Implementació d'algoritmes d'optimització per al disseny computacional de proteïnes

Joan Lapeyra Amat

22 de juny de 2022

Traball de Fi de Grau

Director:
Francisco Javier Larrosa Bondia
(Departament de Ciències de la Computació)

Grau en Enginyeria Informàtica

Especialitat de Computació

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Resum

El disseny computacional de proteïnes es pot modelar com a WCSP, un problema d'optimització de restriccions. WCSP es pot reduir a MaxSAT, un problema d'optimització de restriccions a variables booleanes. Aquest treball consisteix a implementar solvers per WCSP i MaxSAT i avaluar-ne l'eficiència. També implementem un visualitzador d'instàncies WCSP.

Resumen

El diseño computacional de proteínas se puede moldear como WCSP, un problema de optimización de restricciones. WCSP puede reducirse a MaxSAT, un problema de optimización de restricciones a variables booleanas. Este trabajo consiste en implementar solvers para WCSP y MaxSAT y evaluar su eficiencia. También implementamos un visualizador de instancias WCSP.

Abstract

Computational protein design can be modeled as WCSP, a constraint optimization problem. WCSP can be reduced to MaxSAT, a constraint optimization problem with boolean variables. In this project we implement solvers for WCSP and MaxSAT and test their efficiency. We also implement a WCSP instance viewer.

El codi es troba al següent repositori: https://github.com/jlapeyra/TFG

Índex

1 Introducció

Una de les àrees tecnològiques més disruptives de les últimes dècades és la bioinformàtica. Bona part dels avenços recents en biologia són gràcies a modelatges i simulacions informàtiques.

El disseny computacional de proteïnes (CPD) consisteix a trobar configuracions de proteïnes adequades per certes necessitats mèdiques o industrials. Des del punt de vista informàtic, CPD és un problema d'optimització discreta. Com a tal, es pot modelar com a *Weighted Constraint Satisfaction Problem* (WCSP) [1]. És ben sabut que WCSP es pot reduir a *Weighted Boolean Maximum Satisfiability Problem* (MaxSAT) [2].

WCSP i MaxSAT permeten representar problemes diversos en un llenguatge comú. El principal avantatge és que es podem dissenyar algoritmes genèrics (anomenats solvers) per WCSP i MaxSAT, que poden servir per resoldre qualsevol problema concret que es pugui reduir a WCSP i MaxSAT. Sovint aquests solvers són més eficients que solucions específiques per cada tipus de problema concret.

La figura 1 mostra el procés que implica les reduccions entre problemes que hem esmentat. Transformem el problema concret (CPD) en abstracte (WCSP o, si s'escau, MaxSAT a través de WCSP) per tal de resoldre'l. Posteriorment transformem la solució abstracta en concreta.

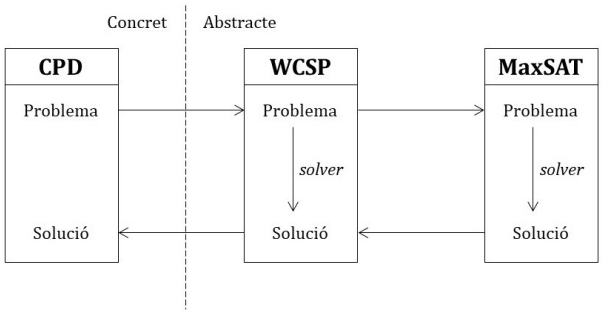

Figura 1: Procés que segueix el disseny computacional d'una proteïna amb el mètode d'aquest treball.

WCSP i MaxSAT poden servir per resoldre altres problemes més enllà de CPD, tal com il·lustra la figura 2. WCSP també pot servir per estudis de genètica [3], per la gestió de les observacions de satèl·lits [4] i per l'assignació de ràdiofreqüències [5]. Per la seva banda, MaxSAT pot servir per la gestió de paquets de software [6], el disseny de debug [7], el problema max-clique [8] i raonament probabilístic [9], entre altres.

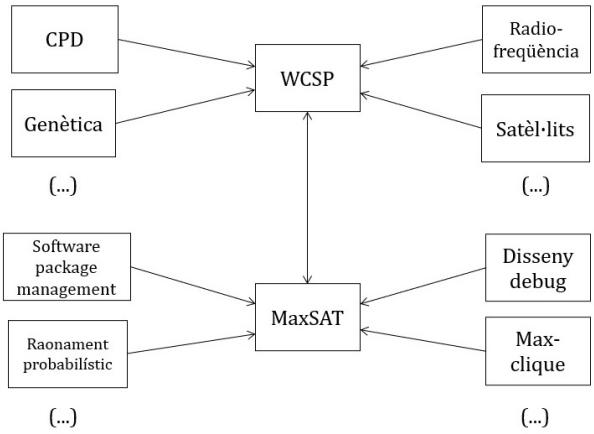

Figura 2: Altres possibilitats que ens ofereixen WCSP i MaxSAT. Cada fletxa representa la reducció d'un problema.

Els solvers de referència actuals tant per WCSP com per MaxSAT combinen un conjunt de tècniques sofisticades, però el seu esquema bàsic sol seguir una estratègia de branch-and-bound

(per WCSP) o d'extracció de cores (per MaxSAT). En els últims anys, però, s'ha vist que una tècnica anomenada Implicit Hitting Sets pot ser competitiva per MaxSAT. [10]

L'objectiu d'aquest projecte és fer una implementació pròpia de solvers basats en Implicit Hitting Sets tant per MaxSAT com per WCSP i provar el seu rendiment en problemes de CPD.

La metodologia que hem seguit és l'aplicació del mètode científic adaptada al context d'una enginyeria. La hipòtesi de partida és que el mètode d'Implicit Hitting Sets pot ser útil i competitiu per resoldre problemes CPD. Per verificar la hipòtesi hem dissenyat i implementat diverses versions de l'algoritme, que podem considerar prototips. Hem comparat el rendiment dels prototips del nostre solver amb solvers de referència en un conjunt de 109 instàncies que s'han fet servir com a benchmarks a diferents publicacions científiques.

Els resultats obtinguts han estat moderadament satisfactoris. En general, els nostres solvers no són més eficients que els solvers de referència, però sí que poden resoldre instàncies petites en temps competitius. Si tenim en compte que les nostres implementacions estan poc optimitzades i que encara tenen marge de millora, considerem que el projecte ha estat productiu.

Com és sabut, el mètode científic no és l'execució lineal d'una seqüència de tasques, sinó un bucle on cada iteració aporta idees, coneixement i intuïcions per la següent.

1.1 Breu definició dels problemes

1.1.1 Weighted Constraint Satisfaction Problem (WCSP)

Suposem que tenim un conjunt de variables. A cada variable podem assignar-li una sèrie de valors (cadascuna té associat un domini discret). Cada assignació (al conjunt de variables) té un cost associat. Per exemple, podríem tenir les variables $\{x, y, z\}$ i que l'assignació $[x = 5, y = 2, z = 3]$ tingués cost 25.

El cost ve determinat pel sumatori d'un conjunt de funcions de cost locals. Per exemple, si les funcions són $f(x, y)$, $g(z)$ i $h(x, y, z)$, el cost serà $f(x, y) + g(z) + h(x, y, z)$.

El problema WCSP consisteix a trobar una assignació que minimitzi el cost.

Una instància WCSP ve definida per:

- Les variables amb els seus dominis
- Les funcions de cost

1.1.2 Weighted Boolean Maximum Satisfiability Problem (MaxSAT)

Suposem que tenim un conjunt de variables booleanes. A cada variable podem assignar-li *cert* o $fals$. Cada assignació (al conjunt de variables) té un cost associat. Per exemple, podríem tenir les variables $\{x, y, z\}$ i que l'assignació $[x = fals, y = fals, z = cert]$ tingués cost 40.

Tenim un conjunt de clàusules com ara $\{(x \vee y), (\neg x \vee y \vee z), (\neg z)\}$. Cada clàusula té associat un cost. El cost d'una assignació és la suma dels costos de les clàusules violades. Per exemple, l'assignació $[x = fals, y = fals, z = cert]$ violaria les clàusules $(x \vee y)$ i $(\neg z)$. El cost de dita assignació seria $cost(x \vee y) + cost(\neg z)$.

El problema MaxSAT consisteix a trobar una assignació que minimitzi el cost.

Una instància MaxSAT ve definida per:

- Les clàusules sobre un conjunt de variables
- El cost associat a violar cada clàusula

1.1.3 Disseny computacional de proteïnes (CPD)

Les proteïnes són grans mol·lècules presents a la majoria de funcions biològiques. Al llarg de milions d'anys la selecció natural s'ha encarregat d'optimitzar-ne la configuració per tal d'adequar-les a les necessitats de l'organisme. Si volem adequar-les a certes necessitats biomèdiques o industrials podem dissenyar-les computacionalment. [1]

Les proteïnes són cadenes d'aminoàcids. Hi ha 20 tipus d'aminoàcids. El disseny computacional de proteïnes (CPD) determina la conformació d'un subconjunt d'aminoàcids de la proteïna. Es volen proteïnes en l'estat més estable, és a dir, que tinguin la mínima energia. Per tant, un algoritme CPD ha de minimitzar l'energia.

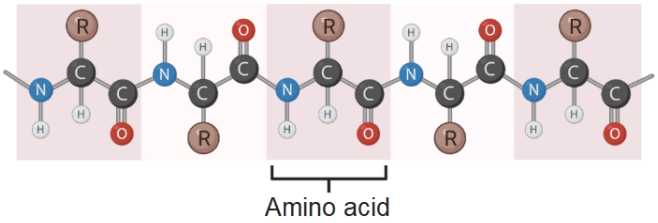

Figura 3: Estructura química del fragment d'una proteïna. L'únic component variable de cada aminoàcid és la cadena lateral (R), per tant, el CPD s'encarrega de determinar la configuració de les cadenes laterals. Font: [11].

1.2 Abast

Ens hem centrat exclusivament en els solvers de WCSP i MaxSAT. Hem partit d'instàncies WCSP publicades en articles científics, de manera que ens hem estalviat la reducció de CPD a WCSP. Tampoc hem implementat la transformació de WCSP a MaxSAT sinó que hem utilitzat un algoritme existent.

Hem partit d'una sèrie d'instàncies WCSP, la majoria de les quals són de CPD. Les hem transformat a MaxSAT amb l'algoritme existent. Hem implementat un solver per WCSP i un per MaxSAT i hem experimentat amb les instàncies esmentades. No hem interpretat les solucions sinó que ens hem fixat en el temps d'execució per avaluar l'eficiència, i en el cost de la solució per comprovar la correctesa.

Per tant, l'esquema introduït a la figura 1 queda reduït al de la figura 4.

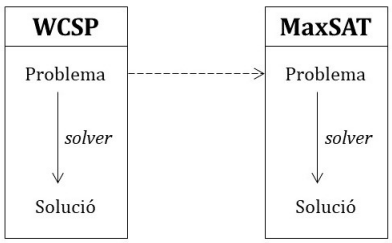

Figura 4: Abast del projecte. Les fletxes en línia contínua representen els solvers que hem implementat.

1.3 Objectiu

L'objectiu genèric del treball és tenir un estudi empíric que compari l'eficiència dels solvers implementats en aquest treball (per WCSP i per MaxSAT) amb l'eficiència de solvers existents, en la resolució d'una sèrie d'instàncies, majoritàriament instàncies CPD.

1.4 Metodologia

Com que és un projecte de recerca, hem seguit el mètode científic.

- **Observació**. Les instàncies CPD es poden modelar com a WCSP. WCSP es pot reduir a MaxSAT
- **Hipòtesi**. Si implementem WCSP amb la tècnica d'Implicit Hitting Sets, podrem resoldre instàncies CPD més ràpidament que amb altres solvers. Creiem que MaxSAT és menys eficient que WCSP per resoldre CPD.
- **Implementació**[1]. Implementem WCSP i MaxSAT amb la tècnica d'Implicit Hitting Sets.
- **Experimentació**. Un cop implementats els algoritmes, els executem amb instàncies CPD per triar la millor versió de cadascun i comparar-ne l'eficiència amb solvers existents.

1.4.1 Observació

El fet que el disseny computacional de proteïnes es pugui expressar en forma de WCSP és una idea que s'ha plantejat en diversos articles científics, que hem analitzat a la secció 2.4.1. També hem analitzat l'estat de l'art de MaxSAT (2.2.4) i WCSP (2.3.4). És ben sabut que WCSP es pot reduir a MaxSAT, tal com es menciona als articles que hem analitzat.

1.4.2 Hipòtesi

La hipòtesi principal és que el mètode d'Implicit Hitting Set serveixi per millorar l'estat de l'art de WCSP i MaxSAT. Com que en estudis anteriors s'ha demostrat que és bona idea reduir CPD a WCSP, esperem que la implementació de WCSP amb Implicit Hitting Set millori l'estat de l'art de CPD. En canvi, reduir CPD a MaxSAT no sembla una bona idea a priori, ja que les instàncies CPD tenen variables amb dominis molt grans, cosa que implica que la reducció a MaxSAT (on les variables són booleanes) tingui moltes variables.

1.4.3 Implementació

La taula 1 indica amb quin algoritme hem resolt cada problema i quins subproblemes inclou. A continuació en donem més detalls.

Problema	Algoritme	Subproblemes
MaxSAT	IMHS	MHS, CoreSAT
WCSP	IMHV	MHV, CoreCSP

Taula 1: Resum de la implementació de MaxSAT i WCSP.

1.4.3.1 Implementació de MaxSAT

Quan no hi ha cap assignació que satisfaci totes les clàusules, Volem trobar una assignació tal que les clàusules que violi sumin el mínim cost possible.

Un *core* d'una fórmula MaxSAT és un subconjunt de clàusules insatisfactible, és a dir, que no hi ha cap assignació que satisfaci totes les clàusules del core. Qualsevol core inclou almenys una clàusula que haurà de ser violada a la solució final. En altres paraules, el conjunt de clàusules violades comparteix clàusules amb tots el cores (no és disjunt amb cap core). Amb la terminologia

[1] La implementació no és un pas estàndard del mètode científic però en el nostre cas és necessari per poder experimentar.

del treball, diem que el conjunt de clàusules violades per la solució *hiteja* tots els cores. A més, és de cost mínim, per tant, és el *Minimum Cost Hitting Set* (MHS) del conjunt de cores. MHS és un problema clàssic en combinatòria i computació del qual se n'han fet diversos estudis i solvers.

La quantitat de cores d'una fórmula MaxSAT pot ser molt alta i no és bona idea precalcular-los tots per poder computar MHS. En canvi, hem aplicat l'algoritme *Implicit MHS* (IMHS), que assumeix que no disposem de tots els cores sinó que són implícits per la semàntica de MaxSAT. L'algoritme va iterant de manera que a cada iteració troba cores nous mitjançant una funció anomenada CoreSAT. A les iteracions també calcula el MHS dels cores trobats fins al moment. No és necessari trobar tots els cores per obtenir la solució.

La funció CoreSAT és un SAT solver que retorna cores quan la funció és insatisfactible. El problema SAT és la versió de decisió de MaxSAT. Mentre MaxSAT consisteix a trobar una assignació que minimitzi el cost de les clàusules violades, SAT consisteix a trobar una assignació que satisfaci totes les clàusules. En cas que no sigui possible, retorna un core, és a dir, un subconjunt de clàusules que per si sol ja és insatisfactible. En el nostre cas, a cada crida de CoreSAT li passem les clàusules de MaxSAT excepte les de l'últim MHS trobat. L'execució acabarà quan CoreSAT retorni satisfactible.

Ho hem programat en Python per tal de disposar de la llibreria *PySAT*, que ens ha ajudat a resoldre els subproblemes MHS i CoreSAT.

Pel que fa CoreSAT hem utilitzat un SAT solver de la llibreria *PySAT* i el core s'ha calculat a través d'assumpcions. Pel que fa MHV hem programat tres versions: un solver de la llibreria *PySAT* (*Hitman*); reducció de MHV a programació lineal 0/1 i resolució amb el solver *cplex*, i implementació manual amb un algoritme recursiu. De l'algoritme IMHS per MaxSAT n'hem programat dues versions, anomenades *lb* i *lub*.

1.4.3.2 Implementació de WCSP

Per resoldre una instància WCSP volem trobar una assignació que minimitzi la suma de les funcions de cost.

Definim l'enduriment d'una funció de cost amb un cert llindar com una *restricció* que se satisfà quan el resultat de la funció és inferior o igual al llindar. L'enduriment d'una fórmula WCSP amb un vector de llindars (un llindar per cada funció) és l'enduriment de totes les funcions, cadascuna amb el llindar corresponent. Per tant, l'enduriment de WCSP ja no és un conjunt de funcions de cost sinó un conjunt de restriccions, és a dir, el *Weighted Constraint Satisfaction Problem* (WCSP) esdevé *Constraint Satisfaction Problem* (CSP). Noteu que com més petits siguin els llindars, més restringit quedarà el CSP. El problema CSP és la versió de decisió de WCSP. La solució a un CSP és una assignació que satisfaci totes les restriccions.

A una solució de WCSP podem associar-li un vector en què cada component correspon al resultat d'una funció de cost amb l'assignació de la solució. L'anomenarem *vector de la solució*. Noteu que la suma dels seus components es igual al cost de la solució i que, per tant, ha de ser mínima. Diem que el cost d'un vector és la suma dels seus components.

Un *soft core* d'una fórmula WCSP és un vector tal que si endurim el WCSP amb el vector, obtenim un CSP insatisfactible. El vector de la solució és més gran que qualsevol soft core en almenys un component. En la nomenclatura del treball direm que el vector de la solució *hiteja* tots els cores. A més, ha de ser de cost mínim, per tant és el *Minimum Cost Hitting Vector* (MHV) del conjunt de cores.

La quantitat de cores d'una fórmula WCSP pot ser molt alta i no és bona idea precalcular-los tots per poder computar MHV. En canvi, hem aplicat l'algoritme *Implicit MHV* (IMHV), que assumeix que no disposem de tots els cores sinó que són implícits per la semàntica de WCSP. L'algoritme va iterant de manera que a cada iteració troba cores nous mitjançant una funció anomenada CoreCSP. A les iteracions també calcula el MHV dels cores trobats fins al moment. No és necessari trobar tots els cores per obtenir la solució.

La funció CoreCSP és un CSP solver que proporciona cores quan la funció és insatisfactible. A cada

crida, el CSP que li passem a la funció és el WCSP endurit amb l'últim MHV trobat. L'execució acabarà quan CoreCSP retorni satisfactible.

Ho hem programat en C++ perquè és un llenguatge bastant eficient.

Hem fet dues versions de CoreCSP: una consisteix a reduir CSP a SAT i resoldre amb el solver *Cadical*; l'altre és implementar manualment l'algoritme Forward Checking per CSP. Pel que fa MHV, l'hem reduït a programació lineal 0/1 i l'hem resolt amb el solver *cplex*. De l'algoritme IMHV per WCSP n'hem programat dues versions, anomenades *lb* i *lub*. La versió *lub* necessita una funció que retorni un *Hitting Vector* (que no cal que sigui mínim): n'hem programat tres versions, anomenades *greedy*, *model* i *callback*.

1.4.3.3 Visualització d'instàncies WCSP

Les instàncies WCSP són fitxers de text amb caràcters numèrics. De per si són difícils de llegir per un humà. A més normalment són molt grans, amb milions de caràcters i diversos MBytes. Hem programat un visualitzador d'instàncies WCSP que ajuda a l'usuari fer-se una idea de com és una instància WCSP.

El visualitzador proporciona dades numèriques (nombre de variables, nombre de funcions de cost, aritat de les funcions, mida dels dominis...), gràfics (costs de les funcions segons l'aritat, presència de costs infinits a les funcions segons l'aritat), el graf d'interacció i la descomposició en arbre. El graf d'interacció és un graf on hi ha un vèrtex per cada variable i una aresta entre dos vèrtexs si les dos variables són entrades d'una mateixa funció. La descomposició en arbre és una manera de resumir el graf d'interacció.

1.4.4 Experimentació

L'objectiu dels experiments és avaluar l'eficiència i correctesa dels nostres solvers i comparar-los amb solvers de referència. Per avaluar la correctesa, hem comprovat que el cost de la solució sigui al mateix en els nostres solvers que en els solvers de referència. Per avaluar l'eficiència ens hem fixat en el temps d'execució.

El nostre benchmark està format per 109 instàncies de WCSP publicades en articles científics. Les podem dividir en quatre grups:

- **CP13**. 16 instàncies. Publicades a [12] (conferència Constraint Programming, 2013).
- **AIJ14**. 47 instàncies. Publicades a [13] (revista Artificial Intelligence Journal, 2014).
- **Bio13**. 25 instàncies. Publicades a [14] (revista Bioinformatics, 2013).
- **UAI17**. 21 instàncies. Publicades a [15] (conferència Uncertainty in Artificial Intelligence, 2017).

Les instàncies de AIJ14, Bio13 i UAI17 són de disseny computacional de proteïnes. Les de CP13 són significativament més petites i representen dos problemes: inferència probabilística pel lligament genètic i gestió de satèl·lits.

Pels experiments de MaxSAT hem traduït les instàncies de WCSP a MaxSAT mitjançant una variació de l'script [16].

