1023)
                newGraph.add_edge(u, v, color='grey', style='solid')  # Edge painted grey and style is dashed

    return newGraph
```

Codi 5. Algoritme exponencial.

Al inici, l'algoritme exponencial crea un graf lineal. Aquest graf lineal ja conté les arestes ponderades amb el nombre total de relacions de cada parella de nodes de la taula.

L'algoritme a partir del graf lineal creat, acoloreix les diferents classes d'equivalències existents segons l'interval exponencial que comprenen. També pinta les arestes discontínues quan no s'han trobat relacions entre les parelles de nodes a la taula equivalent.

A continuació es mostra la taula que conté els grups amb els quals s'agrupen les classes d'equivalències i el color del grup amb el qual es dibuixen:

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, 1023)	Gris

Taula 10. Taula dels grups d'equivalències.

L'interval que compren les classes d'equivalències 0 i 1 simbolitzen els grafs plans.

Exemple 17. A la figura 31 es pot observar un graf exponencial. El nombre de classes d'equivalències del graf és sis. Igual que en el graf lineal de la figura 30, ja que s'ha construït a partir d'aquest. Es pot comprovar també com les classes d'equivalències 2 i 3 pertanyen a un mateix interval, segons la taula d'equivalències anterior. I estan acolorides del mateix color (turquesa). Observem el mateix comportament amb les classes d'equivalència 4 i 6 però amb el color verd.

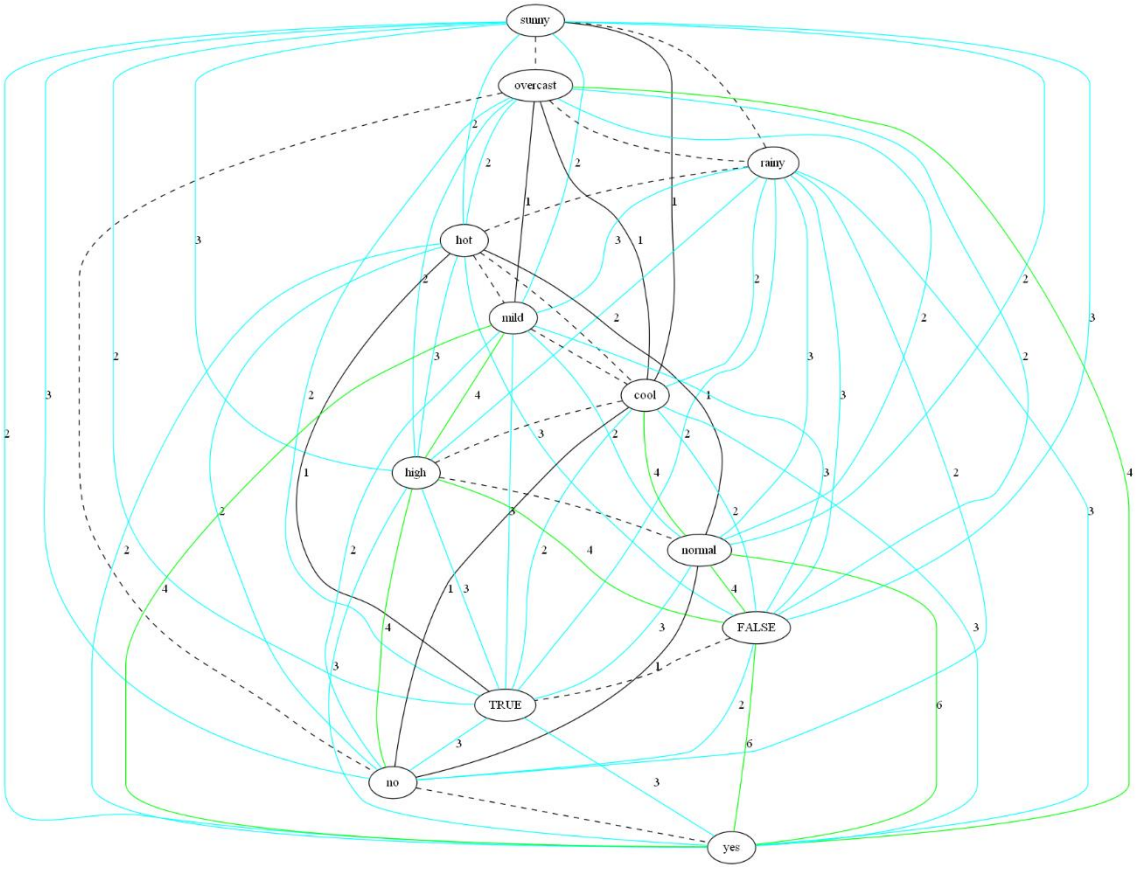


Figura 31. Graf exponencial.

Al suposar que les etiquetes de cada aresta dels grafs formen part de la definició de les 2-estructures, 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 de *"divideix i venceràs"* [29].

La idea amb la que es basa la descomposició dels grafs consisteix en trobar subconjunts per a cada graf, anomenats clans, en els quals els elements continguts es relacionen de la mateixa manera amb tots aquells elements fora del clan. Concretament, en aquesta descomposició els grafs es divideixen en clans primers, un tipus especial de clans, i la construcció de les 2-estructures indica les relacions necessàries entre els clans primers dels grafs.

L'algoritme que divideix els grafs en clans, primer busca tots els clans del graf i després els divideix en clans trivials i clans primers. En els propers apartats s'explica detalladament els tipus de clans.

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 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 32 s'han encerclat en forma de rectangle els clans X = {A, B} i Y = {C, D} del graf G. Ho són perquè la resta de nodes del graf no distingeixen els clans X i Y. Ja que cada node està relacionat amb els clans amb la mateixa classe d'equivalència. O sigui que el color de les arestes de cada node extern als dos nodes que formen cada clan és el mateix.

Per exemple, el subconjunt {A, C} no és un clan, perquè els nodes F, E i D no distingeixen A i C amb la mateixa classe d'equivalència: (F, A) $R_g \neq$ (F, C) R_g, (E, A) $R_g \neq$ (E, C) R_g i (D, A) $R_g \neq$ (D, C) R_g.

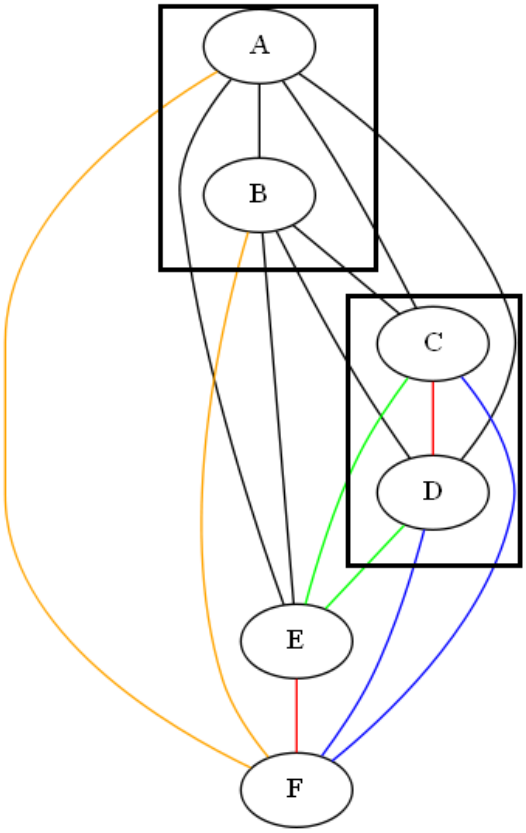


Figura 32. **Clans X i Y de l'exemple 18.**

8.3 CLANS TRIVIALS

Per a qualsevol graf, els subconjunts únics amb el format {x}, $x \in D_g$ i els que contenen tots els nodes del graf són clans. Concretament s'anomenen clans trivials. Contràriament, 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. O sigui que, tots els clans trivials també seran clans primers. Clans que explicarem en proper apartat.

Exemple 19. Considerant la figura 32 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 (sense repeticions) del graf G.

Tots els subconjunts que formen un clan a la figura 32 són: {'A'}, {'B'}, {'C'}, {'D'}, {'E'}, {'F'}, {'B', 'A'}, {'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 dels clans primers dels grafs. I un clan és primer si no compleix la propietat *d'overlapping.*

La propietat *d'overlapping* indica els clans que tenen alguns elements en comú per superposició. Per exemple, els clans A i B estan superposats si $A \cap B \neq 0$, $(A \cap B) \subset A$ i $(A \cap B) \subset B$. Per tant, els clans A i B no seran clans primers.