Els solvers de referència amb què compararem els nostres són:

- ***toulbar2*** [17, 18]. Solver de WCSP. Implementat en C++.
- ***RC2*** [19]. Solver de MaxSAT. Implementat en Python, igual que el nostre solver de MaxSAT.
- ***MaxHS*** [20]. Solver de MaxSAT. Implementat en C++. Igual que nosaltres, utilitza la tècnica d'Implicit Hitting Set. [21]

Hem fixat un temps límit d'una hora per cada execució. Si no s'ha trobat una solució en aquest temps, s'avorta l'execució.

Com hem vist a l'apartat anterior, per certes part del codi hem programat diverses versions. Combinant les versions obtenim 8 prototips per WCSP i 8 per MaxSAT.

La fase preliminar de l'experimentació consisteix en provar tots els prototips amb les 16 instàncies petites de CP13. En aquesta fase descartem els prototips clarament ineficients. Hem descartat la meitat de prototips. La fase final d'experimentació consisteix a executar els prototips seleccionats (4 per WCSP, 4 per MaxSAT) amb totes les 109 instàncies. També hem executat els solvers de referència amb les 109 instàncies.

Les figures 5 i 6 mostren els resultats dels experiments de la fase final.

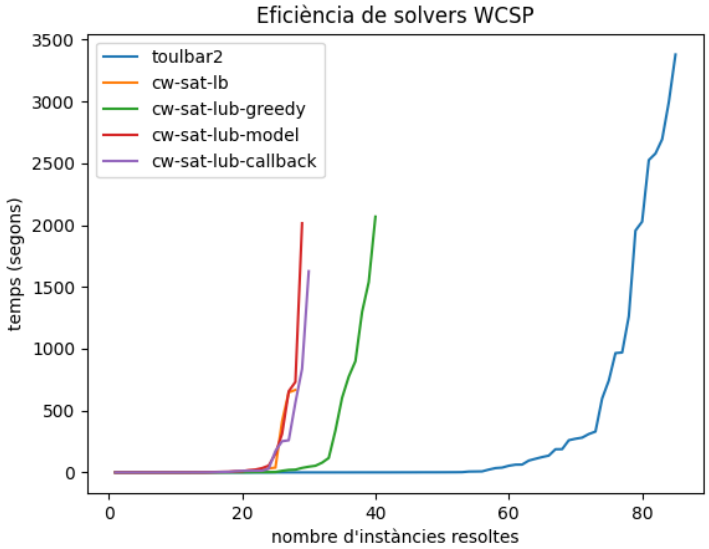

Figura 5: Resultats de WCSP per les 109 instàncies del solver de referència (*toulbar2*) i 4 prototips del nostre solver (*cw-**)

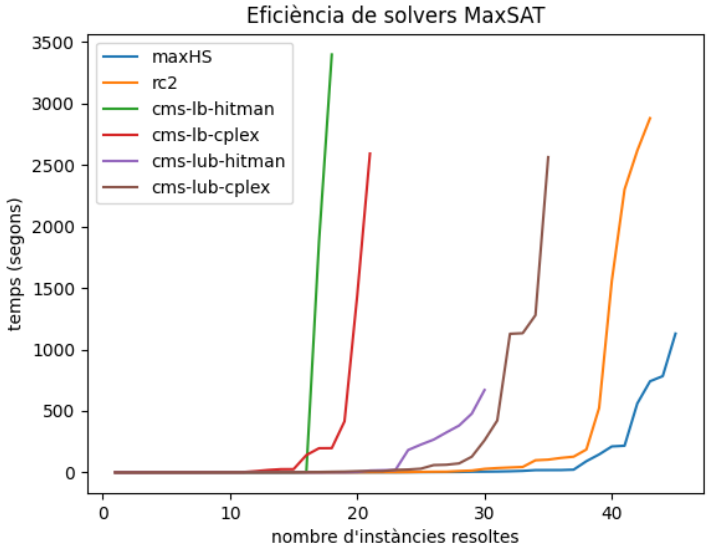

Figura 6: Resultats de MaxSAT per les 109 instàncies dels solvers de referència (*maxHS*, *rc2*) i 4 prototips del nostre solver (*cms-**)

Com podem veure, els solvers de referència són més eficients que els nostres prototips. Concretament, *toulbar2* és capaç de resoldre 85 instàncies WCSP en menys d'una hora, mentre que el nostre

millor prototip WCSP (*cw-sat-lub-greedy*) n'ha resolt 40. Pel que fa MaxSAT, *MaxHS* i *RC2* han resolt 45 i 43 instàncies (respectivament) i el nostre millor prototip MaxSAT (*cms-lub-cplex*) n'ha resolt 35.

La part positiva és que sembla que el nostre solver WCSP seria més eficient que *toulbar2* per resoldre instàncies petites. A la figura 7 veiem que pel conjunt d'instàncies CP13 els 4 prototips són més eficient que *toulbar2*. Curiosament, aquestes instàncies no són de CPD.

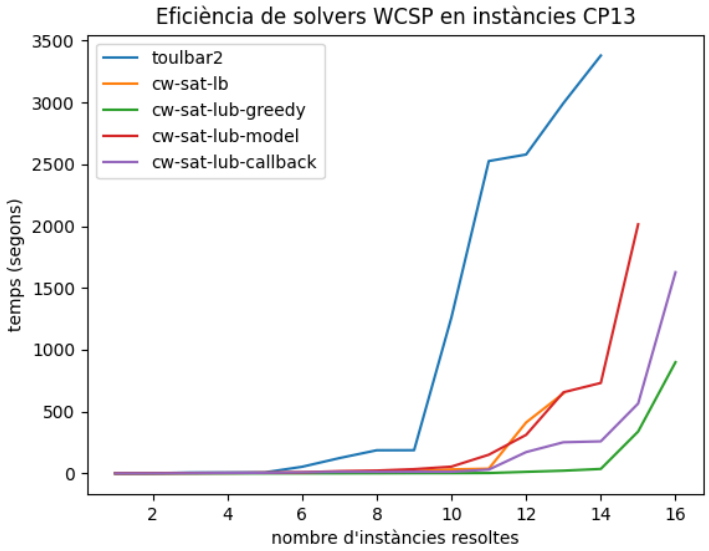

Figura 7: Resultats de WCSP per les 16 instàncies CP14, del solver de referència (*toulbar2*) i 4 prototips del nostre solver (*cw-**)

Si comparem les figures 5 i 6, veiem que en general s'han resolt més instàncies WCSP que MaxSAT, la qual cosa té sentit perquè les instàncies s'han codificat originalment en WCSP. En certa manera, també reforça la hipòtesi de que MaxSAT és menys eficient que WCSP per CPD.

1.5 Context del projecte

Aquest projecte és el Treball de Final de Grau de Joan Lapeyra, realitzat a la Facultat d'Informàtica de Barcelona (FIB), de la Universitat Politècnica de Catalunya (UPC). Està dirigit pel professor Javier Larrosa i hi col·labora la professora Emma Rollon. El treball està fet en col·laboració amb l'àrea de recerca del departament de Ciències de la Computació de la UPC.

2 Definicions i treball relacionat

2.1 Programació amb restriccions

La programació amb restriccions (*constraint programming*, CP) és un paradigma per resoldre problemes combinatoris. La idea bàsica és que l'usuari especifica les restriccions i un solver de propòsit general resol el problema. Aquesta disciplina té múltiples aplicacions: planificació, encaminament de vehicles, configuració, xarxes, bioinformàtica, etc. [22]

La programació amb restriccions pren les seves arrels i es pot expressar en forma de programació lògica amb restriccions (*constraint logic programming*). Aquesta variant de la programació lògica va ser introduïda el 1987 per Jaffar i Lassez. [23, 24]

Actualment cada any se celebra una conferència internacional anomenada *Principles and Practice of Constraint Programming*, més coneguda com a CP. És el principal esdeveniment per presentar investigacions en tots els aspectes de la informàtica amb restriccions: teoria, algoritmes, entorns, llenguatges, models, sistemes, aplicacions, etc. Iniciada el 1995, fins a dia d'avui se n'han celebrat 27 edicions. Cada any se celebra en una ciutat diferent. [25]

S'han estudiat gran varietat de problemes amb restriccions. Els podem classificar en dos grups: de satisfacció i d'optimització. La missió dels primers (que són problemes de decisió) és trobar una assignació que compleixi totes les restriccions. La missió dels segons és trobar una assignació que minimitzi (o maximitzi) una funció de cost. En els problemes de satisfacció, una solució és correcta o incorrecta. En canvi, en els d'optimització, cada solució té un cost i l'objectiu és trobar l'òptima o una que s'acosti a l'òptima. En aquest treball resolem dos problemes d'optimització (WCSP i MaxSAT) i en cadascun dels algoritmes fem una crida a un problema de satisfacció (CSP i SAT, respectivament) adaptat per les necessitats de l'algoritme.

També els podem classificar segons el domini de les variables. En aquest treball hem vist problemes amb domini booleà (SAT i MaxSAT) i dominis enters (CSP i WCSP).

	satisfacció	optimització
domini booleà	SAT	MaxSAT
dominis enters	CSP	WCSP

Taula 2: Classificació de quatre problemes de restriccions.

Altres problemes d'optimització de restriccions que apareixen a aquest treball són MHS i MHV. Són subproblemes dels nostres algoritmes. Els hem resolt de diverses maneres, una de les quals consisteix a reduir-los a programació lineal 0/1, que també és un problema de restriccions.

Els noms dels problemes són una convenció i en algun cas un mateix problema ha rabut més d'un nom. Notablement, moltes publicacions anomenen CFN al que nosaltres anomenem WCSP.

Entre publicacions també poden variar els detalls de la definició de cada problema. Per exemple, en aquest treball, quan parlem de MaxSAT ens referim a MaxSAT *amb pesos*, però hi ha publicacions publicacions, com ara [26], que distingeixen entre "MaxSAT" (sense pesos) i "Weighted MaxSAT" (amb pesos). Podem considerar que MaxSAT sense pesos és un cas particular de MaxSAT on tots els pesos són 1, és a dir, totes les clàusules tenen el mateix cost.

2.2 MaxSAT

2.2.1 Definició de SAT

Sigui X un conjunt de variables booleanes. Un literal positiu és una variable booleana $x \in X$. Un literal negatiu és una variable booleana negada $\neg x$. Una clàusula és una disjunció de literals (p. ex. $x \vee \neg y \vee z$). Una fórmula SAT (en forma normal clausal, CNF) és un conjunt de clàusules. Diem que una fórmula F és satisfactible si hi ha una assignació que satisfà totes les clàusules. Un *core* d'una fórmula insatisfactible F és un subconjunt de F insatisfactible (i.e. $k \subseteq F$ és un core de F si $UNSAT(k)$). [2]

Un SAT solver és una funció $SAT(F)$ que retorna cert si F és satisfactible i fals en cas contrari. Alguns SAT solvers retornen un core quan la fórmula és insatisfactible.

EXEMPLE: $F_1 = \{(x \vee y), (x \vee \neg y), (\neg y)\}$ és satisfactible perquè existeix una assignació $[x = cert, y = fals]$ que satisfà F_1.

EXEMPLE: $F_2 = \{(x), (\neg x), (\neg x \vee y)\}$ és insatisfactible perquè no existeix cap assignació que satisfaci F_2. A més, $k = \{(x), (\neg x)\}$ és un core de F_2 perquè $k \subseteq F_2$ i k és insatisfactible.

2.2.2 Definició de MaxSAT

MaxSAT és la versió d'optimització de SAT, en què cada clàusula c té un cost associat $cost(c) \in \mathbb{N} \cup \{\infty\}$, que és el cost de violar-la. Una fórmula MaxSAT (en CNF) és una fórmula SAT (en CNF) amb els costs associats a cada clàusula. Sigui F una fórmula MaxSAT. Sigui X una assignació a F. Sigui $cost_F(X)$ la següent funció de cost:

$$cost_F(X) = \sum_{c \in F} c[X]$$

on $c[X] = 0$ si X satisfà c, i $c[X] = cost(c)$ si X viola c.

El problema MaxSAT consisteix en minimitzar $cost_F(X)$,

$$MaxSAT(F) = \min_X cost_F(X)$$ [3]

Anomenem clàusules *hard* a les clàusules amb cost infinit, que s'han de complir sempre. Anomenem clàusules *soft* a les clàusules amb cost finit.

EXEMPLE: Suposem una funció MaxSAT $F = \{(x), (\neg x), (\neg x \vee y)\}$ amb costos $cost(x) = 3$, $cost(\neg x) = 2$, $cost(\neg x \vee y) = 7$. Llavors, $MaxSAT(F) = X = [x = cert, y = cert]$, que és l'assignació que minimitza $cost_F$. X només viola la clàusula $(\neg x)$, per tant, $cost_F(X) = (x)[X] + (\neg x)[X] + (\neg x \vee y)[X] = 0 + cost(\neg x) + 0 = 0 + 2 + 0 = 2$.

2.2.3 Exemple pràctic

A continuació veurem dos problemes que es poden modelar com a SAT i MaxSAT.

Un *clique* d'un graf no dirigit és un subconjunt de vèrtexs tal que cada parell de vèrtexs són adjacents. Formalment, $C \subseteq V$ és un clique de $G = (V, E)$ si $\forall u, v \in C : \{u, v\} \in E$. La figura 8 mostra un graf amb un clique.

El problema de trobar un clique en un graf es pot expressar com a SAT. Per cada vèrtex $v \in V$ hi ha una variable booleana x_v que indica si v és al clique. Per cada parell de vèrtexs $u, v \in V$ no adjacents hi ha una clàusula $\neg x_u \vee \neg x_v$ indicant que no pot ser que els dos vèrtexs formin part del clique. Formalment, la fórmula SAT per trobar un clique és:

$$F = \{\neg x_u \vee \neg x_v \mid u, v \in V; \{u, v\} \notin E\}$$

[2] $SAT(F)$ significa que F és satisfactible. $UNSAT(F)$ significa que F és insatisfactible.

[3] $MaxSAT(F)$ pot referir al cost mínim ($\min_X cost_F(X)$) o a l'assignació de cost mínim ($\text{argmin}_X cost_F(X)$).

on $\forall v \in V : (x_v \Leftrightarrow v \in C)$

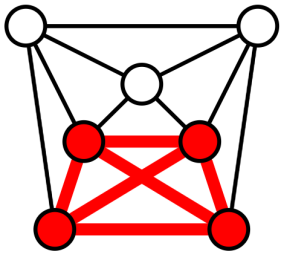

Figura 8: Graf amb un clique ressaltat. Font: [27].

Un *max-clique* d'un graf no dirigit és un *clique* amb el màxim nombre de vèrtexs. El problema de trobar un max-clique es pot expressar com a MaxSAT. Les clàusules de la fórmula SAT tindrien cost infinit, indicant que s'han de complir obligatòriament (clàusules hard). Caldria afegir una clàusula x_v amb cost 1 per cada vèrtex $v \in V$ (clàusules soft). D'aquesta manera, com menys clàusules soft es violin, més vèrtexs hi haurà al clique. Complir totes les clàusules hard implica que els vèrtexs són un clique; minimitzar el cost (de les clàusules soft) implica que hi ha el màxim nombre de vèrtexs. Formalment, la fórmula MaxSAT amb costos és:

$$F = \{(\neg x_u \vee \neg x_v, \infty) \mid u, v \in V; \{u, v\} \notin E\} \cup \{(x_v, 1) \mid v \in V\}$$

on $\forall v \in V : (x_v \Leftrightarrow v \in C)$

EXEMPLE: La funció MaxSAT per trobar un max-clique al graf de la figura 9 estaria formada per les clàusules soft $\{x_1, x_2, x_3, x_4, x_5\}$ (cost 1) i les clàusules hard $\{(\neg x_1 \vee \neg x_3), (\neg x_1 \vee \neg x_4), (\neg x_2 \vee \neg x_4), (\neg x_3 \vee \neg x_5)\}$ (cost ∞).

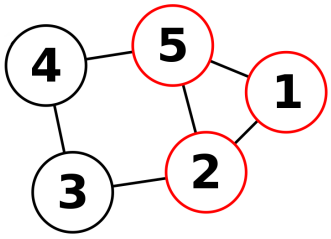

Figura 9: Graf $G = (V, E)$ amb $V = \{1, 2, 3, 4, 5\}$ i $E = \{\{1, 2\}, \{1, 5\}, \{5, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}\}$. El max-clique de G és $C = \{1, 2, 5\}$. Font: [27].

2.2.4 Estat de l'art

D'entre tots els problemes NP-complets, segurament el més estudiat és SAT. No és estrany, doncs, que també s'hagin fet molts estudis sobre MaxSAT, la versió d'optimització de SAT. També se n'han desenvolupat gran varietat de solvers.

Des del 2006, cada any se celebra la *MaxSAT Evaluation*, [28] on es presenten solvers que resolen MaxSAT. És organitzada en col·laboració entre universitats (actualment les de Carnegie Mellon, Helsinki i Toronto) i cada any de celebra en una ciutat diferent d'arreu del món, el 2021 a Barcelona. La *MaxSAT Evaluation* està afiliada a la *International Conference on Theory and Applications of Satisfiability Testing*, més coneguda com a SAT.

La taula 3 i la figura 10 mostren resultats de *MaxSAT Evaluations* recents en la modalitat de MaxSAT amb pesos.

Per aquest treball, hem pres els solvers *MaxHS* i *RC2* com a referència per comparar amb els que hem implementat nosaltres.

MaxHS utilitza la tècnica d'Implicit Hitting Sets. De fet, l'algoritme és molt semblant al nostre [21]. És la posada en pràctica d'aquesta tècnica per MaxSAT. Aquesta tècnica fou proposada a

	2017	2018	2019	2020	2021
#1	MaxHS	RC2-B	RC2-2018	UWrMaxSat	CASHWMaxSAT
#2	MaxHS-MSE16	RC2-A	UWrMaxSAT	MaxHS	MaxHS
#3	QMaxSAT	MaxHS	MaxHS	RC2-B	UWrMaxSAT
#4	QMaxSATuc	Pacose	QMaxSAT2018	RC2-A	EvalMaxSAT-fastMinimize
#5	maxino	QMaxSAT	maxino2018	maxino	EvalMaxSAT-fullMinimize

Taula 3: Rànquing dels 5 solvers que més instàncies van resoldre a la *Weighted Complete Track* de les últimes 5 edicions de la *MaxSAT Evaluation* (2017-2021).

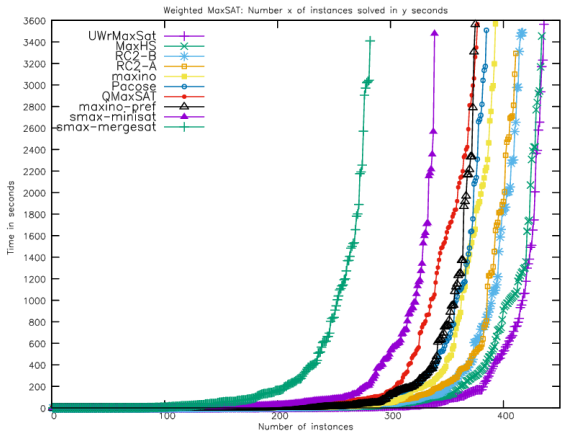

Figura 10: Resultats de la *MaxSAT Evaluation 2020* a la *Weighted Complete Track*. L'eix horitzontal indica el nombre d'instàncies resoltes en el temps màxim que indica l'eix vertical. Font: [28].

inicis de la dècada de 2010 [29] i se n'han fet diversos estudis des de llavors [10, 30]. Tanmateix, no s'havia aplicat mai per WCSP.

2.3 WCSP

2.3.1 Definició de CSP

Sigui $X = \{x_1, x_2, \ldots, x_n\}$ un conjunt de variables. Una variable x_i pren valors d'un domini discret d_i. Una *restricció* és una funció $c : Y \to \{cert, fals\}$ tal que $Y \subseteq X$. Una assignació t de valors dels dominis a variables de in Y satisfà la restricció ssi $c(t) = cert$.

Un *Constraint Satisfaction Problem* (CSP) és una tupla $P = (X, D, C)$ on $X = \{x_1, x_2, \ldots, x_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$ i $C = \{c_1(Y_1), \ldots, c_e(Y_e)\}$ són variables, dominis i restriccions, respectivament. Un CSP és satisfactible si hi ha una assignació de valors dels dominis a les variables que satisfà totes les restriccions. Un CSP solver és una funció $CSP(P)$ que indica si P és satisfactible o no.

Un *hard core* d'un CSP $P = (X, D, C)$ insatisfactible és un subconjunt $C' \subseteq C$ tal que el subproblema (X, D, C') és insatisfactible.

A les figures 11 i 12 il·lustrem dos exemples de CSP.

$P = (X, D, C)$

$X = \{x, y, z\}$
$D = \{d_x, d_y, d_z\}$
$C = \{c_1(x, y), c_2(y, z)\}$

$d_x = \{1,2,3\}$
$d_y = \{1,2\}$
$d_z = \{1,2\}$

$c_1(x,y)$

x	y	
1	1	*cert*
1	2	*fals*
2	1	*cert*
2	2	*cert*
3	1	*cert*
3	2	*fals*

$c_2(y,z)$

y	z	
1	1	*cert*
1	2	*fals*
2	1	*fals*
2	2	*cert*

Figura 11: Exemple 1. Aquest CSP és satisfactible ja que existeixen assignacions que compleixen totes les restriccions. Aquestes assignacions a (x, y, z) són $(1, 1, 1)$, $(2, 1, 1)$, $(2, 2, 2)$ i $(3, 1, 1)$.