Un clan primer també pot incloure altres clans primers. En podem veure un exemple a continuació.

Exemple 20. Sent Y = {'A', 'B'}, Z = {'D', 'C'} i X = {'A', 'B', 'D', 'C', 'F', 'E'} clans primers, Y i Z seran clans primers de X si compleixen els casos següents:

- Primer cas: $Y \cap Z \subset X \cap Y$.

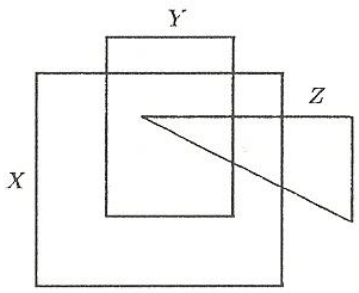


Figura 33. **Primer cas de l'exemple 20.**

- Segon cas: $Y \cap Z \subset Y / X$.

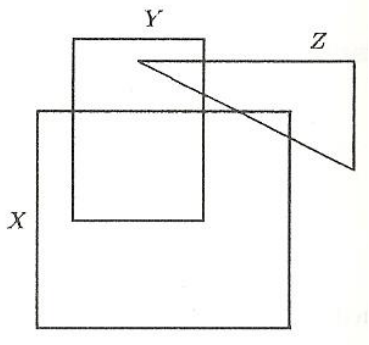


Figura 34. **Segon cas de l'exemple 20.**

8.5 CLANS MÉS FREQÜENTS

Un cas espacial del treball és la descomposició dels grafs a partir dels clans més freqüents, que estan representats pels conjunts d'elements més freqüents d'un conjunt de dades relacionals. En aquesta descomposició diferent es procedeix identificant els conjunts d'elements individuals més freqüents de la taula SQLite d'una base de dades relacional, i estenent-los a conjunts de major grandària sempre que aquests conjunts de dades apareguin suficientment seguits a la taula.

L'algoritme Apriori , que l'executa el programa Apriori , és l'encarregat de trobar els clans més freqüents quan el nombre total de clans amb el que s'ha de descomposar un graf no és físicament computable (no es pot calcular per una màquina de Turing [30].

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

9. 2-ESTRUCTURES

En aquest treball es desenvolupa la teoria de la 2-estructures, i en particular es demostra que cada 2-estructura pot ser construïda a partir de la descomposició en clans primers d'un tipus de graf. De fet, cada tipus de graf representarà el mateix tipus de 2-estructura. La descomposició resultant s'obté a través d'una representació jeràrquica en forma d'arbre. En aquest arbre cada estructura reprodueix un clan primer o un element d'aquest.

Concretament, una 2-estructura és una seqüència de valors agrupats, formada per un subconjunt finit D, anomenat domini, i una relació d'equivalència $R \subseteq E_2(D) \times E_2(D)$ en el conjunt $E_2(D) = \{(u, v) \mid u \in D, v \in D, u \neq v\}$ 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 relacions d'equivalències R les arestes acolorides del graf G:

$$G = (D, R)$$

La divisió en 2-estructures d'una 2-estructura original ha de pertànyer com a mínim a alguna de les tres subclasses especials de 2-estructures bàsiques que existeixen: completes, lineals i primitives.

- Una 2-estructura $G = (D, R)$ és completa quan només té definida una classe d'equivalència $E_2(D)$. Una estructura completa representa un graf complet. Òbviament, 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 incloses en cadascuna de les classes d'equivalències estableixen un ordre total sobre els elements del domini D.
- Una 2-estructura és primitiva quan no pot des composar-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.

Exemple 21. A la figura 35 es pot veure la representació d'una 2-estructura. En ella, cada estructura rectangular representa per un costat els clans trivials (clans primers de longitud u) en forma de nodes, i per l'altre costat la resta d'elements del clan primer que es relacionen de la mateixa manera. Els clans trivials es troben a les fulles del subarbre al que queda connectat cada clan primer representant.

Les arestes internes serveixen per a diferenciar la classe d'equivalència a la que pertany cada clan. I les arestes externes per a diferenciar els elements que formen cada un d'aquests clans. A més, cada aresta interna indica una classe d'equivalència.