$P = (X, D, C)$

$X = \{x, y, z\}$
$D = \{d_x, d_y, d_z\}$
$C = \{c_1(x, y), c_2(y, z), c_3(z)\}$

$d_x = \{1,2,3\}$
$d_y = \{1,2\}$
$d_z = \{1,2\}$

$c_1(x,y)$

x	y	
1	1	*cert*
1	2	*fals*
2	1	*cert*
2	2	*cert*
3	1	*cert*
3	2	*fals*

$c_2(y,z)$

y	z	
1	1	*cert*
1	2	*fals*
2	1	*fals*
2	2	*fals*

$c_3(z)$

z	
1	*fals*
2	*cert*

Figura 12: Exemple 2. Aquest CSP és insatisfactible ja que no hi ha cap assignació que satisfaci totes les restriccions. Un hard core de P és $C' = \{c_2(y, z), c_3(z)\}$ ja que $C' \subseteq C$ i (X, D, C') és insatisfactible.

2.3.2 Definició de WCSP

Weighted CSP (WCSP) és la versió d'optimització de CSP, en què les restriccions són substituïdes per funcions de cost.

Una funció de cost és una funció $f : Y \to \mathbb{N} \cup \{\infty\}$ tal que $Y \subseteq X$. Un WCSP és una tupla $P = (X, D, F)$ on $X = \{x_1, x_2, \ldots, x_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$ i $F = \{f_1(Y_1), \ldots, f_e(Y_e)\}$ són

variables, dominis i funcions de cost, respectivament. Sigui $F(X)$ la funció de cost global:

$$F(X) = \sum_{i=1}^{e} f_i(Y_i)$$

El problema WCSP és la minimització de $F(X)$,

$$WCSP(F) = \min_X F(X)$$ [4]

A la figura 13 il·lustrem un exemple de WCSP.

$$P = (X, D, F)$$

$$X = \{x, y, z\}$$
$$D = \{d_x, d_y, d_z\}$$
$$F = \{f_1(x, y), f_2(y, z)\}$$

$$d_x = \{1,2,3\}$$
$$d_y = \{1,2\}$$
$$d_z = \{1,2\}$$

$f_1(x,y)$

x	y	
1	1	0
1	2	6
2	1	30
2	2	∞
3	1	∞
3	2	44

$f_2(y,z)$

y	z	
1	1	0
1	2	10
2	1	∞
2	2	0

Figura 13: Exemple. La solució a aquest WCSP és l'assignació $(x, y, z) = (1, 1, 1)$, que és l'assignació que minimitza la funció de cost global $F(x, y, z) = f_1(x, y) + f_2(y, z)$. El cost de l'assignació és $F(1, 1, 1) = f_1(1, 1) + f_2(1, 1) = 0 + 0 = 0$.

2.3.3 Exemple pràctic

A continuació veurem dos problemes que es poden expressar com a CSP i WCSP.

El problema de *coloració d'un graf* consisteix a assignar un color a cada vèrtex d'un graf de manera que cada parell de vèrtexs adjacents tenen colors diferents. Formalment, la coloració d'un graf $G = (V, E)$ compleix $\forall\{u, v\} \in E : color(u) \neq color(v)$. La figura 14 mostra un graf colorat amb aquestes restriccions.

Quan representem aquest problema en forma de CSP hi ha una variable c_v per cada vèrtex $v \in V$ que representa el color del vèrtex. El domini de les variables és el conjunt de colors que podem assignar. El conjunt de restriccions és $\{c_u \neq c_v \mid \{u, v\} \in E\}$.

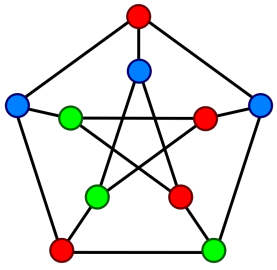

Figura 14: Graf colorat. Font: [31].

Una versió d'optimització del problema és el que anomenem *coloració a grisos d'un graf.* En aquesta versió, el conjunt de colors és una escala de grisos i cada gris té associat un número natural (com més fosc, més gran). L'objectiu és que cada parell de vèrtexs adjacents tinguin colors com més diferents, millor. Formalment a la coloració a grisos d'un graf $G = (V, E)$ volem maximitzar $\sum_{\{u,v\}\in E} |color(u) - color(v)|$, és a dir, volem minimitzar $\sum_{\{u,v\}\in E} -|color(u) - color(v)|$. La figura 15 mostra un graf colorat a grisos amb aquesta restricció.

[4] $WCSP(F)$ pot referir al cost mínim ($\min_X F(X)$) o a l'assignació de cost mínim ($\text{argmin}_X F(X)$).

Quan representem aquest problema en forma de WCSP hi ha una variable c_v per cada vèrtex $v \in V$ que representa el color del vèrtex. El domini de les variables és el conjunt d'enters associats als grisos que podem assignar. El conjunt de funcions de cost és $\{m - |c_u - c_v| \mid \{u, v\} \in E\}$ on m una constant equivalent al valor màxim del domini. m evita que les funcions tinguin outputs negatius, la qual cosa no està permesa a WCSP. La funció de cost global és $\sum_{\{u,v\} \in E} m - |color(u) - color(v)|$.

EXEMPLE: La coloració a grisos del graf de la figura 15 es podria representar amb el WCSP $P = (X, D, F)$ amb variables $X = \{c_1, c_2, c_3, c_4\}$, dominis $D = \{\{0, 50, 100\} \mid x \in X\}$ (0 és blanc, 50 és gris 50%, 100 és negre) i funcions $F = \{100 - |c_1 - c_2|, 100 - |c_2 - c_3|, 100 - |c_3 - c_4|, 100 - |c_4 - c_2|\}$. L'assignació que mostra la figura ($c_1 = 0, c_2 = 100, c_3 = 0, c_4 = 50$) té cost $(100 - |0 - 100|) + (100 - |100 - 0|) + (100 - |0 - 50|) + (100 - |50 - 100|) = 0 + 0 + 50 + 50 = 100$, que és el mínim possible.

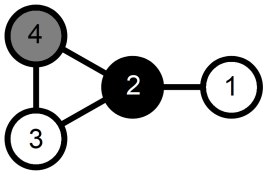

Figura 15: Graf colorat a grisos. El graf és $G = (V, E)$ amb $V = \{1, 2, 3, 4\}$ i $E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 2\}\}$.

2.3.4 Estat de l'art

D'entre els quatre problemes mencionats (SAT, MaxSAT, CSP i WCSP), probablement WCSP és el menys estudiat. Bona part de les publicacions que el mencionen ho fan amb el nom de *Cost Function Networks* (CFN).

El Centre de Recerca en Informàtica de Lens, de la Universitat d'Artois, ha organitzat quatre competicions internacionals de solvers CSP [32 - 34]. La tercera edició, de 2008, va ser l'única en incloure WCSP. També inclou els problemes CSP i MaxCSP. Els defineix així:

- CSP. Problema de decisió. Troba una instanciació completa de variables que satisfaci totes les restriccions.
- MaxCSP. Problema d'optimització. Troba una instanciació completa de variables que maximitzi el nombre de restriccions satisfetes.
- WCSP. Problema d'optimització. Troba una instanciació completa de variables amb cost mínim.

A la categoria CSP es van presentar catorze solvers. A MaxCSP se'n van presentar quatre. A WCSP només se'n va presentar un, el *toulbar2*. Pel que fa MaxCSP, *toulbar2* va guanyar a les dues modalitats on es va presentar (2-ARY-EXT i N-ARY-EXT) i *Sugar* va guanyar a les altres tres modalitats (2-ARY-INT, N-ARY-INT i GLOBAL).

Aquesta competició (*Third International CSP Solver Competition*, CPAI08) estava afiliada a la CP 2008: *14th International Conference on Principles and Practice of Constraint Programming.*

Més enllà de les competicions, el 2016 la revista *Constraints* va publicar l'article [2], en què avaluava cinc solvers d'optimització NP-hard: *toulbar2*, *clpex*, *daoopt*, *maxhs* i *gecode*. Van experimentar amb un gran nombre de benchmarks de diversos problemes, incloent WCSP. L'article conclou que els cinc solvers són eficaços i que cap és globalment millor que els altres, sinó que el millor varia segons el tipus de problema. "Per a cada font de benchmarks, el millor solver sol ser un solver que es dedica a aquest tipus de problemes (és a dir, *toulbar2* per a WCSP, *maxhs* per a MaxSAT, *gecode* per a CP). Tanmateix, alguns solvers, com ara *maxhs*, *cplex* i *toulbar2*, van funcionar bé en diversos problemes", afirmen. Així, destaquen "l'eficàcia de codificar un problema en un llenguatge relacionat i explotar tecnologies de resolució complementàries".

La figura 16 mostra els resultats empírics de dos de les sis col·leccions de benchmarks que analitza l'article: CNF (àlies WCSP)[5] i MaxCSP. Observem que a les dues col·leccions *toulbar2* és el solver que més instàncies ha resolt menys d'una hora. L'article apunta que *toulbar2* "domina clarament" en les instàncies de disseny de proteïnes, part de la col·lecció CFN. Més endavant veurem altres estudis que demostren l'eficiència de *toulbar2* per al disseny computacional de proteïnes.

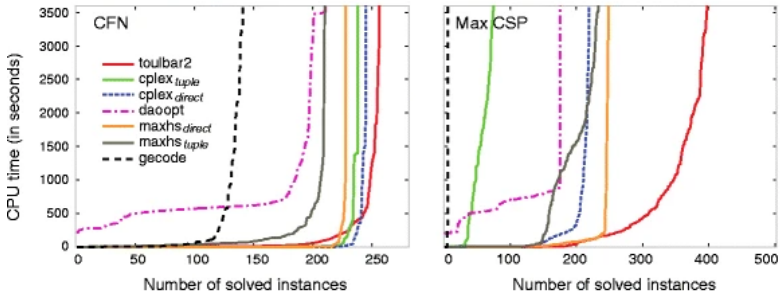

Figura 16: Resultats de [2] corresponents a CNF i MaxCSP, sobre l'eficiència de diferents solvers d'optimització. L'eix horitzontal indica el nombre d'instàncies resoltes en el temps màxim que indica l'eix vertical.

2.3.5 El solver *toulbar2*

toulbar2 [17, 18] és un solver de codi obert per a WCSP implementat en C++.

Fou desenvolupat per equips de Touluse (INRAE MIAT) i Barcelona (UPC, IIIA-CSIC), ciutats que li donen nom. Per part de la UPC, hi van col·laborar els dos professors que tutelen aquest TFG. Actualment és mantingut per Simon de Givry, de Touluse.

Ha guanyat diverses competicions sobre models gràfics probabilístics i deterministes. És el cas d'algunes categories de UAI 2008, UAI 2010, UAI 2014 i la ja mencionada CPAI08.

toulbar2 utilitza algoritmes de *best-first* i *branch-and-bound* aprofitant consistències d'arc *soft*. També incorpora *parallel variable neighborhood search*. Els algoritmes de *toulbar2* es basen en una vintena d'articles científics de diversos autors publicats entre els anys 2000 i 2020.

Si el nostre algoritme de WCSP és prou eficient, l'incorporarem a *toulbar2*.

2.3.6 Aplicacions de WCSP

A part del disseny computacional de proteïnes, WCSP també té aquestes aplicacions:

- **Gestió de satèl·lits.** Decidir quines observacions a la Terra fa un satèl·lit és un problema d'optimització modelable a WCSP. [4] el defineix així: "donats (1) un conjunt d'imatges candidates per al dia següent, cadascuna associada amb un pes que reflecteixi la seva importància, (2) un conjunt de restriccions imperatives que expressen limitacions físiques [...], seleccionar un subconjunt de candidats que compleixi totes les restriccions i maximitzi la suma dels pesos dels candidats seleccionats".
- **Genètica.** WCSP també pot servir per modelar problemes relacionats amb la genètica, des de detectar errors a les dades [3] fins a estudiar el lligament genètic [12]. Estudiar el lligament genètic pot servir, entre altres, per detectar predisposicions per tenir certes malalties [35].
- **Assignació de radiofreqüències**. Tal com indica [5], aquest problema consisteix en "proporcionar canals de comunicació a partir de recursos espectrals limitats mantenint al mínim les interferències que pateixen aquells que volen comunicar-se en una xarxa de comunicacions de ràdio determinada".

[5] La definició de CNF de [2] és equivalent a la nostra definició de WCSP

2.4 Disseny computacional de proteïnes (CPD)

Les proteïnes són grans mol·lècules presents a la gran majoria de funcions biològiques. Al llarg de milions d'anys la selecció natural s'ha encarregat d'optimitzar-ne la configuració per tal d'adequar-les a les necessitats de l'organisme. Si volem adequar-les a certes necessitats biomèdiques o industrials podem dissenyar-les computacionalment. [1]

Les proteïnes són cadenes d'aminoàcids. L'estructura química d'un aminoàcid és la que mostra la figura 17. La cadena lateral (R) és l'únic component variable de l'aminoàcid. En general, hi ha 20 tipus de cadenes laterals, per tant, hi ha 20 tipus d'aminoàcids. Els aminoàcids s'uneixen per enllaços peptídics. La successió d'aminoàcids units per enllaços peptídics forma una llarga cadena que conforma la proteïna. [36] Tal com mostra la figura 18, la cadena d'aminoàcids (estructura primària) pot prendre diverses formes per formar l'estructura secundària. El plegament de l'estructura secundària dona lloc a la terciària i la interacció entre subunitats, la quaternària. [11]

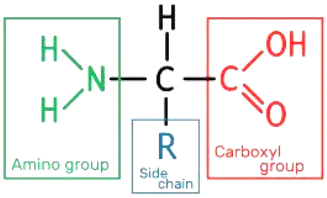

Figura 17: Estructura química d'un aminoàcid. La lletra R representa una cadena d'àtoms (la cadena lateral); la resta de lletres representen un àtom cada una (carboni, nitrogen, hidrogen, oxigen). Font: [36].

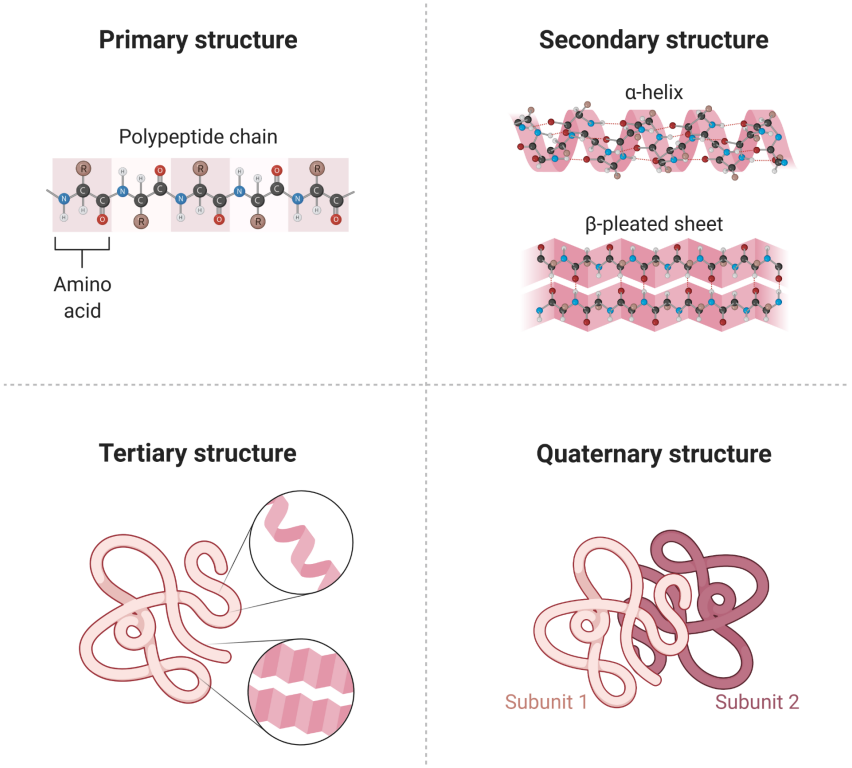

Figura 18: Els quatre nivells de l'estructura d'una proteïna. Font: [11].

Cadascuna de les cadenes laterals dels aminoàcids pot prendre posicions diferents, conegudes com a rotàmers. La figura 19 en mostra un exemple. La posició de les cadenes laterals té efecte sobre l'energia de la proteïna, com veiem a l'exemple de la figura 20. [37, 38]

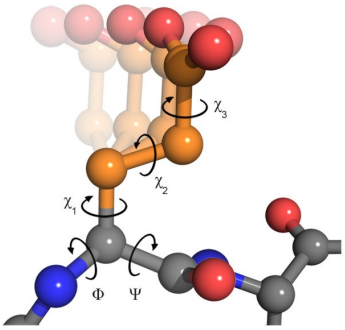

Figura 19: Graus de llibertat d'una cadena lateral amb els quals pot conformar diferents rotàmers. Font: [37].

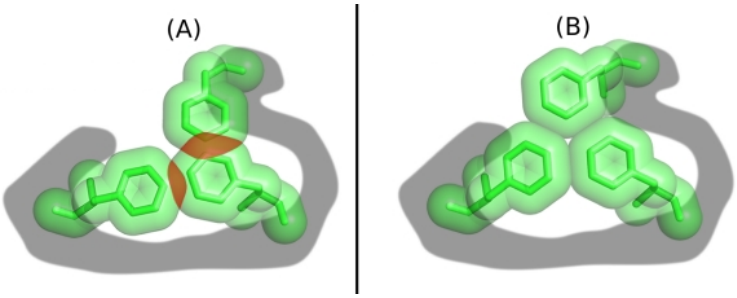

Figura 20: Exemple de l'efecte dels rotàmetres sobre l'energia. En el cas (A) els radis electrònics dels àtoms de les cadenes laterals col·lisionen, cosa que és energèticament desfavorable. En el cas (B) no hi ha col·lisions de manera que el sistema està en un estat més estable, és a dir, amb menor energia. Font: [38].

Volem la proteïna en el seu estat més estable, en altres paraules, volem minimitzar la seva energia. La funció (simplificada) d'energia segons la llista de rotàmers r, és:

$$E(r) = \sum_i E(r_i) + \sum_{j<i} E(r_i, r_j)$$

[1]

El disseny computacional de proteïnes (CPD) determina la conformació d'un subconjunt de rotàmers (és a dir, un subconjunt d'aminoàcids) de la proteïna. L'entrada de l'algoritme són les funcions d'energia de casa rotàmer i cada parell de rotàmers. La sortida és la seqüència de rotàmers que minimitza l'energia.

A part de la conformació òptima (la que minimitza l'energia) també són interessants conformacions pròximes a l'òptima. [39]

La modelació de CPD a WCSP és bastant senzilla. Hi ha una variable per cada aminoàcid a determinar. El domini de les variables és la combinació de tipus d'aminoàcid i conformació del rotàmer. La funció de cost és la funció d'energia. Les funcions locals són funcions unàries i binàries que expressen l'energia de rotàmers i de la interacció entre parells de rotàmers. A WCSP els costos són positius però a CPD l'energia és negativa. Això es pot resoldre fàcilment sumant una constant a cada funció d'energia. [13, 39]

2.4.1 Estat de l'art

Segons [39] el disseny computacional de proteïnes té un "potencial excepcional". "S'ha aplicat amb èxit per augmentar la termoestabilitat i solubilitat de les proteïnes; alterar l'especificitat cap a altres molècules, i dissenyar centres actius per construir enzims *de novo*". Tanmateix, afirma que els mètodes de CPD "encara han de madurar". "En particular, es necessiten tècniques d'optimització computacional més eficients per explorar el gran espai combinatori i facilitar la incorporació de models de proteïnes més realistes i flexibles".

Alguns problemes abstractes amb què es pot modelar CPD són WCSP, MaxSAT, DEE/A*, programació lineal entera (ILP), programació lineal 0/1, programació quadràtica 0/1, optimització quadràtica 0/1 i models gràfics. [13, 14].

Tal com diu [1] (publicat el 2019), el solver *toulbar2* "ha millorat significativament l'estat de l'art de l'eficiència de CPD en el model discret per parelles" i refereix que [14] ho va demostrar "amb un gran nombre de benchmarks empírics". Aquest article, de 2013, havia conclòs que l'enfocament basat en WCSP per a CPD "proporciona speedups notables, que ens permeten explorar grans espais de conformació de seqüències de manera molt més eficient que l'algoritme DEE/A* o els algoritmes ILP d'última generació". Les figures 21 i 22 mostren el resultat d'altres experiments que van demostrar l'eficiència de *toulbar2*, publicats el 2014 i el 2019 per investigadors que van desenvolupar aquest solver.

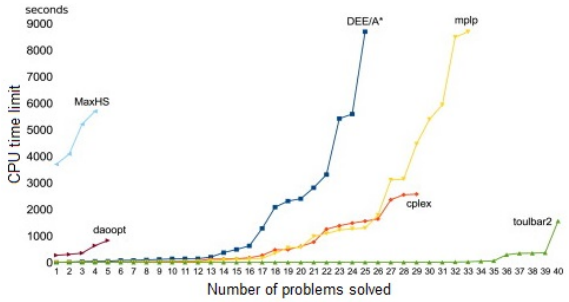

Figura 21: Resultats de [13], sobre l'eficiència de diferents solvers en CPD.

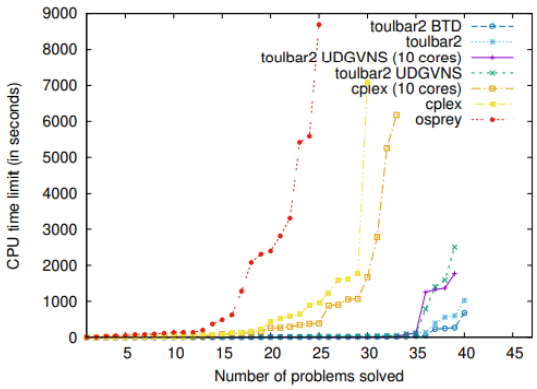

Figura 22: Resultats de [39], sobre l'eficiència de diferents solvers en CPD.

2.4.2 Aplicacions de CPD

Aquestes són algunes de les aplicacions del disseny de proteïnes per la medicina i la indústria:

- **Anticossos.** Els anticossos són proteïnes molt útils per la medicina. Gràcies a la seva gran versatilitat actualment són la classe més gran de bioterapèutics. Computacionalment podem dissenyar anticossos a mida. [40, 41]
- **Detergents.** Els enzims són proteïnes que actuen de catalitzador, és a dir, augmenten la velocitat d'una reacció química. En els detergents s'utilitzen enzims. Interessa que tinguin termoestabilitat, sigui a temperatures baixes o altes. Els podem dissenyar computacionalment. [42, 43]
- **Degradació de plàstics.** Els enzims també poden servir per catalitzar la biodegradació de plàstics. Per exemple, s'han fet diversos estudis sobre la degradació de tereftalat de polietilè (PET), un dels plàstics més utilitzats i que ha generat greus problemes mediambientals. Podem dissenyar aquests enzims computacionalment. [44]

2.5 Complexitat

CSP i SAT són problemes *NP-complets*. Les seves versions d'optimització –WCSP i MaxSAT– són problemes *NP-hard*. [45] A més, WCSP i MaxSAT són *strongly NP-hard* i també *NP-optimització* tal com [46] descriu aquests conceptes.

Tal com indica [47], el problema del disseny de proteïnes també és NP-hard.

Els problemes esmentats són *decidibles*, és a dir, teòricament qualsevol solver pot resoldre qualsevol instància en temps finit. Ara bé, finit no vol dir factible. A la pràctica el temps de resolució de certes instàncies en certs solvers és inabastable. L'objectiu dels solvers és trobar en un temps raonable solucions a instàncies útils, aplicables a la vida real.

3 Implicit Hitting Sets & Vectors

3.1 Hitting Sets

3.1.1 Minimum Cost Hitting Set Problem (MHS)

Considerem un conjunt F tal que cada $u \in F$ té un cost associat $cost(u)$.

Siguin $h, k \subseteq F$: quan h i k no són disjunts (i.e. $k \cap h \neq \emptyset$), diem que h *hiteja* k.

Sigui $\mathcal{C}$ un conjunt de subconjunts de F (i.e. $\mathcal{C} \subseteq \{k \subseteq F\}$), un *hitting set* (HS) de $\mathcal{C}$ és un conjunt $h \subseteq F$ tal que $\forall k \in \mathcal{C} : h$ hiteja k.

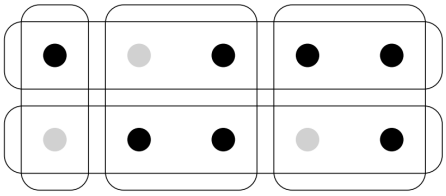

Figura 23: Exemple. Els conjunt de punts representa F. Els conjunt de requadres representa $\mathcal{C}$. El conjunt de punts grisos representen un hitting set de $\mathcal{C}$. Figura obtinguda de [10].

Sigui $\mathcal{HS}(\mathcal{C})$ el conjunt de hitting sets de $\mathcal{C}$, el *minimum cost hitting set* (MHS) de $\mathcal{C}$ és

$$MHS(\mathcal{C}) = \min_{h \in \mathcal{HS}(\mathcal{C})} cost(h) \ [6]$$

on $cost(h) = \sum_{u \in h} cost(u)$.

Tingueu en compte que hi podria haver més d'un MHS.

EXEMPLE: Suposem que F és un conjunt de persones d'entre les quals volem seleccionar un equip per complir una missió. Necessitem certes habilitats per la missió. Per cada habilitat hi ha un conjunt $k \subseteq F$ de persones que la tenen. El cost de contractar una persona $u \in F$ és $cost(u)$. Volem trobar l'equip més barat de manera que hi hagi almenys una persona per cada habilitat.

Noteu que trobar un $h \in \mathcal{HS}(\mathcal{C})$ és fàcil, però trobar un MHS és NP-hard. [48]

3.1.2 Implicit Minimum Cost Hitting Set Problem (IMHS)

Un *Implicit Minimum Cost Hitting Set Problem* (IMHS) és un MHS en què $\mathcal{C}$ no està disponible explícitament, sinó que és implícit per la semàntica de F. L'anomenarem $\mathcal{C}_F$.

EXEMPLE: Sigui G un graf cíclic, volem eliminar el conjunt més petit d'arestes de manera que el graf esdevingui acíclic. Aquí F és el conjunt d'arestes a G, i $\mathcal{C}_F$ és el conjunt de cicles a G. El conjunt de cicles és implícit per les arestes del graf. El conjunt de cicles d'un graf pot ser gran i difícil de calcular, així que aquest és un bon exemple d'IMHS.

A continuació presentem l'algoritme 1, per a IMHS. L'algoritme assumeix que tenim una funció MHS($\mathcal{C}$) que calcula el minimum cost hitting set. També assumeix que tenim una funció IHS(h, F, k) que ens indica si h és un hitting set de $\mathcal{C}_F$. Si no és el cas, la funció retorna un $k \in \mathcal{C}_F$ no hitejat per h, que certifica la resposta negativa.

EXEMPLE: En el problema de l'eliminació de cicles, h és un subconjunt d'arestes i IHS(h, F, k) és un algoritme que indica si eliminar h fa que el graf sigui acíclic. Si el graf es manté cíclic, la funció

[6] $MHS(\mathcal{C})$ pot referir al cost mínim ($\min_{h \in \mathcal{HS}(\mathcal{C})} cost(h)$) o al conjunt de cost mínim ($\text{argmin}_{h \in \mathcal{HS}(\mathcal{C})} cost(h)$).

retorna un cicle k que és manté a $F - h$. És a dir, $k \subseteq F - h$, ja que k i h són disjunts (i.e. h no hiteja k).

La correctesa de l'algoritme ve de la següent propietat:

Propietat 1. Sigui $\mathcal{K} \subset \mathcal{C}$. Si h és MHS de $\mathcal{K}$ i h és HS de $\mathcal{C}$, llavors h és MHS de $\mathcal{C}$.

Demostració. Afegint conjunts per hitejar, el cost del MHS no pot disminuir. □

Per tant, podem resoldre el problema IMHS fent créixer $\mathcal{K}$ tal que $\mathcal{K} \subseteq \mathcal{C}_F$ fins que el MHS de $\mathcal{K}$ sigui un HS de $\mathcal{C}_F$. Aquesta és la idea de l'algoritme 1. Aquest algoritme fou proposat per primera vegada a [49].

Function `IMHS(`F`)`
begin
  $\mathcal{K} = \emptyset; h = \emptyset; lb = 0;$
  while *not IHS*(h, F, k) **do**
    /* $\mathcal{K} \subseteq \mathcal{C}_F$, $h \in MHS(\mathcal{K})$ */ ;
    $\mathcal{K} := \mathcal{K} \cup \{k\}$;
    $h :=$MHS$(\mathcal{K})$;
    $lb := cost(h)$;
  end
  return h;
end

Algorithm 1: Algoritme que resol l'implicit minimum cost hitting set problem (IMHS) mitjançant lower bounds (lb).

Tingueu en compte que la implementació de $IHS()$ i $MHS()$ afecta al nombre d'iteracions del bucle.

A continuació mostrem l'algoritme 2, una variació de l'algoritme 1 que enlloc de resoldre el problema calculant una seqüència de lower bounds, combina lower bounds i upper bounds. La idea és alternar el càlcul del hitting sets (no necessàriament de cost mínim) i minimum cost hitting sets. Tingueu en compte que calcular hitting sets no mínims és computacionalment fàcil.

L'algoritme va fent créixer $\mathcal{K} \subseteq \mathcal{C}_F$ i calcula HS de $\mathcal{K}$ fins que se'n troba un que també és HS de $\mathcal{C}_F$. Això ens proporciona un upper bound de l'òptim. Llavors calcula un MHS de $\mathcal{K}$, que ens proporciona un lower bound de l'òptim. Continua el procés fins que lower bound i upper bound es troben. Aquest algoritme fou proposat per primera vegada a [50].

Function `IMHS(`F`)`
begin
  $\mathcal{K} = \emptyset; h = \emptyset; lb = 0; ub = \infty;$
  while $lb < ub$ **do**
    /* $\mathcal{K} \subseteq \mathcal{C}_F$, $h \in \mathcal{HS}(\mathcal{K})$, $lb \leq cost(MHS(\mathcal{C}_F)) \leq ub$*/ ;
    if *IHS*(h, F, k) **then**
      $ub := \min\{ub, cost(h)\}$;
      $h :=$MHS$(\mathcal{K})$;
      $lb := \max\{lb, cost(h)\}$;
    end
    else
      $\mathcal{K} := \mathcal{K} \cup \{k\}$;
      $h :=$HS$(\mathcal{K})$;
    end
  end
  return h;
end

Algorithm 2: Algoritme que resol l'implicit minimum cost hitting set problem (IMHS) mitjançant lower i upper bounds (lub).

3.1.3 MaxSAT com a IMHS

Siguin $\mathcal{H}$ els conjunts de clàusules tals que, si les eliminem, F esdevé satisfactible,

$$\mathcal{H} = \{h \subseteq F \mid \ SAT(F - h)\}$$

llavors, una altra manera de definir MaxSAT és

$$MaxSAT(F) = \min_{h \in \mathcal{H}} cost(h)$$

on $cost(h) = \sum_{c \in h} cost(c)$.

Sigui $\mathcal{C}$ el conjunt de cores de F (i.e, $\mathcal{C} = \{k \subseteq F \mid UNSAT(k)\}$) i $\mathcal{HS}(\mathcal{C})$ el conjunt de hitting sets de $\mathcal{C}$.

Propietat 2. $\mathcal{H}$ és el conjunt de hitting sets de $\mathcal{C}$ (i.e, $\mathcal{H} = \mathcal{HS}(\mathcal{C})$)

Demostració. Suposem que $h \in \mathcal{H}$ no és un hitting set de $\mathcal{C}$. Per definició de $\mathcal{H}$, $SAT(F-h)$. Com que h no és un hitting set de $\mathcal{C}$, existeix $k \in \mathcal{C}$ no hitejat per h, és a dir, $k \cap h = \emptyset$. Per definició de $\mathcal{C}$, $UNSAT(k)$. Per tant, k és unsat, $F - h$ és sat i $k \subseteq F - h$, que és impossible.

Suposem que existeix un h hitting set de $\mathcal{C}$ i $h \notin \mathcal{H}$. Per definició de $\mathcal{H}$, $UNSAT(F - h)$, és a dir que $F - h$ és un core. Com que h hiteja tots els cores, h hiteja $F - h$, és a dir, $h \cap (F - h) \neq \emptyset$, que és impossible.

□

Per tant, podem resoldre MaxSAT trobant $h \subset F$ de mínim cost que hiteji tots els cores de F, és a dir, trobant un minimum cost hitting set de $\mathcal{C}$, que és el conjunt de cores de F:

$$MaxSAT(F) = \min_{h \in \mathcal{HS}(\mathcal{C})} cost(h) = MHS(\mathcal{C})$$

$\mathcal{C}$ no està disponible explícitament, sinó que és implícit en F (concretament, $\mathcal{C} = \{k \subseteq F \mid UNSAT(k)\}$), per tant, aquest és un dels casos en què convé aplicar IMHS.

Per tal d'aplicar els algoritmes de IMHS vistos anteriorment necessitem una funció que, donat h (MHS de $\mathcal{K} \subseteq \mathcal{C}$) indiqui si h és un hitting set de $\mathcal{C}$ i, si no, retorni un $k \in \mathcal{C}$ no hitejat per h. La propietat següent mostra que un SAT solver amb la capacitat de retornar un core de fórmules insatisfactibles compleix els requisits.

Propietat 3. Sigui $\mathcal{K} \subseteq \mathcal{C}$ un subconjunt de cores i h un MHS de $\mathcal{K}$:

- si $(F - h)$ és satisfactible, llavors h hiteja tots els cores i, per la propietat 1, h és un MHS de $\mathcal{C}$.
- si $(F - h)$ és insatisfactible, llavors hi ha un core k de $(F - h)$ que pertany a $\mathcal{C}$ i h no hiteja k.

Llavors assumim l'existència d'una funció CoreSAT$(F - h, k)$ que retorni si $(F - h)$ és satisfactible o no, i en cas que no ho sigui assigni a k un core de $(F - h)$.

També assumim l'existència d'una funció MHS$(\mathcal{K})$ que retorna un minimum cost hitting set de $\mathcal{K}$ i una funció HS$(\mathcal{K})$ que retorna un hitting set de $\mathcal{K}$.

A continuació veiem dos algoritmes per resoldre MaxSAT que són l'adaptació a MaxSAT dels algoritmes de IMHS que hem vist anteriorment. L'algoritme 3 és l'adaptació de l'algoritme 1 i l'algoritme 4 és l'adaptació de l'algoritme 2.

La complexitat temporal dels algoritmes és $O(|\mathcal{C}| \times m) = O(2^{|F|} \times m)$ on m és la complexitat temporal d'una crida a MHS.

```
Function MaxSAT(F)
begin
    K = ∅; h = ∅; lb = 0;
    while not CoreSAT(F − h, k) do
        /* K ⊆ C, h ∈ MHS(K) */ ;
        K := K ∪ {k};
        h :=MHS(K);
        lb := cost(h);
    end
    return lb;
end
```

Algorithm 3: Algoritme que resol MaxSAT mitjançant lower bounds (lb).

```
Function MaxSAT(F)
begin
    K = ∅; h = ∅; lb = 0; ub = ∞;
    while lb < ub do
        /* K ⊆ C_F, h ∈ HS(K), lb ≤ cost(MHS(C_F)) ≤ ub*/ ;
        if CoreSAT(F − h, k) then
            ub := min{ub, cost(h)};
            h :=MHS(K);
            lb := max{lb, cost(h)};
        end
        else
            K := K ∪ {k};
            h :=HS(K);
        end
    end
    return lb;
end
```

Algorithm 4: Algoritme que resol MaxSAT mitjançant lower i upper bounds (lub).

3.2 Hitting Vectors

3.2.1 Minimum Cost Hitting Vector Problem (MHV)

Siguin $V_1, V_2, \ldots, V_e$ subconjunts finits de $\mathbb{N}$. Siguin $\vec{h} = (h_1, h_2, \ldots, h_e)$ i $\vec{k} = (k_1, k_2, \ldots, k_e)$ vectors e-dimensionals amb $h_i, k_i \in V_i$. Diem que $\vec{h}$ *és dominat per* $\vec{k}$, escrit $\vec{h} \leq \vec{k}$, quan $\forall i : h_i \leq k_i$. Quan $\vec{h} \not\leq \vec{k}$, diem que $\vec{h}$ *hiteja* $\vec{k}$ (i.e. $\exists i : h_i > k_i$).

Diem que $\vec{h}$ és un *hitting vector* (HV) d'un conjunt de vectors $\mathcal{C}$ quan $\vec{h}$ hiteja tots els vectors de $\mathcal{C}$ (i.e. $\forall \vec{k} \in \mathcal{C} : \vec{h} \not\leq \vec{k}$). Sigui $\mathcal{HV}(\mathcal{C})$ el conjunt de hitting vectors de $\mathcal{C}$ (i.e. $\mathcal{HV}(\mathcal{C}) = \{\vec{h} \mid \forall \vec{k} \in \mathcal{C} : \vec{h} \not\leq \vec{k}\}$).

Diem que el cost d'un vector és la suma dels seus components: $cost(\vec{h}) = \sum_{i=1}^{e} h_i$.

El *minimum cost hitting vector* (MHV) de $\mathcal{C}$ és

$$MHV(\mathcal{C}) = \min_{\vec{h} \in \mathcal{HV}(\mathcal{C})} cost(\vec{h}) = \min_{\vec{h} \in \mathcal{HV}(\mathcal{C})} \sum_{i=1}^{e} h_i \ ^{7}$$

Tingueu en compte que hi podria haver més d'un MHV.

EXEMPLE: Siguin $V_1 = V_2 = V_3 = \{0, 1, 2, \ldots, 30\}$ i $\mathcal{C} = \{\vec{k_1}, \vec{k_2}, \vec{k_3}\} = \{(1, 2, 30), (2, 1, 10), (30, 20, 5)\}$. Un hitting vector de $\mathcal{C}$ és $\vec{h} = (0, 21, 0)$. El seu cost és $0 + 21 + 0 = 21$. Un minimum cost hitting vector de $\mathcal{C}$ és $\vec{h} = (3, 0, 6)$. El seu cost és $3 + 0 + 6 = 9$.

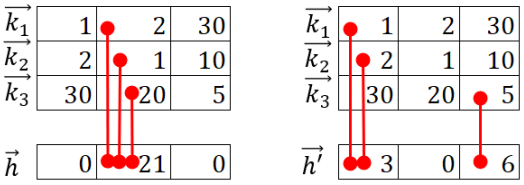

Figura 24: Representació gràfica de l'exemple. Cada línia vermella uneix un h_i amb un k_{ji} tals que $h_i > k_{ji}$. Per definició, $\vec{h}$ hiteja $\vec{k_j}$ si $\exists i : h_i > k_{ji}$

3.2.2 Implicit Minimum Cost Hitting Vector Problem (IMHV)

Igual que en el cas de Hitting Sets, hi ha situacions en què $\mathcal{C}$ no està disponible explícitament. En aquests casos parlem d'*Implicit Minimum Cost Hitting Vector Problem* (IMHV), que és molt similar a IMHS però amb vectors. Així que adaptem els algoritmes d'IMHS (1 i 2) a IMHV, que introduïm als algoritmes 5 i 6. La funció IHV$(\vec{h}, F, \vec{k})$ indica si $\vec{h}$ és un hitting set de $\mathcal{C}_F$, i si ho ho és, la funció proporciona un $\vec{k} \in \mathcal{C}_F$ no hitejat per $\vec{h}$, que certifica la resposta negativa. També suposem que tenim una funció que computa un MHV i, en el segon algoritme, una funció que retorna un HV.

[7] $MHV(\mathcal{C})$ pot referir al cost mínim ($\min_{\vec{h} \in \mathcal{HV}(\mathcal{C})} cost(\vec{h})$) o al vector de cost mínim ($\text{argmin}_{\vec{h} \in \mathcal{HV}(\mathcal{C})} cost(\vec{h})$).

Function `IMHV(`F`)`
begin
  $\mathcal{K} = \emptyset; \vec{h} = \vec{0}; lb = 0;$
  while *not* $IHV(\vec{h}, F, \vec{k})$ **do**
    /* $\mathcal{K} \subseteq \mathcal{C}_F$, $\vec{h} \in MHV(\mathcal{K})$ */ ;
    $\mathcal{K} := \mathcal{K} \cup \{\vec{k}\}$;
    $\vec{h} :=$ MHV($\mathcal{K}$);
    $lb := cost(\vec{h})$;
  end
  return $\vec{h}$;
end

Algorithm 5: Algoritme que resol l'implicit minimum cost hitting vector problem (IMHV) mitjançant lower bounds (lb).

Function `IMHV(`F`)`
begin
  $\mathcal{K} = \emptyset; \vec{h} = \vec{0}; lb = 0; ub = \infty;$
  while $lb < ub$ **do**
    /* $\mathcal{K} \subseteq \mathcal{C}_F$, $\vec{h} \in \mathcal{HV}(\mathcal{K})$, $lb \leq cost(MHV(\mathcal{C}_F)) \leq ub$*/ ;
    if $IHV(\vec{h}, F, k)$ **then**
      $ub := \min\{ub, cost(\vec{h})\}$;
      $\vec{h} :=$ MHV($\mathcal{K}$);
      $lb := \max\{lb, cost(\vec{h})\}$;
    end
    else
      $\mathcal{K} := \mathcal{K} \cup \{k\}$;
      $\vec{h} :=$ HV($\mathcal{K}$);
    end
  end
  return $\vec{h}$;
end

Algorithm 6: Algoritme que resol l'implicit minimum cost hitting vector problem (IMHV) mitjançant lower i upper bounds (lub).

3.2.3 WCSP com a IMHV

Considerem una funció de cost $f_i(Y_i)$. Sigui V_i el conjunt de costos finits presents a f_i (és a dir, $V_i = \{f_i(t) \mid t \text{ assignació de } Y_i\} - \{\infty\}$.

Podem assumir, sense perdre generalitat, que l'element més petit de V_i és 0.