En la 2-estructura de la figura 36 veiem cinc classes d'equivalències diferents: la negra, la blava, la vermella, la groga i la verda.

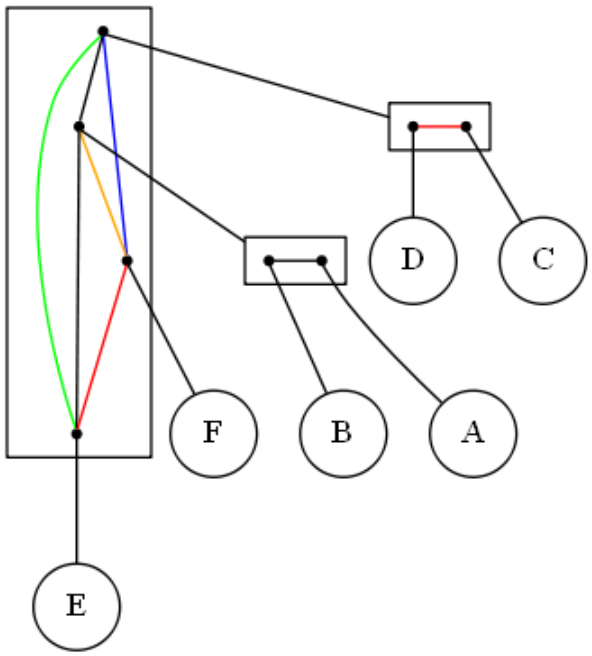


Figura 35. 2-estructura de l'exemple 21.

Com hem comentat anteriorment, una de les propietats que compleix la descomposició de les 2-estructures és que poden factoritzar en altres 2-estructures: completes, lineals o primitives. En aquest cas podem observar com la estructura rectangular més gran s'ha factoritzat en dues 2-estructures primitives.

9.1 CREACIÓ DE LES 2-ESTRUCTURES

Com s'ha anomenat en la creació dels grafs, el llenguatge *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-estrcutrues es segueix el mateix procés que en la creació dels grafs. Ja que com hem comentat moltes vegades en aquest document, un graf complet i dirigit amb les arestes acolorides es pot convertir en una 2-estructura. Per la construcció de les 2-estructures en beneficiem d'aquesta definició per crear cada 2-estructura a partir d'un graf en format *DOT* .

A la de la figura 36 es pot veure un exemple senzill d'un un 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 la figura 37.

La primera línia de l'script indica el tipus de graf i el seu nom entre cometes dobles. A més de la propietat stricto, 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 del graf: En aquest cas i en ordre:

- Permetre arestes entre els subgrafs.
- El tipus de lletra.
- La mida del tipus de lletra.
- La direcció de la posició dels subgrafs.
- L'estil dels nodes. Els nodes externs als subgraf es representen amb nodes i els interns amb punts.

De les línies 7 a 10 es mostren els diferents elements de tots els clans primers, els nodes externs a qualsevol subgraf, que indiquen els clans trivials de la 2-estructura.

Cada subgraf de la figura 36 es dibuixa mitjançant un rectangle i representa un clan primer. A dins de cada clan primer es representen les relacions existents (classes d'equivalència) entre cada element del clan.

Finalment es creen les arestes entre els elements dels subgrafs i els elements externs a ells. Aquestes arestes poden anar fins a un altre subgraf o fins un element del propi subgraf.(clan trivial).

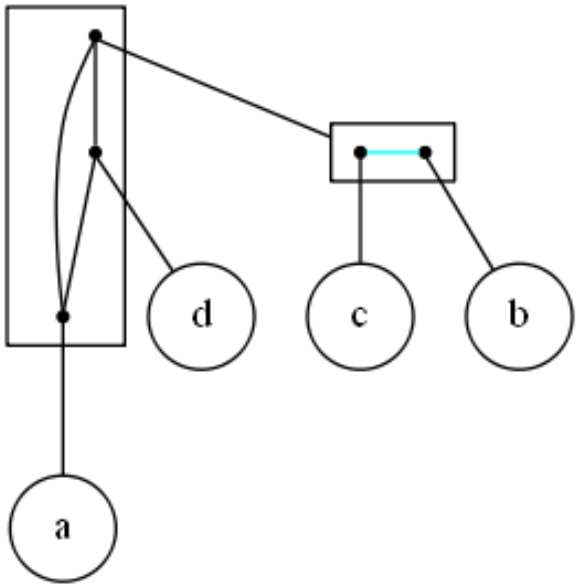


Figura 37. 2-estructura equivalent a la figura 36.

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.

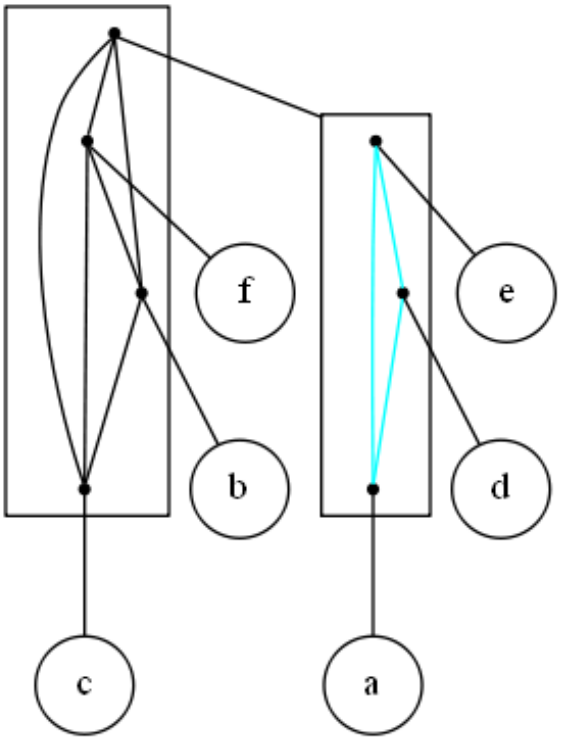


Figura 38. Limitació del llenguatge *DOT* .

Com es pot comprovar a la figura 38, les estructures en forma de rectangle que representen els clans primers o clústers d'un graf, 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 [R]. Això es deu a que els subgrafs o clústers no formen part del llenguatge *DOT* , sinó a una convenció sintàctica adherida per alguns dels motors de disseny del programari Graphviz.

10. EXEMPLE: SIMPLE

Aquest és un exemple senzill en què es mostra un procés que modela el comportament d'un usuari. A continuació es pot veure el codi que genera el model del tipus de graf i del tipus de 2-estructura equivalent que es volen crear.

```
if __name__ == '__main__':
    optionData = int(six.moves.input("Please enter the option for the type of file you provide:\n [1] = ARFF\n [2] = "
                                     "TXT\n [3] = DB\n"))

    columnNames, rows, cursor, tableName = Interface.inputFileOptions(optionData)  # Manages the data entry
    initGraph, rows = Graph.initGraph(tableName, cursor)  # Initialize the graph

    optionStructure = int(six.moves.input(
        "Please enter the option of 2-structure you want to create:\n [1] = plain\n [2] = plain "
        "with threshold\n [3] = linear\n [4] = exponential\n"))

    graph, graphName = Interface.graphOptions(optionStructure, initGraph, rows)  # Create a type of graph

    if optionData == 1 or optionData == 2:
        frequentClans = str(six.moves.input("Want to work with the more frequent clans?\nyes/no\n"))

        if frequentClans == "yes":
            support = float(six.moves.input("Please enter the probability for the more frequent subsets creation:\n"))
            moreFrequentSubsets = Subset.moreFrequentSubsets(tableName.strip("\n"), optionData,
                                                             support)  # The more frequents subsets

            structureName = Interface.structureOptions(optionStructure, graph,
                                                       moreFrequentSubsets)  # Create 2-structure from graph

        elif frequentClans == "no":
            structureName = Interface.structureOptions(optionStructure, graph, None)  # Create 2-structure from graph

    elif optionData == 3:
        structureName = Interface.structureOptions(optionStructure, graph, None)  # Create 2-structure from graph

    if nx.number_of_nodes(graph) <= 15:
        Interface.openGraphviz(graphName)  # Open graph in Graphviz program
    Interface.openGraphviz(structureName)
```

Codi 6. Codi per a generar els grafs i les 2-estructures.

En el codi s'observa el cicle del procés que es duu a terme (fitxer de dades -> graf -> descomposició del graf -> 2-estructura). Quan es crea el graf i la 2-estructura es mostra a l'usuari el resultat.