Donat $h_i \in V_i$ definim l'enduriment de f_i amb h_i com a

$$f_i^{h_i}(t) = \begin{cases} cert & \text{si } f_i(t) \leq h_i \\ fals & \text{si } f_i(t) > h_i \end{cases}$$

Donat un vector $\vec{h} = (h_1, h_2, \ldots, h_e)$ tal que $h_i \in V_i$ definim l'enduriment de $F = \{f_1(Y_1), \ldots, f_e(Y_e)\}$ amb $\vec{h}$ com a

$$F_{\vec{h}} = \{f_1^{h_1}(Y_1), \ldots, f_e^{h_e}(Y_e)\}$$

Tingueu en compte que $f_i^{h_i}$ és una restricció, no una funció de cost. Per tant, $F_{\vec{h}}$ és un CSP, no un WCSP. Tingueu en compte que si $\vec{h} \leq \vec{h}'$ llavors $F_{\vec{h}'}$ és una relaxació de $F_{\vec{h}}$. L'enduriment de F més restringit és $F_{\vec{0}}$ i el més relaxat és el CSP que només prohibeix les tuples de WCSP que tinguin cost infinit.
Propietat 4. Si $\vec{h} \leq \vec{h}'$, llavors $F_{\vec{h}}$ `satisfactible` $\Rightarrow F_{\vec{h}'}$ `satisfactible` i, per contrarecíproc, $F_{\vec{h}'}$ `insatisfactible` $\Rightarrow F_{\vec{h}}$ `insatisfactible`.

Sigui $\mathcal{H}$ el conjunt de vectors $\vec{h}$ que fan $F_{\vec{h}}$ satisfactible,

$$\mathcal{H} = \{\vec{h} \mid F_{\vec{h}} \texttt{ satisfactible}\}$$

Aleshores, una altra manera de definir WCSP és

$$WCSP(F) = \min_{\vec{h} \in \mathcal{H}} \sum_i h_i$$

Un *soft core* de F és un vector $\vec{k}$ tal que $F_{\vec{k}}$ és insatisfactible. Sigui $\mathcal{C}$ el conjunt de cores de F (i.e. $\mathcal{C} = \{\vec{k} \mid F_{\vec{k}} \texttt{ insatisfactible}\}$).

Igual que en el cas de MaxSAT tenim:
Propietat 5. $\mathcal{H}$ és el conjunt de hitting vectors de $\mathcal{C}$ (i.e. $\mathcal{H} = \mathcal{HV}(\mathcal{C})$)

Demostració. Demostrem els dos sentits de la implicació per contradicció:

- ($\Rightarrow$) Suposem que existeix $F_{\vec{h}}$ satisfactible tal que existeix un $F_{\vec{k}}$ insatisfactible tal que $\vec{h} \leq \vec{k}$. Per la propietat 4 això és una contradicció.
- ($\Leftarrow$) Suposem que existeix $F_{\vec{h}}$ insatisfactible tal que per tot $F_{\vec{k}}$ insatisfactible $\vec{h} \not\leq \vec{k}$. Com que $F_{\vec{h}}$ és insatisfactible, s'ha de complir $\vec{h} \not\leq \vec{h}$, que és una contradicció.

□

Per tant, podem resoldre WCSP trobant un vector $\vec{h}$ de mínim cost que hiteji tots els elements de $\mathcal{C}$, és a dir, tots els soft cores de F.

$$WCSP(F) = \min_{\vec{h} \in \mathcal{HV}(\mathcal{C})} \sum_i h_i$$

Igual que en el cas de MaxSAT, no necessitem tenir $\mathcal{C}$ explícitament, ja que és implícit per F. Per tant, emprarem l'algoritme de IMHV.

Necessitem una funció CoreCSP$(F, \vec{h}, \vec{k})$ que retorni si $F_{\vec{h}}$ és satisfactible o no, i en cas que no ho sigui assigni a $\vec{k}$ un soft core de F tal que $\vec{h} \leq \vec{k}$. Això sempre és possible ja que $\vec{h}$ és pròpiament un soft core.

També necessitem una funció MHV$(\mathcal{K})$ que retorna un minimum cost hitting vector de $\mathcal{K}$ i una funció HV$(\mathcal{K})$ que retorna un hitting vector de $\mathcal{K}$.

A continuació veiem dos algoritmes per resoldre WCSP que són l'adaptació a WCSP dels algoritmes de IMHV que hem vist anteriorment. L'algoritme 7 és l'adaptació de l'algoritme 5 i l'algoritme 8 és l'adaptació de l'algoritme 6.

La complexitat temporal dels algoritmes és $O(|\mathcal{C}| \times m) = O((\prod_i |V_i|) \times m)$ on m és la complexitat temporal d'una crida a MHV.

L'algoritme bàsic implementant aquesta idea va ser proposat per primera vegada a [12].

Function `WCSP(`F`)`
begin
  $\mathcal{K} = \emptyset; \vec{h} = \vec{0}; lb = 0;$
  while *not CoreCSP*$(F, \vec{h}, \vec{k})$ **do**
    /* $\mathcal{K} \subseteq \mathcal{C}$, $\vec{h} \in MHV(\mathcal{K})$ */ ;
    $\mathcal{K} := \mathcal{K} \cup \{\vec{k}\};$
    $\vec{h} :=$ MHV$(\mathcal{K})$;
    $lb := cost(\vec{h})$;
  end
  return lb;
end

Algorithm 7: Algoritme que resol WCSP mitjançant lower bounds (lb).

Function `WCSP(`F`)`
begin
  $\mathcal{K} = \emptyset; \vec{h} = \vec{0}; lb = 0; ub = \infty;$
  while $lb < ub$ **do**
    /* $\mathcal{K} \subseteq \mathcal{C}$, $\vec{h} \in \mathcal{HV}(\mathcal{K})$, $lb \leq cost(MHV(\mathcal{C})) \leq ub$*/ ;
    if *CoreCSP*$(F, \vec{h}, \vec{k})$ **then**
      $ub := \min\{ub, cost(\vec{h})\};$
      $\vec{h} :=$ MHV$(\mathcal{K})$;
      $lb := \max\{lb, cost(\vec{h})\};$
    end
    else
      $\mathcal{K} := \mathcal{K} \cup \{\vec{k}\};$
      $\vec{h} :=$ HV$(\mathcal{K})$;
    end
  end
  return lb;
end

Algorithm 8: Algoritme que resol WCSP mitjançant lower i upper bounds (lub).

4 Implementació de MaxSAT

Hem decidit implementar el solver de MaxSAT en Python per poder fer ús de la llibreria *PySAT*, que ens ajudarà a resoldre alguns subproblemes.

Per implementar els algoritmes presentats a la secció 3.1.3, necessitem resoldre els següents sobproblemes:

- CoreSAT: un SAT solver que proporcioni un core quan la instància és insatisfactible.
- MHS: un algoritme que calculi el minimum cost hitting set.
- HS: un mètode que retorni un hitting set.

A continuació expliquem com hem resolt aquests sobproblemes.

Al llarg d'aquesta secció plantegem diverses maneres d'implementar alguns fragments. A la fase d'experimentació determinem quina versió és la millor.

Estructures de dades: La llibreria *PySAT*, a més de contenir solvers que ens resoldran subproblemes, ens proporciona estructures de dades per representar fórmules SAT i MaxSAT. En aquestes estructures, un literal positiu es representa amb un enter positiu; un literal negatiu es representa amb un enter negatiu; una clàusula es representa amb una llista de literals, és a dir, amb una llista d'enters. L'entrada del nostre algoritme és un fitxer que conté les dades d'una instància MaxSAT en format DIMACS CNF [51]. En general, el fitxer té extensió *.wcnf*. Una funció de *PySAT* s'encarrega de llegir el fitxer i generar l'estructura de dades. [52]

4.1 CoreSAT

Recordatori: Un CoreSAT és una funció CoreSAT($F, h,\ k$) que indica si la fórmula SAT $F - h$ és satisfactible i si no ho és, proporciona un core k insatisfactible tal que $k \subseteq F - h$.

La manera habitual en què els SAT solvers poden generar cores és mitjançant assumpcions. Considerem una fórmula SAT satisfactible $F = \{c_1, c_2, \ldots, c_e\}$. Un conjunt d'assumpcions és un conjunt de literals $A = \{l_1, l_2, \ldots, l_p\}$. Aleshores, un SAT solver basat en assumpcions SAT(F, A, K) diu si $F \cup A$ és satisfactible. Si no és satisfactible, retornarà $K \subseteq A$ tal que $F \cup K$ ja és insatisfactible.

Pel propòsit del nostre MaxSAT solver, la idea és modificar la fórmula F afegint un literal nou a cada clàusula,

$$F' = \{c_1 \vee b_1, c_2 \vee b_2, \ldots, c_e \vee b_e\}$$

on b_i són variables noves, anomenades *variables de bloqueig*. F' és sempre satisfactible ja que cada clàusula c_i queda desactivada per la seva variable de bloqueig b_i ($c_i \vee b_i$ és satisfactible perquè si $b_i = cert$, llavors $(c_i \vee b_i) = cert$). Per activar una clàusula c_i podem afegir l'assumpció $\neg b_i$. Clarament, $F' \cup A$ amb $A = \{\neg b_1, \neg b_2, \ldots, \neg b_e\}$ és equivalent a F. De manera similar, llavors $F - h$ és equivalent a $F' \cup A$ on $A = \{\neg b_i \mid \ c_i \in F - h\}$. Per tant, una manera d'implementar CoreSAT() és la que mostra l'algoritme 9.

La llibreria *PySAT* disposa d'un SAT solver basat en assumpcions. Utilitzatem el SAT solver per defecte de *PySAT*, que és *MiniSat 2.2*.

Estructures de dades: A l'àmbit de la funció MaxSAT no guardem la fórmula original $F = \{c_1, c_2, \ldots, c_e\}$ sinó la fórmula amb les variables de bloqueig $F' = \{c_1 \vee b_1, c_2 \vee b_2, \ldots, c_e \vee b_e\}$, ja que a cada crida al SAT solver necessitem F' i no F. Com que tenim una variable de bloqueig per cada clàusula, una variable de bloqueig (negada) $\neg b_i$ pot servir per identificar una clàusula c_i. Això és útil perquè la representació de les variables (enters) és més simple que la de les clàusules (llistes d'enters). D'aquesta manera, a la implementació final h i k no seran subconjunts de F sinó subconjunts de $B = \{\neg b_1, \neg b_2, \ldots, \neg b_e\}$. Amb aquestes estructures de dades, l'algoritme 9 queda transformat en l'algoritme 10.

Function `CoreSAT`(F, h, k)
/* PRE: $h \subseteq F$ */
/* POST: return $SAT(F - h)$ */
/* POST: $UNSAT(F - h) \Rightarrow (UNSAT(k) \wedge k \subseteq F - h)$ */
begin
  $F' := \{c_i \vee b_i \mid c_i \in F\}$;
  $A := \{\neg b_i \mid \ c_i \in F - h\}$;
  if *not SAT*(F', A, k') **then**
    $k := \{c_i \mid \neg b_i \in k'\}$;
    return $false$;
  end
  return $true$;
end

Algorithm 9: SAT solver que retorna cores.

Function `CoreSAT`(F', B, h, k)
/* PRE: $F' = \{c_i \vee b_i \mid \ c_i \in F\}$ */
/* PRE: $B = \{\neg b_i \mid c_i \in F\}$ */
/* PRE: $h \subseteq B$ */
/* POST: return $SAT(F' \cup B - h)$ */
/* POST: $UNSAT(F' \cup B - h) \Rightarrow (UNSAT(F' \cup k) \wedge k \subseteq B - h)$ */
begin
  $A := B - h$;
  return SAT(F', A, k) ;
end

Algorithm 10: SAT solver que retorna cores. Implementació amb les estructures de dades que hem fet servir.

4.2 MHS

Per resoldre el minimum cost hitting set problem hem provat tres mètodes:

- Emprant un algoritme que proporciona la llibreria *PySAT*
- Transformant MHS en programació lineal 0/1 i resolent-lo amb el solver *cplex*
- Implementant nosaltres un algoritme recursiu que resolgui MHS

Tal com veurem a la fase experimental, la opció més eficient ha resultat ser la segona.

Recordatori: Volem programar la funció MHS($\mathcal{K}$) on $\mathcal{K} \subseteq \{k \mid k \subseteq F\}$ i cada $u \in F$ té un cost $cost(u)$. La funció ha de retornar un hitting set de cost mínim. Un hitting set de $\mathcal{K}$ és un conjunt $h \subseteq F$ tal que per tot $k \in \mathcal{K}$, h hiteja k, és a dir, $h \cap k \neq \emptyset$. El cost d'un conjunt h és la suma dels costos dels seus elements $\sum_{u \in h} cost(u)$.

4.2.1 MHS a la llibreria *PySAT*

La llibreria *PySAT* té un mòdul anomenat *Hitman* dedicat a resoldre el problema MHS. Ho fan transformant MHS a MaxSAT i resolent amb el MaxSAT solver *RC2*. [52]

4.2.2 MHS en programació lineal (*cplex*)

En aquesta versió hem reduït MHS a programació lineal 0/1. L'hem codificat de les següent manera:

$$\begin{aligned} \text{minimitzar} \quad & c_1 h_1 + \cdots + c_n h_n \\ & a_{11} h_1 + \cdots + a_{1n} h_n \geq 1 \\ \text{subjecte a} \quad & \vdots \\ & a_{m1} h_1 + \cdots + a_{mn} h_n \geq 1 \end{aligned} \tag{1}$$

on h_i són variables 0/1 que indiquen si l'element $u_i \in F$ pertany al hitting set. c_i és el cost de u_i. a_{ji} és 1 si $u_i \in k_j$ (0 en cas contrari).

Recordem que $F = \{u_1, ..., u_n\}$ i $\mathcal{K} = \{k_1, ..., k_m\}$] Tal com podeu observar, la funció a minimitzar ($c_1 h_1 + \cdots + c_n h_n = \sum_{h_i=1} c_i = \sum_{u_i \in h} c_i = \sum_{u_i \in h} cost(u_i)$) és el cost del hitting set, i cada restricció ($a_{j1} h_1 + \cdots + a_{jn} h_n \geq 1$) obliga que hi hagi almenys un element u_i que pertanyi a k_j (llavors $a_{ij} = 1$) i al hitting set (llavors $h_i = 1$).

El solver de programació lineal que hem utilitzat és *cplex*, que fou desenvolupat per IBM. Té una API per Python, la llibreria *DOcplex*, que és la que hem fet servir. [53] *cplex* té una versió gratuïta i una versió gratuïta i una versió per subscripció. La versió gratuïta permet resoldre problemes petits i la versió per subscripció permet resoldre qualsevol problema. [54] A l'ordinador on hem executat els experiments tenim la versió per subscripció.

4.2.3 Algoritme recursiu per MHS

La nostra implementació de MHS segueix l'estructura de l'algoritme 11. A cada crida recursiva selecciona un conjunt $k \in \mathcal{C}$ per hitejar, ho prova amb cada element $u \in k$ i fa una crida recursiva eliminant de C tots els elements que contenen u, és a dir, tots els elements hitejats per $\{u\}$. La unió de $\{u\}$ amb el resultat de la crida recursiva és un hitting set de $\mathcal{C}$. Després de recórrer k retorna el hitting set de cost mínim. Cada HS trobat proporciona un upper bound (ub) que permetrà podar l'espai de cerca: no s'executarà cap fragment de codi si es pot assegurar que el cost del HS resultant serà superior o igual a ub.

Function `MHS`$(\mathcal{C}, ub)$
begin
  if $\mathcal{C} = \emptyset$ **then return** 0;
  $k = \mathcal{C}.pop()$;
  for $u \in k$ **do**
    if $cost(u) < ub$ **then**
      $\mathcal{C}' := \{k' \in \mathcal{C} \mid u \notin k'\}$;
      $h' := \{u\} \cup$ `MHS`$(\mathcal{C}', ub - cost(u))$;
      if $cost(h') < ub$ **then**
        $h := h'$;
        $ub := cost(h')$;
      end
    end
  end
  return h;
end

Algorithm 11: Algoritme pel minimum cost hitting set problem (versió *static*). $\mathcal{C}$ conté els conjunts que han de ser hitejats. ub és un upper bound de l'òptim. $\mathcal{C}.pop()$ selecciona el pròxim conjunt a hitejar.

La complexitat temporal de l'algoritme és $O(\prod_{k \in \mathcal{C}} |k|)$. L'alçada de l'arbre de cerca combinatòria que recorrem és com a mínim $|h|$ i com a màxim $min(|\mathcal{C}|, |F|)$, on h és la sortida de l'algoritme i $F = \bigcup_{k \in \mathcal{C}} k$. El factor de ramificació a cada crida recursiva és $|k|$, on $k \in \mathcal{C}$ és el conjunt seleccionat a la crida en qüestió. ub serveix per podar l'arbre de cerca.

Aquestes qüestions depenen en gran part de la funció $\mathcal{C}.pop()$. Ens interessa que seleccioni:

- Conjunts petits, per reduir el factor de ramificació

- Conjunts amb elements que apareguin en molts altres conjunts, perquè així amb un sol element hitegem molts conjunts i reduïm l'alçada de l'arbre de cerca.
- Conjunts amb elements costosos, perquè així *ub* decreix ràpid i poda l'arbre de cerca.

La funció *pop* selecciona amb els següents criteris:

1. El conjunt més petit
2. En cas d'empat, el conjunt k que maximitzi $\min\{aparicions(u) \mid u \in k\}$ on $aparicions(u) = |\{k' \in \mathcal{C} \mid u \in k'\}|$.
3. En cas d'empat, el conjunt k que maximitzi $\min\{cost(u) \mid u \in k\}$.

L'odre amb què el *for* recorre k també pot fer variar el temps t'execució. Els criteris d'odre són els següents:

1. Va abans l'element amb major nombre d'aparicions a altres conjunts.
2. En cas d'empat, va abans l'element amb menor cost.

Per evitar haver de calcular repetidament el nombre d'aparicions d'un element, guardem aquesta informació en un vector. El nombre d'aparicions depèn de $\mathcal{C}$, de manera que serà diferent a cada crida recursiva.

En aquesta primera versió de l'algoritme (anomenada *static*) a cada crida recursiva es fa una còpia de $\mathcal{C}$ amb les modificacions corresponents. Això és una ineficient des del punt de vista de memòria. Per això hem fet una nova versió (*dynamic*) on hi ha un sol $\mathcal{C}$ que modifiquem abans de la crida recursiva i guardem la informació necessària per restablir-lo després de la crida ($\mathcal{R}$). La complexitat espacial de la versió *static* és $O(\texttt{mem\_size}(\mathcal{C}) \times |\mathcal{C}|)$ mentre que la de la versió *dynamic* és simplement $O(\texttt{mem\_size}(\mathcal{C}))$.

Function `MHS`$(\mathcal{C}, ub)$
begin
  if $\mathcal{C} = \emptyset$ **then return** 0;
  $k = \mathcal{C}.pop()$;
  for $u \in k$ **do**
    if $cost(u) < ub$ **then**
      $\mathcal{R} := \{k' \in \mathcal{C} \mid \ u \in k'\}$;
      $\mathcal{C} := \mathcal{C} - \mathcal{R}$;
      $h' := \{u\} \cup$ `MHS`$(\mathcal{C}, ub - cost(u))$;
      $\mathcal{C} := \mathcal{C} \cup \mathcal{R}$;
      if $cost(h') < ub$ **then**
        $h := h'$;
        $ub := cost(h')$;
      end
    end
  end
  return h;
end

Algorithm 12: Algoritme pel minimum cost hitting set problem (versió *dynamic*).

Estructures de dades: A les dues versions (*static* i *dynamic*) és convenient que $\mathcal{C}$ no sigui una llista de conjunts sinó una llista d'índexs on cada índex identifica un conjunt que guardem a part. A la versió *static* això permet estalviar memòria i a la versió *dynamic* implica que els canvis es fan a una estructura de dades petita (llista d'índexs) mentre l'estructura gran (llista de conjunts) es manté invariable.

4.3 HS

Tal com hem avançat a 3.1.3, l'algoritme de MaxSAT amb lower i upper bounds (*lub*) necessita un mètode que retorni un hitting set (HS). Calcular un hitting set de $\mathcal{K} \subseteq \{k \mid k \subseteq F\}$ és trivial ja que el conjunt de tots els elements (F) és pròpiament un hitting set. Tanmateix, ens interessa trobar hitting sets amb costos petits. Ho farem amb una heurística *greedy*.

Fins ara, per simplificar, hem considerat que MHS és una funció però realment és una classe semblant a la que es mostra a l'algoritme 13. La classe s'encarrega de guardar els conjunts per hitejar $\mathcal{K}$ i un hitting set h. La funció `getMHS` calcula un MHS i guarda el resultat a h. Quan s'afegeix un nou conjunt k a $\mathcal{K}$ (amb la funció `addSet`) s'actualitza h per tal que segueixi sent un HS. Si h ja hiteja k no cal fer res més. En cas contrari, s'afegeix a h l'element de k amb menor cost. La funció `getHS` es limita a retornar h, que ja és un HS.

En resum, el HS es calcula de forma *greedy* (afegint els elements de menor cost de cada conjunt nou) a partir de l'últim MHS computat.