Desglossarem l'exemple més detalladament en els apartats següents.

10.1 DADES D'ENTRADA

Les dades relacionals d'un fitxer ARFF que utilitzarem per a l'exemple es converteixen en la següent taula SQLite:

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

Figura 39. **Taula de l'exemple SIMPLE.**

10.2 TIPUS DE GRAF

El tipus de graf que s'ha volgut generar a partir de les dades de la taula SQLite és un graf lineal (figura 40).

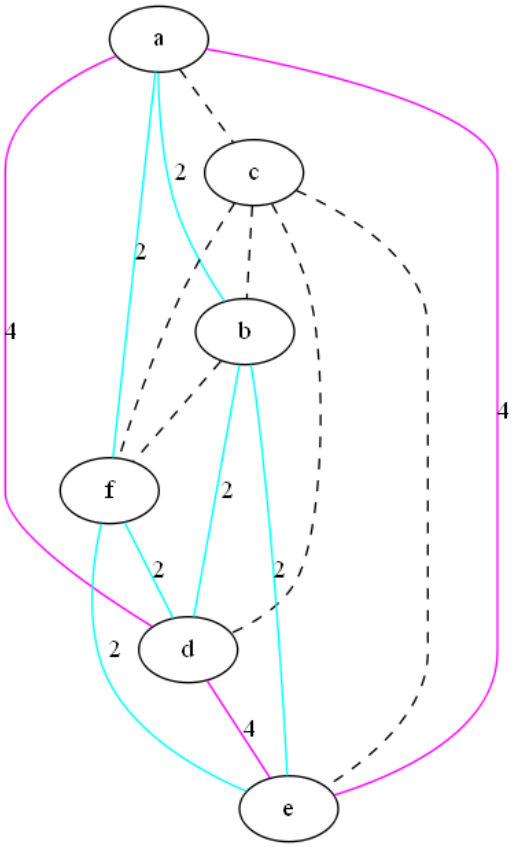


Figura 40. **Graf lineal de l'exemple SIMPLE.**

Les arestes a-c, c-f, c-b, c-d, c-e i b-c indiquen els nodes que no tenen cap relació d'equivalència a la taula. La resta d'arestes, que es troben ponderades, indiquen els nodes que tenen com a mínim una relació d'equivalència entre ells. Concretament les arestes ponderades mostren el número total de relacions d'equivalència de la taula entre cada parell de nodes. El color turquesa defineix la classe d'equivalència on el nombre total de relacions d'equivalències a la taula és igual a 2. I el color magenta defineix la classe d'equivalència on el nombre total de relacions d'equivalències a la taula és igual a 4.

Observem doncs, tres classes d'equivalències: no existeix aresta (no hi ha relació entre un parell de nodes a la taula), color turquesa (hi ha dues relacions entre un parell de nodes a la taula) i color magenta (hi ha quatre relacions entre un parell de nodes a la taula)[1]. La 2-estructura resultant tindrà tres classes d'equivalències anteriors. Sinó el resultat no seria el correcte.

[1] Veure taula d'equivalències de la pàgina 52.

10.3 DESCOMPOSICIÓ DEL GRAF

En aquest apartat es duu a terme la descomposició del graf en clans primers per a la construcció dels nodes, de les arestes i dels clústers de la 2-estructura. Per exemple, el subconjunt {'b', 'f'} és un clan ja que la resta de nodes no el distingeixen (les arestes cap a els elements 'b' i 'f de la resta són del mateix color).

Els clans primers del graf de la figura 40 són: {'a'}, {'c'}, {'b'}, {'f'}, {'d'}, {'e'}, {'b', 'f'}, {'a', 'd', 'e'}, {'b', 'f', 'd', 'a', 'e'} i {'b', 'f', 'd', 'a', 'c', 'e'}. Els clans primers {'a'}, {'c'}, {'b'}, {'f'}, {'d'}, {'e'} i {'b', 'f', 'd', 'a', 'c', 'e'} també són clans trivials perquè la seva longitud és u o perquè conté tots els elements del graf. La resta de clans primers ({'b', 'f'}, {'a', 'd', 'e'} i {'b', 'f', 'd', 'a', 'e'}) no són trivials.

També hi ha clans primers dins d'altres clans primers. Per exemple, els clan {'b', 'f'} i {'a', 'd', 'e'} són subconjunts del clan {'b', 'f', 'd', 'a', 'e'}. I aquest últim del clan trivial {'b', 'f', 'd', 'a', 'c', 'e'}.

10.4 2-ESTRUCTURA

Una vegada trobats els clans primers es dibuixa la 2-estructura amb ells. Cada clan primer, però no trivial, amb el nombre d'elements igual o superior a dos forma un clúster. A l'exemple els clans primers {'b', 'f'}, {'a', 'd', 'e'} {'b', 'f', 'd', 'a', 'e'} formaran un clúster. El clan que conté tots els elements del graf no es visualitzarà en una 2-estructura.

Cada element del clúster té forma de punt i estarà relacionat amb la classe d'equivalència corresponent del graf de la figura 40. Per exemple com que els elements del clan primer {'a', 'd', 'e'} es relacionen entre ells amb el mateix color (magenta), els nodes interns del clúster {'a', 'd', 'e'} tindran les arestes d'aquest color. De cada clúster penjaran els clans trivials que contingui en forma de nodes.

Podem veure la 2-estructura de l'exemple SIMPLE a la figura següent:

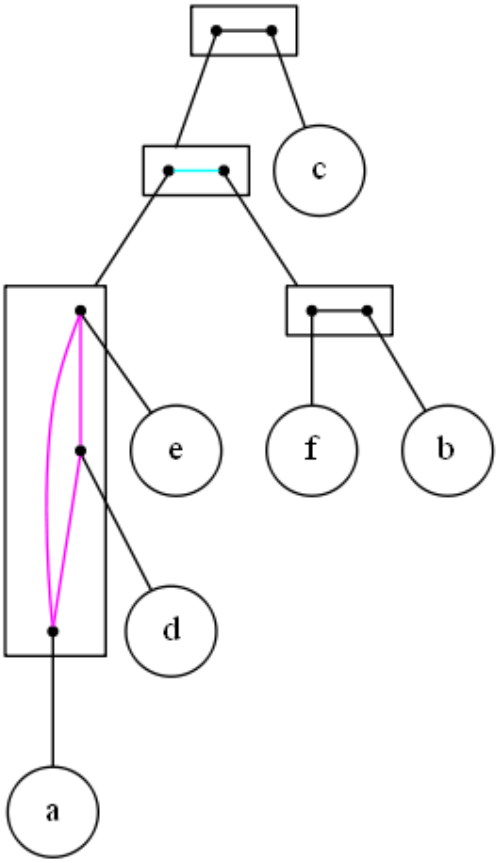


Figura 41. 2-estructura lineal de l'exemple SIMPLE.

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-ne contemplar moltes més.

Un altre punt interesant seria l'anàlisi en profunditat quan les 2-estructures es generin a partir de diferents taules SQLite.

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

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 i una part més pràctica.

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 teoria es troba l'estudi de les 2-estructures, tema totalment desconegut, que gràcies a la seva exploració m'ha resultat molt interessant.

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 i el suport que he rebut per a desenvolupar el 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] Gaifman Locality. http://wwwis.win.tue.nl/~tcalders/teaching/advancedDB09

/instructions/solutions02.pdf (Últim accés: 30/04/2017).

[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).

GLOSSARI

DOT Graph Description Language

XML eXtensible Markup Language

KISS Keep It Simple, Stupid!

HTTP Hypertext Transfer Protocol

FTP File Transfer Protocol

SSH Secure Shell

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.

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 (i els seus grafs relacionats). 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 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

Per exemple, descarregar el paquet six de https://pypi.python.org/pypi/six#downloads:

Figura 42. Lloc de descàrrega el paquet six.