Class `MHS`
begin
  /* Inv: $h \in \mathcal{HS}(\mathcal{K})$ */
  Variable $\mathcal{K} := \emptyset$;
  Variable $h := \emptyset$;
  Function `getMHS()`
  begin
    ... /* h :=MHS($\mathcal{K}$) */
    return h ;
  end
  Function `getHS()`
  begin
    return h ;
  end
  Function `addSet(`k`)`
  begin
    $\mathcal{K} := \mathcal{K} \cup \{k\}$;
    if $k \cap h = \emptyset$ **then**
      $u := \text{argmin}_{u \in k}\, cost(u)$;
      $h := h \cup \{u\}$;
    end
  end
end

Algorithm 13: Classe que gestiona que gestiona MHS i HS. La funció `getMHS` retorna un minimum cost hitting set de $\mathcal{K}$. La funció `getHS` retorna un hitting set de $\mathcal{K}$. La funció `addSet` afegeix un conjunt a $\mathcal{K}$.

4.4 MaxSAT

Amb les estructures de dades vistes en el present capítol, reescrivim els algoritmes presentats a 3.1.3. A més, podem suposar que quan CoreSAT retorna cert, proporciona una assignació a les variables (*assign*) que satisfà la fórmula. Així la funció MaxSAT, pot retornar el cost mínim i l'assignació de cost mínim.

```
Function MaxSAT(F)
begin
    F' = {c_i ∨ b_i | c_i ∈ F} ;
    B = {¬b_i | c_i ∈ F} ;
    K = new MHS(∅) ;
    h = ∅; lb = 0;
    while not CoreSAT(F', B, h, k, assign) do
        K.addSet(k);
        h := K.getMHS();
        lb := cost(h);
    end
    return (lb, assign);
end
```

Algorithm 14: Algoritme que resol MaxSAT mitjançant lower bounds (lb). Implementació amb les estructures de dades que hem fet servir.

```
Function MaxSAT(F)
begin
    F' = {c_i ∨ b_i | c_i ∈ F} ;
    B = {¬b_i | c_i ∈ F} ;
    K = new MHS(∅) ;
    h = ∅; lb = 0; ub = ∞;
    while lb < ub do
        if CoreSAT(F', B, h, k, assign) then
            ub := min{ub, cost(h)};
            h := K.getMHS();
            lb := max{lb, cost(h)};
        end
        else
            K.addSet(k);
            h := K.getHS();
        end
    end
    return (lb, assign);
end
```

Algorithm 15: Algoritme que resol MaxSAT mitjançant lower i upper bounds (lub). Implementació amb les estructures de dades que hem fet servir..

4.5 Resum de versions

Segons la implementació de la funció MaxSAT, tenim dues opcions:

- **lb**: versió amb lower bounds
- **lub**: versió amb lower i upper bounds

Segons la implementació de MHS tenim les següents opcions:

- **hitman**: resolució amb el solver *Hitman* de la llibreria *PySAT*
- **cplex**: reducció de MHS a programació lineal 0/1 i resolució amb el solver *cplex*
- Algoritme recursiu (implementació pròpia):
  - **static**: a cada crida recursiva es fa una còpia del conjunt abans de modificar-lo
  - **dynamic**: a cada crida recursiva es modifica el conjunt sense copiar-lo.

La combinació d'opcions produeix un total de $2 \times 4 = 8$ prototips que analitzarem a la secció d'experiments (7.3) i veurem que el millor prototip és la combinació de *lub* i *cplex*.

4.6 Exemple d'execució

El nostre solver integra totes les versions que hem vist, les podem especificar pels paràmetres de consola.

```
$ python3 MaxSAT.py --help
usage: MaxSAT.py [-h] [-m] [-a {lb,lub}] [-s {hitman,cplex,static,dynamic}] file

positional arguments:
  file                  path to a .wcnf file

optional arguments:
  -h, --help            show this help message and exit
  -m, --model           print optimal model
  -a {lb,lub}, --algorithm {lb,lub}
                        default: lub
  -s {hitman,cplex,static,dynamic}, --hit_solver {hitman,cplex,static,dynamic}
                        default: cplex
```

A continuació es mostra l'exemple d'execució amb la instància CP13-404 i les opcions *lub* i *hitman*. A cada iteració el solver imprimeix una línia amb informació de l'estat de la resolució.

```
$ python3 MaxSAT.py ../instances_wcnf/CP13-404.wcnf -m -a lub -s hitman
 HS,  1 cores found,  hitting set size = 1
 HS,  2 cores found,  hitting set size = 2
 HS,  3 cores found,  hitting set size = 3
 HS,  4 cores found,  hitting set size = 4
 HS,  5 cores found,  hitting set size = 5
 HS,  6 cores found,  hitting set size = 6
 HS,  7 cores found,  hitting set size = 7
 HS,  8 cores found,  hitting set size = 8
 HS,  9 cores found,  hitting set size = 9
 HS,  10 cores found,  hitting set size = 10
 HS,  11 cores found,  hitting set size = 11
```

```
 HS,  12 cores found,  hitting set size = 12
 HS,  13 cores found,  hitting set size = 13
 HS,  14 cores found,  hitting set size = 14
 HS,  15 cores found,  hitting set size = 15
 HS,  16 cores found,  hitting set size = 16
 HS,  17 cores found,  hitting set size = 17
 HS,  18 cores found,  hitting set size = 18
 HS,  19 cores found,  hitting set size = 19
 HS,  20 cores found,  hitting set size = 20
 HS,  21 cores found,  hitting set size = 21
 HS,  22 cores found,  hitting set size = 22
 HS,  23 cores found,  hitting set size = 23
 HS,  24 cores found,  hitting set size = 24
 HS,  25 cores found,  hitting set size = 25
 HS,  26 cores found,  hitting set size = 26
 HS,  27 cores found,  hitting set size = 27
 HS,  28 cores found,  hitting set size = 28
 HS,  29 cores found,  hitting set size = 29
 HS,  30 cores found,  hitting set size = 30
 HS,  31 cores found,  hitting set size = 31
 HS,  32 cores found,  hitting set size = 32
 HS,  33 cores found,  hitting set size = 33
 HS,  34 cores found,  hitting set size = 34
 HS,  35 cores found,  hitting set size = 35
 HS,  36 cores found,  hitting set size = 36
 HS,  37 cores found,  hitting set size = 37
 HS,  38 cores found,  hitting set size = 38
 HS,  39 cores found,  hitting set size = 39
 HS,  40 cores found,  hitting set size = 40
 HS,  41 cores found,  hitting set size = 41
 HS,  42 cores found,  hitting set size = 42
 HS,  43 cores found,  hitting set size = 43
 HS,  44 cores found,  hitting set size = 44
 HS,  45 cores found,  hitting set size = 45
 HS,  46 cores found,  hitting set size = 46
 HS,  47 cores found,  hitting set size = 47
 HS,  48 cores found,  hitting set size = 48
 HS,  49 cores found,  hitting set size = 49
 HS,  50 cores found,  hitting set size = 50
 HS,  51 cores found,  hitting set size = 51
MHS,  51 cores found,  hitting set size = 11,  lower bound = 15,  upper bound = 58
 HS,  52 cores found,  hitting set size = 12
 HS,  53 cores found,  hitting set size = 13
 HS,  54 cores found,  hitting set size = 14
 HS,  55 cores found,  hitting set size = 15
 HS,  56 cores found,  hitting set size = 16
 HS,  57 cores found,  hitting set size = 17
 HS,  58 cores found,  hitting set size = 18
 HS,  59 cores found,  hitting set size = 19
```

[...]

```
 HS,  1680 cores found,  hitting set size = 39
MHS,  1680 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1681 cores found,  hitting set size = 38
 HS,  1682 cores found,  hitting set size = 39
 HS,  1683 cores found,  hitting set size = 40
 HS,  1684 cores found,  hitting set size = 41
MHS,  1684 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1685 cores found,  hitting set size = 38
```

```
 HS,  1686 cores found,  hitting set size = 39
 HS,  1687 cores found,  hitting set size = 40
 HS,  1688 cores found,  hitting set size = 41
MHS,  1688 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1689 cores found,  hitting set size = 38
 HS,  1690 cores found,  hitting set size = 39
MHS,  1690 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1691 cores found,  hitting set size = 38
 HS,  1692 cores found,  hitting set size = 39
 HS,  1693 cores found,  hitting set size = 40
 HS,  1694 cores found,  hitting set size = 41
MHS,  1694 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1695 cores found,  hitting set size = 38
 HS,  1696 cores found,  hitting set size = 39
 HS,  1697 cores found,  hitting set size = 40
 HS,  1698 cores found,  hitting set size = 41
MHS,  1698 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1699 cores found,  hitting set size = 38
 HS,  1700 cores found,  hitting set size = 39
 HS,  1701 cores found,  hitting set size = 40
 HS,  1702 cores found,  hitting set size = 41
 HS,  1703 cores found,  hitting set size = 42
 HS,  1704 cores found,  hitting set size = 43
MHS,  1704 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1705 cores found,  hitting set size = 38
 HS,  1706 cores found,  hitting set size = 39
 HS,  1707 cores found,  hitting set size = 40
 HS,  1708 cores found,  hitting set size = 41
 HS,  1709 cores found,  hitting set size = 42
 HS,  1710 cores found,  hitting set size = 43
MHS,  1710 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1711 cores found,  hitting set size = 38
 HS,  1712 cores found,  hitting set size = 39
MHS,  1712 cores found,  hitting set size = 37,  lower bound = 46,  upper bound = 47
 HS,  1713 cores found,  hitting set size = 38
 HS,  1714 cores found,  hitting set size = 39
 HS,  1715 cores found,  hitting set size = 40
MHS,  1715 cores found,  hitting set size = 38,  lower bound = 47,  upper bound = 47
Solution found with cost 47

optimal model:
-1 2 -3 -4 5 6 7 -8 -9 -10 -11 12 -13 14 -15 -16 -17 -18 -19 20 21 22 23 24 25 -26 27
-28 29 -30 -31 -32 -33 -34 35 36 37 -38 39 40 -41 42 43 -44 -45 -46 47 -48 49 -50 -51
-52 -53 -54 -55 56 57 -58 59 -60 -61 -62 63 -64 65 66 -67 68 69 -70 71 72 73 74 75 76
-77 -78 -79 80 81 -82 83 84 85 86 -87 -88 -89 90 -91 -92 -93 94 95 -96 -97 -98 99 100
101 102 103 104 105 -106 -107 108 -109 -110 111 -112 113 -114 115 -116 -117 -118 -119
-120 121 -122 123 -124 -125 -126 -127 -128 129 130 -131 -132 -133 134 -135 -136 137
-138 139 140 -141 -142 -143 144 145 146 -147 -148 -149 150 151 152 153 -154 155 -156
-157 158 -159 -160 -161 -162 -163 -164 165

cost: 47
real_time: 1.2993977069854736
cpu_time: 1.265023508
```

5 Implementació de WCSP

Hem decidit implementar el solver de WCSP en C++.

Per implementar els algoritmes presentats a la secció 3.2.3, necessitem resoldre els següents sobproblemes:

- CoreCSP: un CSP solver que proporcioni un core quan la instància és insatisfactible.
- MHV: un algoritme que calculi el minimum cost hitting vector.
- HV: un mètode que retorni un hitting vector.

A continuació expliquem com hem resolt aquests sobproblemes.

Al llarg d'aquesta secció plantegem diverses maneres d'implementar alguns fragments. A la fase d'experimentació determinem quina versió és la millor.

Estructures de dades:

Hem implementat estructures de dades per representar problemes WCSP. Tenim una classe per representar funcions de cost i una per representar instàncies WCSP. Per exemple, una funció de cost es representa amb un vector que té una posició per cada possible input de la funció. L'entrada del nostre algoritme és un fitxer en format WCSP (tal com el defineixen a [55]) que conté les dades d'una instància. El programa llegeix el fitxer i el transforma amb les estructures de dades esmentades.

El format WCSP assumeix que tots els dominis són nombres naturals consecutius començant per 0, ja que de fet el valor de cada element del domini és irrellevant. Per exemple, un domini de mida 4 sempre és $\{0, 1, 2, 3\}$. En el context d'una funció de cost sovint serà convinent representar els costos com a índexs d'un vector de costos que conté tots els outputs de la funció ordenats. Per simplificar l'explicació, ignorarem aquests fets.

5.1 CoreCSP

Recordatori: Un CoreCSP és una funció CoreCSP$(F, \vec{h}, \vec{k})$, on F és una fórmula WCSP, que indica si la fórmula CSP $F_{\vec{h}}$ és satisfactible i si no ho és, proporciona un soft core $\vec{k}$ tal que $F_{\vec{k}}$ és insatisfactible i $\vec{h} \leq \vec{k}$. $F_{\vec{h}}$ és l'enduriment de F amb $\vec{h}$. $F_{\vec{h}}$ és una fórmula CSP formada per una restricció $f_i^{h_i}$ per cada funció f_i de F, on $f_i^{h_i}(t) = (f_i(t) \leq h_i)$.

A continuació veurem dues maneres d'implementar-ho (CSP com a SAT i Forward Checking) i a una millora aplicable a les dues versions, que consisteix a proporcionar més d'un core.

Tal com veurem a la fase experimental, la opció més eficient ha resultat ser la primera.

5.1.1 CSP com a SAT

Com a problema NP-complet, CSP es pot reduir a SAT, així que hem reduït CSP a SAT per resoldre'l amb el SAT solver *CaDiCaL* [56 - 58], guanyador de la SAT Race 2019.

Siguin $X = \{x_1, x_2, \ldots, x_n\}$ les variables del CSP i $D = \{d_1, d_2, \ldots, d_n\}$ els dominis del CSP, per cada variable CSP $x_i \in X$ i cada valor del domini de la variable CSP $a \in d_i$ tenim una variable SAT x_{ia} que és certa si x_i pren el valor a.

Per cada variable x_i tenim una clàusula que indica que x_i ha de prendre un valor del domini:

$$\forall x_i \in X : \bigvee_{a \in d_i} x_{ia}$$

Per cada variable x_i tenim un conjunt de clàusules que indiquen que x_i no pot prendre més d'un valor del domini:

$$\forall x_i \in X : \forall a, b \in d_i \text{ t.q. } a \neq b : (\neg x_{ia} \vee \neg x_{ib})$$

Al llarg de l'execució de WCSP cal resoldre diferents CSP. Per no haver de fer la reducció cada vegada, la fórmula SAT és genèrica pel problema WCSP i concretem el CSP mitjançant d'assumpcions a les *variables de bloqueig* següents. Siguin $F = \{f_1(Y_1), \ldots, f_e(Y_e)\}$ (amb $Y_i \subseteq X$) les funcions de cost del WCSP, per cada funció $f_i(Y_i) \in F$ i possible output finit positiu de la funció $c \in \{f_i(t) \mid t \text{ assignació de } Y_i\} - \{\infty, 0\}$, tenim una variable de bloqueig b_{ic}. Per un CSP $F_{\vec{h}}$ la variable b_{ic} que és certa quan $c \leq h_i$. Noteu que no calen les variables b_{i0} i $b_{i\infty}$ ja que sempre $0 \leq h_i$ i mai $\infty \leq h_i$.

Per cada funció de cost $f_i(Y_i) \in F$ i assignació t de Y_i, tenim una clàusula que prohibeix l'assignació t i s'activa quan $f_i(t) > h_i$:

$$\forall f_i(Y_i) \in F : \forall t \text{ assignació de } Y_i : \begin{cases} (\bigvee_{a \in t} \neg x_{ia}) \vee b_{if_i(t)} & \text{si } 0 < f_i(t) < \infty \\ \bigvee_{a \in t} \neg x_{ia} & \text{si } f_i(t) = \infty \end{cases} \tag{2}$$

Si $f_i(t) \leq h_i$, llavors $b_{if_i(t)} = cert$ i la resta de la clàusula $\bigvee b_{if_i(t)}$ és irrellevant, per tant, t es permet. Si $f_i(t) > h_i$, llavors $b_{if_i(t)} = fals$ i la resta de la clàusula $\bigvee b_{if_i(t)}$ s'ha de complir i, per tant, t es prohibeix.

El SAT solver *CaDiCaL* té l'opció de tenir assumpcions. Tal com vist a 4.1, un SAT solver basat en assumpcions és una funció $\text{SAT}(F, A, K)$ on F és una fórmula SAT i A és un conjunt de literals. Indica si $F \cup A$ és satisfactible i si no és satisfactible, retorna K tal que $F \cup K$ ja és insatisfactible. L'algoritme 16 mostra com es resol un CSP partint de la fórmula SAT F' que s'ha generat al principi de l'execució de WCSP tal com hem indicat. Les assumpcions són literals de les variables de bloqueig. Generem el conjunt d'assumpcions A a partir de $\vec{h}$ i quan el problema és insatisfactible generem el core $\vec{k}$ a partir de K inversament a com hem generat les assumpcions.

Function `CoreCSP_SAT`$(F', \vec{h}, \vec{k})$
/* PRE: F' és una fórmula SAT, adaptació d'un WCSP F tal com hem vist */
/* PRE: $|F| = |\vec{h}| = e$ */
/* POST: return $true \Leftrightarrow F_{\vec{h}}$ satisfactible */
/* POST: $F_{\vec{h}}$ insatisfactible $\Rightarrow$ ($F_{\vec{k}}$ insatisfactible $\wedge$ $\vec{h} \leq \vec{k}$) */
begin
  $A := \{b_{ic} \mid \ c \leq h_i\} \cup \{\neg b_{ic} \mid \ c > h_i\}$;
  if *not SAT(*F', A, K*)* **then**
    for $i \in 1..e$ **do**
      $k_i := \max\{c \mid \neg b_{ic} \notin K\}$;
    end
    return $false$;
  end
  return $true$;
end

Algorithm 16: CSP solver que retorna un core quan el problema és insatisfactible. Implementació havent reduït CSP a SAT.

5.1.2 Forward Checking

L'algoritme de Forward Checking va provant valors per les variables, i per les variables no assignades descarta les valors incompatibles amb els valors actuals. Quan una variables es queda sense valors fa backtrack.

S'entén bastant quan l'apliquem al problema de les n reines. Aquest problema consisteix en col·locar n reines a un tauler $n \times n$ de tal manera que no hi hagi més d'una reina a una mateixa fila, columna o diagonal. Podem assumir que hi haurà una reina a cada columna. La figura 25 mostra una resolució per força bruta, provant totes les opcions i fent backtrack quan hi ha un conflicte. La

figura 26 mostra una resolució per Forward Checking, on a cada assignació es descarten les caselles incompatibles amb l'assignació. Es fa backtrack quan una columna no assignada es queda sense caselles disponibles.

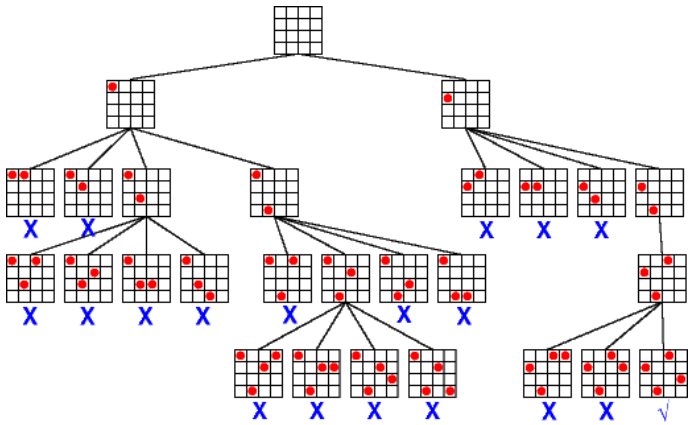

Figura 25: El problema de les 4 reines resolt per força bruta, partint de la base que hi ha una reina per columna. Font: [59]

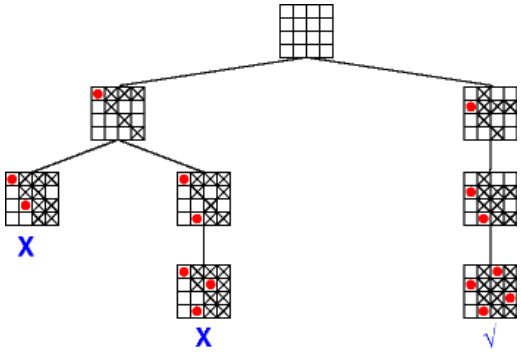

Figura 26: El problema de les 4 reines resolt per Forward Checking, partint de la base que hi ha una reina per columna. Font: [59]

L'algoritme 17 és una simplificació de l'algoritme de Forward Checking aplicat a CoreCSP. En aquest pseudocodi, per simplificar, s'assumeix que totes les restriccions són binàries però el codi final accepta tot tipus de restriccions. Quan assigna un valor a a una variable x_i descara el valor b per una variable x_j si hi ha una restricció $c_{ij}(x_i, x_j)$ tal que ho prohibeix $(c_{ij}(a, b) = false)$. Descartar un valor vol dir eliminar-lo del domini. Es fa backtrack quan algun domini es queda buit.