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

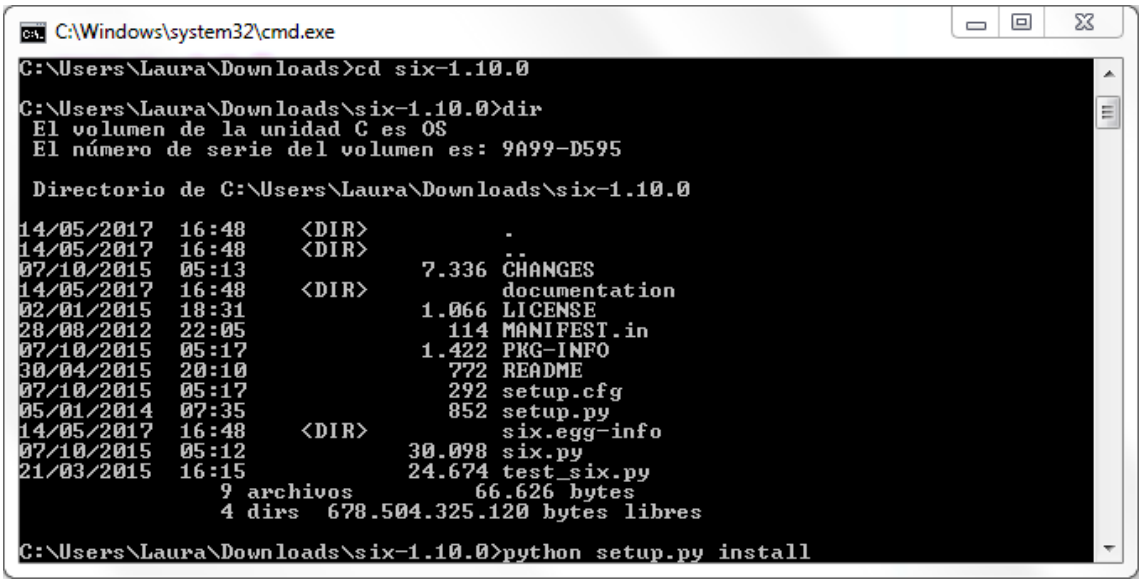


Figura 43. Terminal de Windows.

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 adminsitrador.

- 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 fer-ho 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 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 [R] per a la visualització dels grafs i les 2-estructures. Per a la instal·lació 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

Alhora de crear una nova 2-estructura s'ha de 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), i main.py. Al seleccionar 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. Recordem que els tipus de fitxers que processa el paquet de software són: ARFF, TXT i DB. I el nom d'aquest fitxer.

Es crearà la base de dades amb les dades relacionals que incorporava el fitxer proporcionat i es demanarà quin tipus de 2-estructura es vol crear. Recordem que els tipus de 2-estrcutures que existeixen per aquest paquet de software són: plana, plana amb llindar, lineal i exponencial.

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 44 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 44. Menú principal.

Una vegada creada la 2-estructura i el seu graf corresponent, apareixerà dues finestres (figures 44 i 45) on es podrà visualitzar el resultat.

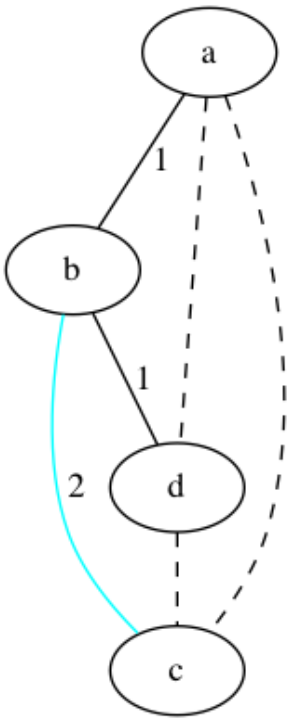


Figura 45. Graf lineal de l'exemple 22l.

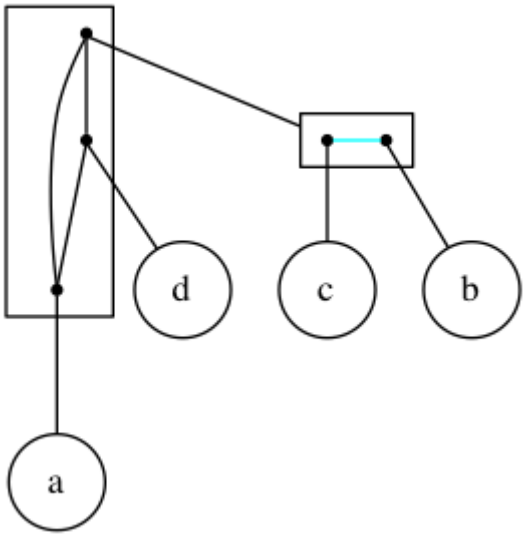


Figura 46. 2-estructura lineal de l'exemple 22.