5.1.3 Més d'un core

A través d'experiments hem detectat que WCSP inverteix molt més temps en resoldre MHV que no pas en CoreCSP (en veurem més detalls al capítol 7 d'experimentació). Arran d'això hem decidit que a cada iteració en el lloc de CoreCSP s'executarà CoresCSP, que executa repetidament CoreCSP amb dos objectius: proporcionar diversos cores i que els cores siguin com més grans possibles. L'objectiu final és que MHV disposi de millors cores per acostar-se al resultat final i reduir el nombre total d'iteracions. L'algoritme 18 mostra la implementació de CoresCSP$(F, \vec{h}, C)$ assumint l'existència de CoreCSP$(F, \vec{h}, \vec{k})$ tal com l'hem definit prèviament.

El bucle intern s'encarrega d'augmentar el core de $F_{\vec{h}}$. A cada iteració prova d'endurir el WCSP amb un vector $\vec{h}'$ major que el core $\vec{k}$ (augmenta una component) i si $F_{\vec{h}'}$ encara és insatisfactible tindrem un core major $(\vec{k} < \vec{h}' \leq \vec{k}_{\text{següent}})$. El procés s'atura quan $F_{\vec{h}'}$ és insatisfactible. Ens quedem amb l'últim core trobat, que és el més gran. La resta són redundants[8].

[8] Un vector $\vec{k}'$ és redundant a $\vec{k}$ si $\vec{k} \geq \vec{k}'$, per tant, $\forall \vec{h} : \vec{h} \geq \vec{k} \rightarrow \vec{h} \geq \vec{k}'$, i per tant, $MHV(\{..., \vec{k}\}) = MHV(\{..., \vec{k}, \vec{k}'\})$, és a dir, $\vec{k}'$ és irrellevant pel MHV quan tenim $\vec{k}$.

Function `ForwardChecking`$(X, D, C, core)$
begin
  if $|X| = 0$ **then return** $true$;
  $x_i := X.pop()$;
  for $a \in d_i$ **do**
    $D' := D$;
    if `LookAhead`$(X, x_i, a, D', C, core)$ *and* `ForwardChecking`$(X, D', C, core)$ **then**
      return $true$;
  end
  $X.push(x_i)$;
  return $false$;
end
Function `LookAhead`$(X, x_i, a, D', C, core)$
begin
  for $c_{ij} \in C$ **do**
    for $b \in d_j$ *s.t. not* $c_{ij}(a, b)$ **do** d'_j.pop(b);
    if $|d'_j| = 0$ **then** $core$.push$(ij$; **return** false;
  end
  for $c_{ij} \in C$ *s.t.* $|d_j| \neq |d'_j|$ **do** $core$.push(ij);
  return $true$;
end

Algorithm 17: Forward Checking que resol CoreCSP. L'entrada és un CSP binari on X és el conjunt de variables, D és el cinjunt de dominis i C és el conjunt de restriccions binàries. Si hi ha una restricció entre x_i i x_j, s'indica c_{ij} (o equivalent, c_{ji}). El domini $d_i \in D$ és el domini de la variable $x_i \in X$. El paràmetre *core* és un paràmetre de sortida on si el CSP no és satisfactible, el (hard) core es retorna com un conjunt de parells (i, j).

Function `CoresCSP`$(F, \vec{h}, C)$
/* PRE: $|F| = |\vec{h}| = e$ */
/* POST: return $true \Leftrightarrow F_{\vec{h}}$ satisfactible */
/* POST: $F_{\vec{h}}$ insatisfactible $\Rightarrow (\forall \vec{k} \in C : F_{\vec{k}}$ insatisfactible $\wedge\ \vec{h} \leq \vec{k})$ */
begin
  $C := \emptyset$;
  $unsat := true$;
  while $unsat$ **do**
    $unsat := false$;
    $\vec{h}' := \vec{h}$;
    while *not* $CoreCSP(F, \vec{h}', \vec{k})$ *and* $\vec{k} \neq \vec{\infty}$ **do**
      $unsat := true$;
      $\vec{h}' := \vec{k}$;
      $i := \text{argmin}_i\, h_i$;
      $h'_i := \min\{f_i(t) \mid f_i(t) > h'_i$; t assignació de $Y_i\}$;
      /* següent cost de la funció $f_i \in F$ */
    end
    if $unsat$ **then**
      $C := C \cup \{\vec{k}\}$;
      for $i \in 1..e$ **do**
        if $k_i < \infty$ **then** $h_i := \infty$;
      end
    end
  end
  return $(C == \emptyset)$; /*Si no hem trobat cap core, és satisfactible*/
end

Algorithm 18: CSP solver que proporciona un conjunt de cores quan el problema és insatisfactible. CoreCSP$(F, \vec{h}, \vec{k})$ és un CSP solver que retorna un core $\vec{k}$ quan $F_{\vec{h}}$ és insatisfactible.

El bucle extern s'encarrega de trobar diversos cores independents[9]. Fixeu-vos que una funció f endurida amb ∞, f^∞ és una restricció que ho accepta tot. Això vol dir que, posem per cas, un WCSP F endurit amb $(0, \infty, 1, 3)$ tindria "desactivada" la segona funció. Quan el bucle intern ha trobat un core $\vec{k}$, modifiquem $\vec{h}$ per tal que $F_{\vec{h}}$ tingui desactivades les funcions que $F_{\vec{k}}$ té activades.

5.2 MHV

Recordatori: Volem programar una funció MHV($\mathcal{K}$) on $\mathcal{K}$ és un conjunt de vectors e-dimensionals. La funció ha de retornar un hitting vector de cost mínim. Un hitting vector de $\mathcal{K}$ és un vector e-dimensional $\vec{h}$ que per tot $\vec{k} \in \mathcal{K}$, $\vec{h}$ hiteja $\vec{k}$, és a dir, $\vec{h} \not\leq \vec{k}$, és a dir $\exists i : h_i > k_i$. El cost d'un vector $\vec{h}$ és la suma dels seus components $\sum_i h_i$. Els dominis dels components són $V_1, V_2, \ldots, V_e$, és a dir $h_i, k_i \in V_i$.

D'entrada tenim dues opcions per resoldre MHV, similars a les que hem implementat per MHS: reduir-lo a programació lineal i implementar un algoritme recursiu. En el cas de MHS hem vist als experiments preliminars que l'algoritme recursiu és clarament ineficient, per tant, ja no hem provat d'implementar-lo per MHV. En el cas de MHS també teníem l'opció d'emprar un solver existent específic per MHS. Però a diferència de MHS, MHV és un problema que ens hem inventat pel propòsit d'aquest projecte, per tant, no existeix cap solver específic per MHV.

Per tot això, hem fet una sola implementació de MHV: reduir-lo a programació lineal entera 0/1 i resoldre-ho amb el solver *cplex*.

El model té una variable 0-1 H_{ij} per cada element $v_{ij} \in V_i$ de cada domini V_i. v_{ij} és el j-èssim element (en ordre creixent) del domini V_i. H_{ij} és 1 quan $h_i > v_{ij}$ ($\vec{h}$ és el hitting vector) i 0 en cas contrari.

Per mantenir la consistència cal introduir les restriccions que indiquen que $H_{i(j+1)} > H_{ij}$ ja que $h_i > v_{i(j+1)} \rightarrow h_i > v_{ij}$ perquè $v_{i(j+1)} > v_{ij}$:

$$\forall ij : H_{i(j+1)} > H_{ij}$$

Volem minimitzar el cost, que és $\sum_i h_i$. Sabem que $h_i = v_{i0} + \sum_j H_{ij}(v_{ij} - v_{i(j-1)})$, per tant, el model ha de

$$\text{minimitzar } \sum_{ij} H_{ij}(v_{ij} - v_{i(j-1)})$$

Per cada vector $\vec{k} \in \mathcal{K}$ s'ha de complir que $\vec{h}$ hiteja $\vec{k}$, és a dir $\exists i : h_i > k_i$, és a dir, $\exists i : H_{i(k'_i+1)} = 1$ on $\vec{k'}$ és la representació de $\vec{k}$ amb índexs dels dominis ($\forall i : k_i = v_{ik'_i}$). De fet, a la implementació els vectors són representats com $\vec{k'}$, amb índexs dels dominis.

on j és l'índex de k_i al domini ($k_i = v_{ij}$). Per tant cal afegir les següents restriccions:

$$\forall \vec{k} \in \mathcal{K} : \sum_i H_{i(k'_i+1)} \geq 1$$

5.3 HV

Tal com hem avançat a 3.2.3, l'algoritme de WCSP amb lower i upper bounds (*lub*) necessita un mètode que retorni un hitting vector (HV). Calcular un hitting vector és trivial ja que, si existeix, $(\max(V_1), ..., \max(V_e))$ és un hitting vector. Tot i així, ens interessa trobar hitting vectors amb components petits, que s'acostin al MHV.

Veurem tres maneres de fer-ho:

[9] Dos vectors són independents si no són redundants.

- De forma *greedy*
- Afegint una restricció al model lineal de MHV
- Aturant el model quan ha trobat un HV millor que el que tenim (*callback*)

5.3.1 HV *greedy*

Fins ara, per simplificar, hem considerat que MHV és una funció però realment és una classe semblant a la que es mostra a l'algoritme 19. La classe s'encarrega de guardar els vectors per hitejar $\mathcal{K}$ i un hitting vector $\vec{h}$. La funció `getMHV` calcula un MHV i guarda el resultat a $\vec{h}$. Quan s'afegeix un nou vector $\vec{k}$ a $\mathcal{K}$ (amb la funció `addVector`) s'actualitza $\vec{h}$ per tal que segueixi sent un HV. Si $\vec{h}$ ja hiteja $\vec{k}$ no cal fer res més. En cas contrari s'incrementa un component h_i de $\vec{h}$ per tal que $h_i > k_i$ i per tant $\vec{h}$ hiteji $\vec{k}$. Es fa de manera que suposi el mínim increment possible.

Class `MHV`
begin
  /* Inv: $\vec{h} \in \mathcal{HV}(\mathcal{K})$ */
  Variable $\mathcal{K} := \emptyset$;
  Variable $\vec{h} := \vec{0}$;
  Function `getMHV()`
  begin
    ... /* h :=MHV($\mathcal{K}$) */
    return $\vec{h}$;
  end
  Function `getHV()`
  begin
    return $\vec{h}$;
  end
  Function `addVector(`$\vec{k}$`)`
  begin
    $\mathcal{K} := \mathcal{K} \cup \{\vec{k}\}$;
    if $\vec{h} \leq \vec{k}$ **then**
      $i := \text{argmin}_{i:k_i \geq h_i} \min\{v_{ij} \mid v_{ij} > k_i\} - h_i$; /*component que suposa un increment menor*/
      $h_i := \min\{v_{ij} \mid v_{ij} > k_i\}$; /*element del domini posterior a k_i*/
    end
  end
end

Algorithm 19: Classe que gestiona que gestiona MHV i HV. La funció `getMHV` retorna un minimum cost hitting vector de $\mathcal{K}$. La funció `getHV` retorna un hitting vector de $\mathcal{K}$. La funció `addVector` afegeix un vector a $\mathcal{K}$.

5.3.2 Model lineal MHV amb una restricció addicional

En aquesta versió cada cop que volem calcular un HV executem el model per MHV però enlloc de minimitzar el cost, posem la restricció de que el cost sigui menor a l'últim cost trobat.

5.3.3 Model lineal MHV amb *callback*

En aquesta versió cada cop que volem calcular un HV executem el model per MHV però abortem l'execució quan el model ha trobat una solució amb cost menor a l'últim cost trobat. Ho fem implementant una funció *callback* a *cplex*.

5.4 WCSP

Amb les estructures de dades i optimitzacions vistes en el present capítol, reescrivim els algoritmes presentats a 3.2.3. A més, podem suposar que quan CoresCSP retorna cert, proporciona una assignació a les variables (*assign*) que satisfà el problema. Així la funció WCSP, pot retornar el cost mínim i l'assignació de cost mínim.

Function `WCSP(`F`)`
begin
  $\mathcal{K}$ = `new MHV(`$\emptyset$`)` ;
  $\vec{h} = \vec{0}; lb = 0;$
  while *not CoresCSP*$(F, \vec{h}, C, assign)$ **do**
    for $\vec{k} \in C$ **do** $\mathcal{K}$`.addVector(`$\vec{k}$`)`;
    $\vec{h}$:= $\mathcal{K}$`.getMHV();`
    $lb := cost(h)$;
  end
  return $(lb, assign)$;
end

Algorithm 20: Algoritme que resol WCSP mitjançant lower bounds (lb). Implementació amb les estructures de dades que hem fet servir.

Function `WCSP(`F`)`
begin
  $\mathcal{K}$ = `new MHV(`$\emptyset$`)` ;
  $\vec{h} = \vec{0}; lb = 0; ub = \infty;$
  while $lb < ub$ **do**
    if *not CoresCSP*$(F, \vec{h}, C, assign)$ **then**
      $ub := \min\{ub, cost(\vec{h})\}$;
      $\vec{h}$:= $\mathcal{K}$`.getMHV();`
      $lb := \max\{lb, cost(\vec{h})\}$;
    end
    else
      for $\vec{k} \in C$ **do** $\mathcal{K}$`.addVector(`$\vec{k}$`)`;
      $\vec{h}$:= $\mathcal{K}$`.getHV();`
    end
  end
  return $(lb, assign)$;
end

Algorithm 21: Algoritme que resol WCSP mitjançant lower i upper bounds (lub). Implementació amb les estructures de dades que hem fet servir.

5.5 Resum de versions

Segons la implementació de CoreCSP dues opcions:

- **fc**: algoritme de Forward Checking (implementació pròpia)
- **sat**: reducció de CSP a SAT resolució amb el solver *CaDiCaL*

Segons la implementació de la funció WCSP, tenim dues opcions:

- **lb**: versió amb lower bounds
- **lub**: versió amb lower i upper bounds

Segons la implementació de HV (només aplicable a la versió *lub* tenim tres opcions:

- **greedy**: a mesura que s'afegeixen vectors es va calculant voraçment un HV
- **model**: crida al model lineal de MHV sense l'objectiu de minimitzar el cost però amb la restricció de millorar-lo
- **callback**: crida al model lineal de MHV i abortar l'execució quan s'ha trobat un HV millor que l'anterior

La combinació d'opcions produeix un total de $2 \times (1 + 1 \times 3) = 8$ prototips que analitzarem a la secció d'experiments (7.4) i veurem que el millor prototip és la combinació de *sat*, *lub* i *greedy*.

5.6 Exemple d'execució

El nostre solver integra totes les versions que hem vist, les podem especificar pels paràmetres de consola.

```
$ ./main --help
USAGE:
./main file.wcsp [-s] [-c sat|fc] [-a lb|lub] [-v greedy|model|callback] [-h]
    file.wcsp : input WCSP file
    -s|--solution : print optimal solution
    -c|--csp-solver   sat|fc                (default: sat)
    -a|--alg-version  lb|lub                (default: lub)
    -v|--hv-method    greedy|model|callback (default: greedy)
        --hv_method only makes sense when --alg-version lub
    -h|--help : this message is printed
```

A continuació es mostra l'exemple d'execució amb la instància CP13-404 i les opcions *sat*, *lub* i *greedy*. A cada iteració el solver imprimeix una línia amb informació de l'estat de la resolució.

```
$ ./main ../instances_preproc/CP13-404.wcsp -s -c sat -a lub -v greedy
../instances_preproc/CP13-404.wcsp
Options: sat, lub, greedy
lb 67 ub 154 before adjustment
279 unsorted cost functions
lb 0 ub 87 after adjustment
costs ready
88 variables 679 functions 67 lb, 87 ub
Iteration <i>  (MHV  lb <lb> ub <ub>) | (HV  cores <found> <independent> <new>)
time <csp> <mhv> <hv>
Iteration 0  HV  cores 19 19 19  time 0.00357 0 0
Iteration 1  HV  cores 31 31 12  time 0.005554 0 0
Iteration 2  HV  cores 38 38 7  time 0.007134 0 0
Iteration 3  HV  cores 43 43 5  time 0.008066 0 0
Iteration 4  HV  cores 47 46 4  time 0.008814 0 0
Iteration 5  HV  cores 49 48 2  time 0.009234 0 0
Iteration 6  HV  cores 50 49 1  time 0.009483 0 0
Iteration 7  HV  cores 51 50 1  time 0.009726 0 0
Iteration 8  HV  cores 52 51 1  time 0.009959 0 0
Iteration 9 MHV  lb 91 ub 129  time 0.010077 0.018068 0
Iteration 10  HV  cores 62 61 10  time 0.012004 0.018068 0
Iteration 11  HV  cores 69 68 7  time 0.013271 0.018068 0
Iteration 12  HV  cores 74 73 5  time 0.014388 0.018068 0
Iteration 13  HV  cores 78 76 4  time 0.015668 0.018068 0
Iteration 14  HV  cores 80 78 2  time 0.016387 0.018068 0
```

```
Iteration 15  HV  cores 81 79 1  time 0.016689 0.018068 0
Iteration 16  HV  cores 82 80 1  time 0.016946 0.018068 0
Iteration 17  HV  cores 83 81 1  time 0.017185 0.018068 0
Iteration 18  HV  cores 84 81 1  time 0.017417 0.018068 0
Iteration 19 MHV  lb 94 ub 126  time 0.017526 0.029296 0
Iteration 20  HV  cores 95 92 11  time 0.019079 0.029296 0
Iteration 21  HV  cores 102 99 7  time 0.020256 0.029296 0
Iteration 22  HV  cores 106 102 4  time 0.021046 0.029296 0
Iteration 23  HV  cores 110 106 4  time 0.021819 0.029296 0
Iteration 24  HV  cores 111 107 1  time 0.022106 0.029296 0
Iteration 25  HV  cores 112 108 1  time 0.022405 0.029296 0
Iteration 26  HV  cores 113 109 1  time 0.022774 0.029296 0
Iteration 27  HV  cores 114 110 1  time 0.023071 0.029296 0
Iteration 28 MHV  lb 99 ub 126  time 0.023179 0.042327 0
Iteration 29  HV  cores 122 118 8  time 0.024523 0.042327 0
Iteration 30  HV  cores 126 122 4  time 0.025367 0.042327 0
Iteration 31  HV  cores 129 125 3  time 0.025957 0.042327 0
Iteration 32  HV  cores 130 126 1  time 0.026218 0.042327 0
Iteration 33  HV  cores 131 127 1  time 0.026463 0.042327 0
Iteration 34  HV  cores 132 128 1  time 0.026721 0.042327 0
Iteration 35  HV  cores 133 129 1  time 0.026997 0.042327 0
Iteration 36 MHV  lb 102 ub 122  time 0.027103 0.08115 0
Iteration 37  HV  cores 140 136 7  time 0.028731 0.08115 0
Iteration 38  HV  cores 143 139 3  time 0.029354 0.08115 0
Iteration 39  HV  cores 145 141 2  time 0.030478 0.08115 0
Iteration 40  HV  cores 147 142 2  time 0.032417 0.08115 0
Iteration 41  HV  cores 148 143 1  time 0.032704 0.08115 0
Iteration 42  HV  cores 149 144 1  time 0.032976 0.08115 0
Iteration 43  HV  cores 150 145 1  time 0.033256 0.08115 0
Iteration 44  HV  cores 151 146 1  time 0.033527 0.08115 0
Iteration 45 MHV  lb 104 ub 122  time 0.033631 0.127556 0
Iteration 46  HV  cores 157 151 6  time 0.034474 0.127556 0
Iteration 47  HV  cores 162 156 5  time 0.035409 0.127556 0
Iteration 48  HV  cores 164 158 2  time 0.035887 0.127556 0
Iteration 49  HV  cores 166 160 2  time 0.036272 0.127556 0
Iteration 50  HV  cores 167 161 1  time 0.036526 0.127556 0
```

[...]

```
Iteration 225  HV  cores 478 436 2  time 0.128691 6.89588 0
Iteration 226  HV  cores 479 437 1  time 0.129135 6.89588 0
Iteration 227 MHV  lb 113 ub 114  time 0.129952 7.2688 0
Iteration 228  HV  cores 484 440 5  time 0.131362 7.2688 3.1e-05
Iteration 229  HV  cores 486 442 2  time 0.132021 7.2688 3.1e-05
Iteration 230  HV  cores 487 443 1  time 0.13251 7.2688 3.1e-05
Iteration 231  HV  cores 488 444 1  time 0.13283 7.2688 3.1e-05
Iteration 232  HV  cores 489 445 1  time 0.133111 7.2688 3.1e-05
Iteration 233 MHV  lb 113 ub 114  time 0.133221 7.71302 3.1e-05
Iteration 234  HV  cores 490 446 1  time 0.133558 7.71302 3.1e-05
Iteration 235 MHV  lb 113 ub 114  time 0.133712 8.42995 3.1e-05
Iteration 236  HV  cores 491 447 1  time 0.134039 8.42995 3.1e-05
Iteration 237 MHV  lb 113 ub 114  time 0.134162 9.11024 3.1e-05
Iteration 238  HV  cores 492 448 1  time 0.134581 9.11024 3.1e-05
Iteration 239 MHV  lb 114 ub 114  time 0.134746 9.76526 3.1e-05

optimal solution: (variable values)
1 1 1 1 0 3 3 1 1 1 1 1 1 0 1 3 1 1 1 0 1 1 0 1 1 3 1 0 3 1 1 0 0 1 1 0 1 1 1 0 1 1 1 1
1 1 3 1 1 0 1 1 1 3 3 1 3 1 1 1 1 1 1 1 1 1 0 1 0 0 0 2 1 0 3 1 3 1 1 3 1 1 1 1 1 1 2 3 0

Optimal cost: 114
```

```
Real time: 9.96431
CPU time: 18.178
```

Com que estem en la versió *lub* en algunes iteracions es calcula el MHV i en altres el HV. Veiem que al principi MHV es calcula aproximadament un cop cada deu iteracions i el final es fa amb més freqüència. Observem que l'optimització de trobar més d'un core per iteració ha fet efecte, ja que amb 239 iteracions s'han trobat 498 cores (dels quals 448 són independents entre si). Tot i així el temps total dedicat a MHV (9,77 segons) segueix sent molt més alt al de CoresCSP (0,135 segons). Com que HV es computa de forma *greedy*, el temps dedicat a HV és negligible (31 microsegons). És una mica més significatiu a les versions *model* i *callback*, en què es crida el model.

El temps de CPU és el doble que el temps real. Això es deu a que *cplex* utilitza paral·lelisme.

6 Visualitzador d'instàncies WCSP

El visualitzador proporciona les següents dades numèriques sobre una instància WCSP:

- Nombre de variables
- Mida màxima dels dominis
- Nombre de funcions de cost
- Top cost. És el número amb què es representa l'infinit. També és un upper bound global.
- Aritat màxima de les funcions de cost
- Nombre de funcions de cost no unàries
- Nombre de components connexes del conjunt de variables. Entenem que dos variables són adjecents si són entrades d'una mateixa funció.
- Lower i upper bounds inicials trobats per *toulbar2*.
- Amplada de la descomposició en arbre.
- Alçada de la descomposició en arbre.
- Nombre de clústers (vèrtexs de la descomposició en arbre).

Opcionalment també pot proporcionar les següents dades:

- Nombre de funcions amb cada aritat
- Mida de cada domini
- Nombre de funcions *hard*, *soft* i *mixta*. Una funció *hard* té costos exclusivament 0 i ∞. Una funció *soft* no té costos ∞. Una funció *mixta* té costos ∞ i altres.
- Indicadors estadístics sobre la *undefultness*: mitjana, mediana, mitjana$\pm$2(desviació mitjana). La *undefaultness* d'una funció és la divisió del nombre de tuples amb cost no *default* entre el nombre total de possibles tuples.
- Indicadors estadístics sobre la complexitat: mitjana, mediana, mitjana$\pm$2(desviació mitjana). La complexitat d'una funció és la divisió del nombre de costos diferents dividit entre el nombre total de possibles tuples.

La descomposició en arbre (*tree decomposition*) és un mapeig del graf d'interacció en un arbre. Aquest mapeig el fa *toulbar2*. La figura 27 mostra un exemple de com es descompon un graf en un arbre

6.1 Exemple d'execució

Si l'executem amb l'opció `--help` obtenim una guia.

```
$ python3 view-wcsp.py --help
usage: view-wcsp.py [-h] [--preproc [FILE|DIR]] [--report [PDF|DIR]] [--csv [CSV]]
                    [--domain] [--arity] [--hardness] [--graph [PDF|DIR]]
                    [--tree [PDF|DIR]] [--scatter [PDF|DIR]] [--bar [PDF|DIR]]
                    [--noshow] [--undefaultness] [--complexity]
                    wcsp_file [wcsp_file ...]

positional arguments:
```

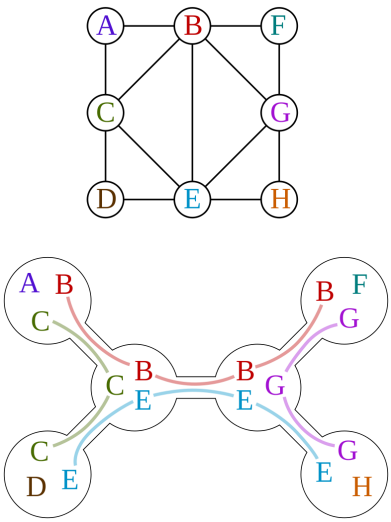

Figura 27: Graf de 8 vèrtexs i la seva descomposició en un arbre de 6 clústers. Cada aresta del graf connecta dos vèrtexs que queden junts en algun clúster de l'arbre.

```
  wcsp_file             input wcsp instance

optional arguments:
  -h, --help            show this help message and exit
  --preproc [FILE|DIR], -p [FILE|DIR]
                        Preprocess instance with toulbar2. Preproessed instance(s) saved
                        as FILE or in DIR/. By default, saved as preproc.wcsp or in
                        preproc/. Remaining info will be computed from preprocessed
                        instance(s).
  --report [PDF|DIR], -r [PDF|DIR]
                        Store info & plots to pdf (one file per instance) Saved as PDF
                        or in DIR/. By default, saved as report.pdf or in report/.
  --csv [CSV], -C [CSV]
                        Store info to csv excel (one file with all instances). Saved as
                        CSV. By default, saved as excel.csv.
  --domain, -d          Get domain sizes of each variable of the instance.
  --arity, -a           Get arity of each function of the instance.
  --hardness, -H        Get hardness of each function of the instance. A function is
                        either hard (costs 0 and top) or soft (costs not top) or mixt
                        (costs include top).
  --graph [PDF|DIR], -g [PDF|DIR]
                        Get interaction graph. If PDF or DIR given, figure(s) are saved
                        there.
  --tree [PDF|DIR], -t [PDF|DIR]
                        Get tree decomposition of interaction graph. If PDF or DIR
                        given, figure(s) are saved there.
  --scatter [PDF|DIR], -s [PDF|DIR]
                        Plot scatter graph showing (tuple) cost vs (function) arity. If
                        PDF or DIR given, figure(s) are saved there.
  --bar [PDF|DIR], -b [PDF|DIR]
                        Plot bar graph showing (function) hardness vs (function) arity.
                        If PDF or DIR given, figure(s) are saved there.
  --noshow, -n          Don't show plots. By default plots are shown (windows emerge) if
                        just one wcsp_file given.
  --undefaultness, -u   Get statistical indexs on the undefaultness of every function in
                        the instance. Let f be a function: undefaultnss(f) =
                        num_tuples_cost_not_default(f) / num_possible_tuples(f)
  --complexity, -c      Get statistical indexs on the complexity of every function in
                        the instance. Let f be a function: complexity(f) =
                        num_distinct_costs(f) / num_possible_tuples(f)
```

Al següent exemple l'executem per obtenir informació numèrica.

```
$ python3 wcsp-viewer.py ../instances_preproc/CP13-404.wcsp --arity --domain
--hardness --undefaultness --complexity

../../instances_preproc/CP13-404.wcsp instance:
 Computing information...
 Done.

 Number of variables: 88
 Maximum domain size: 4
 Number of cost functions: 680
 Top cost (global upper bound): 154
 Maximum arity: 3
 Number of non-unary cost functions: 591
 Number of connected components: 2
 Initial lower and upper bounds: [67, 154[ 56.494%
 Tree decomposition width: 19
 Tree decomposition height: 43
 Number of clusters: 35

 Arity:
 1 functions of arity 0
 88 functions of arity 1
 573 functions of arity 2
 18 functions of arity 3

 Hardness:
 542 hard functions
 49 mixt functions
 89 soft functions

 Domain:
 62 domains of size 2
 1 domains of size 3
 25 domains of size 4

 Undefaultness:
 [let f be a function: undefaultness(f) = num_tuples_cost_not_default(f) / num_possible_tuples(f)]
 mean: 0.3498468137254902
 median: 0.25
 mean-2*stdev: -0.17801208449445916
 mean+2*stdev: 0.8777057119454396

 Complexity:
 [let f be a function: complexity(f) = num_distinct_costs(f) / num_possible_tuples(f)]
 mean: 0.3854166666666667
 median: 0.5
 mean-2*stdev: 0.051241870912039134
 mean+2*stdev: 0.7195914624212942
```

Si executem `python3 view-wcsp.py ../instances_preproc/CP13-404.wcsp --bar --scatter --graph --tree` obtenim els següents gràfics:

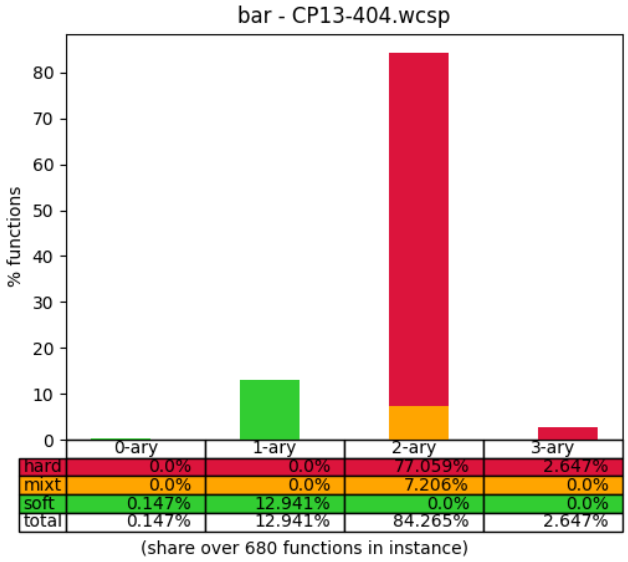

Figura 28: Gràfic de barres de la instància CP13-404.

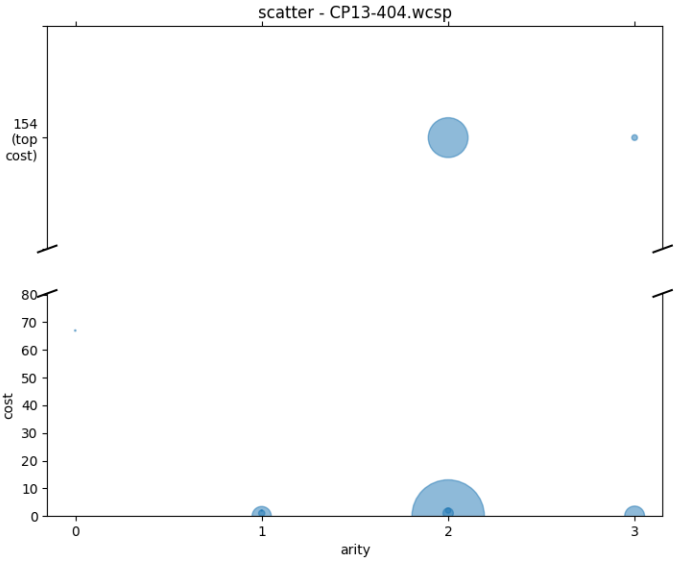

Figura 29: Gràfic de dispersió de la instància CP13-404.

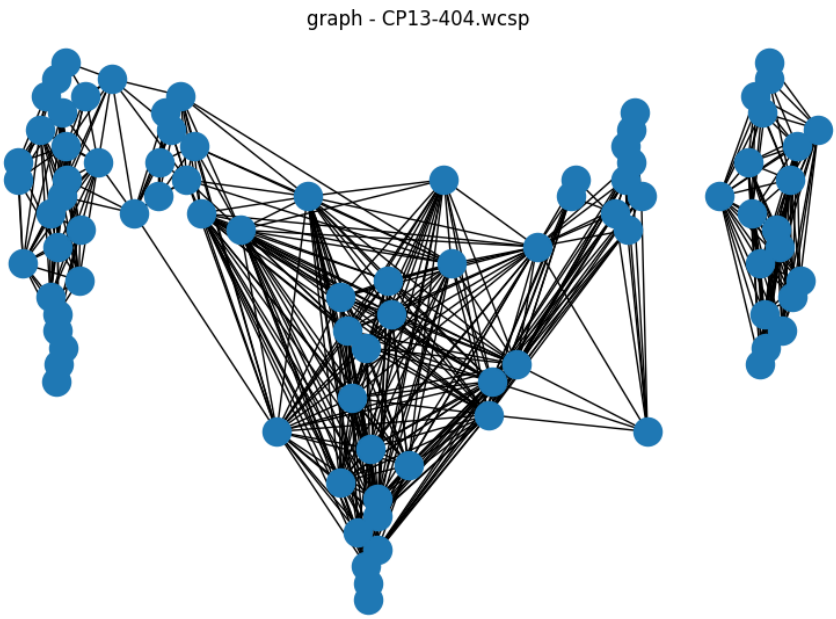

Figura 30: Graf d'interacció de la instància CP13-404.

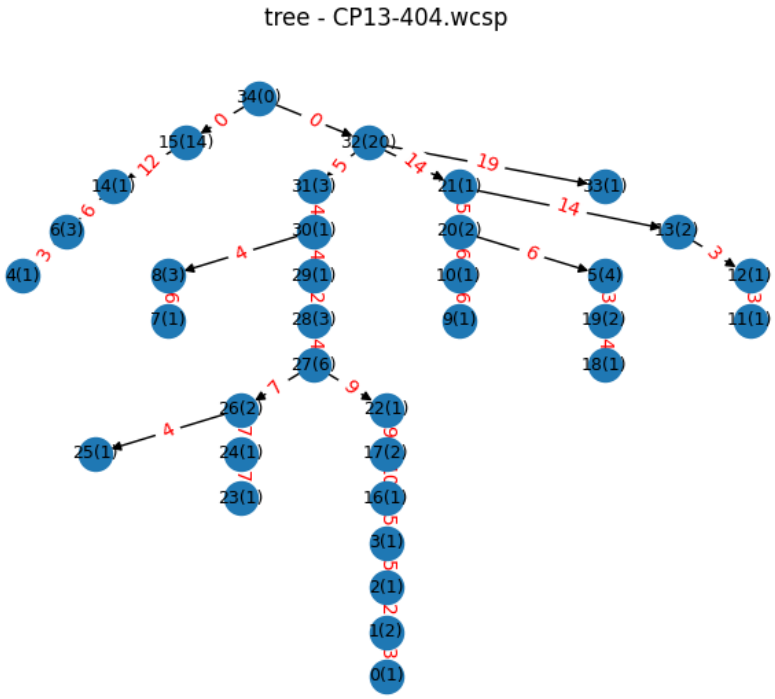

Figura 31: Descomposició en arbre del graf d'interacció de la instància CP13-404.

7 Experiments

L'objectiu dels experiments és avaluar l'eficiència i correctesa dels nostres solvers i comparar-los amb solvers de referència. Per avaluar la correctesa, hem comprovat que el cost de la solució és al mateix en els nostres solvers que en els solvers de referència. En tots els casos és així. Per avaluar l'eficiència ens ham fixat en el temps d'execució, tal com analitzem a continuació.

7.1 Benchmarkcs

L'entrada dels problemes no pot ser qualsevol sinó que ha de representar algun problema útil. Pels experiments hem partit 109 instàncies WCSP extretes de quatre publicacions científiques. Els quatre benchmarks són:

- **CP13**. 16 instàncies. Publicades a [12] (conferència Constraint Programming, 2013). Disponibles a [60, 61].
- **AIJ14**. 47 instàncies. Publicades a [13] (revista Artificial Intelligence Journal, 2014). Disponibles a [62].
- **Bio13**. 25 instàncies. Publicades a [14] (revista Bioinformatics, 2013). Disponibles a [63].
- **UAI17**. 21 instàncies. Publicades a [15] (conferència Uncertainty in Artificial Intelligence, 2017). Disponibles a [64].

Les instàncies de AIJ14, Bio13 i UAI17 són de disseny computacional de proteïnes. En canvi CP13 conté 3 instàncies de gestió de satèl·lits i 13 instàncies d'inferència probabilística pel lligament genètic (*pedigree*). Inicialment vam incloure CP13 (tot i no ser instàncies CPD) pel fet de ser instàncies petites. Més endavant veurem que el nostre solver WCSP ha estat especialment bo en aquest conjunt.

Hem preprocessat les instàncies amb *toulbar2*.

Pels experiments de MaxSAT hem traduït les instàncies de WCSP a MaxSAT mitjançant una variació de l'script [16]. La tècnica per traduir de WCSP a MaxSAT és molt similar a la de traduir de CSP a SAT que hem fet servir a 5.1.1.

Les taules 5 i 6 inclouen la llista d'instàncies i les característiques principals (nombre de variables, mida màxima dels dominis, nombre de funcions i aritat màxima de les funcions). La taula 4 mostra les mitjanes de les característiques per cada conjunt. Com podem observar, els benchmarks de CPD tenen funcions amb aritats no superiors a 2. En canvi CP13 té aritats fins a 4. Les instàncies CP13 tenen un nombre alt de variables i funcions, però la mida reduïda dels dominis fa que siguin relativament fàcils de resoldre. UAI17 és el conjunt amb instàncies més grans i més difícils de resoldre.

Hem conservat el nom original de les instàncies i hi hem afegit un prefix que indica el benchmark del qual provenen. En el cas de AIJ14 hem posat dos prefixos per conservar la divisió original en dos grups.

Benchmark	Nombre d'instàncies	Nombre de variables	Mida màxima dels dominis	Nombre de funcions	Aritat màxima
CP13	16	505,44	4,56	1468,38	3,44
AIJ14	47	35,21	126,79	760,15	2,00
Bio13	25	34,52	131,24	738,64	2,00
UAI17	21	172,00	407,14	3020,00	2,00

Taula 4: Mitjanes de les característiques de les instàncies per cada benchmark.

Nom de la instància	Núm. variables	Mida màx. dominis	Núm. funcions	Aritat màx.
CP13-404.wcsp	88	4	680	3
CP13-503.wcsp	123	4	562	3
CP13-505.wcsp	233	4	2200	3
CP13-pedigree13.wcsp	567	3	1400	3
CP13-pedigree18.wcsp	717	5	1887	4
CP13-pedigree20.wcsp	273	5	726	3
CP13-pedigree25.wcsp	633	5	1626	4
CP13-pedigree30.wcsp	727	5	1913	4
CP13-pedigree31.wcsp	745	5	1821	4
CP13-pedigree33.wcsp	467	4	1327	4
CP13-pedigree39.wcsp	536	5	1339	3
CP13-pedigree41.wcsp	600	5	1665	4
CP13-pedigree44.wcsp	545	4	1511	4
CP13-pedigree51.wcsp	684	4	1814	3
CP13-pedigree7.wcsp	464	4	1179	3
CP13-pedigree9.wcsp	685	7	1844	3
AIJ14F-1C9O.55p.19aa.usingEref_self_digit2.wcsp	55	198	1541	2
AIJ14F-1CM1.17p.19aa.usingEref_self_digit2.wcsp	16	187	137	2
AIJ14F-1HZ5.12p.19aa.usingEref_self_digit2.wcsp	12	168	79	2
AIJ14F-1HZ5.12p.9aa.usingEref_self_digit2.wcsp	12	41	76	2
AIJ14F-1PGB.11p.19aa.usingEref_self_digit2.wcsp	11	180	67	2
AIJ14F-1PGB.11p.9aa.usingEref_self_digit2.wcsp	11	48	62	2
AIJ14F-1PIN.28p.19aa.usingEref_self_digit2.wcsp	28	198	407	2
AIJ14F-1UBI.13p.19aa.usingEref_self_digit2.wcsp	13	177	92	2
AIJ14F-1UBI.13p.9aa.usingEref_self_digit2.wcsp	13	44	92	2
AIJ14F-2DHC.14p.19aa.usingEref_self_digit2.wcsp	14	182	106	2
AIJ14F-2PCY.18p.8aa.usingEref_self_digit2.wcsp	17	43	132	2
AIJ14F-2TRX.11p.8aa.usingEref_self_digit2.wcsp	11	48	67	2
AIJ14L-1BK2.matrix.24p.17aa.usingEref_self_digit2.wcsp	21	84	213	2
AIJ14L-1BRS.matrix.44p.20aa.usingEref_self_digit2.wcsp	38	178	673	2
AIJ14L-1C9O.matrix.43p.17aa.usingEref_self_digit2.wcsp	35	177	523	2
AIJ14L-1CDL.matrix.40p.19aa.usingEref_self_digit2.wcsp	38	170	714	2
AIJ14L-1CM1.matrix.42p.19aa.usingEref_self_digit2.wcsp	42	173	823	2
AIJ14L-1CSE.matrix.97p.18aa.usingEref_self_digit2.wcsp	40	114	380	2
AIJ14L-1CSK.matrix.30p.9aa.usingEref_self_digit2.wcsp	18	41	152	2
AIJ14L-1CSP.matrix.30p.17aa.usingEref_self_digit2.wcsp	14	142	95	2
AIJ14L-1CTF.matrix.39p.9aa.usingEref_self_digit2.wcsp	36	43	571	2
AIJ14L-1DKT.matrix.46p.18aa.usingEref_self_digit2.wcsp	45	166	896	2
AIJ14L-1ENH.matrix.36p.17aa.usingEref_self_digit2.wcsp	36	166	660	2
AIJ14L-1FNA.matrix.38p.8aa.usingEref_self_digit2.wcsp	29	45	350	2
AIJ14L-1FYN.matrix.23p.19aa.usingEref_self_digit2.wcsp	21	183	210	2
AIJ14L-1GVP.matrix.52p.17aa.usingEref_self_digit2.wcsp	52	174	1203	2
AIJ14L-1HNG.matrix.85p.17aa.usingEref_self_digit2.wcsp	66	168	1110	2
AIJ14L-1L63.matrix.83p.17aa.usingEref_self_digit2.wcsp	73	175	1680	2
AIJ14L-1LZ1.matrix.59p.9aa.usingEref_self_digit2.wcsp	53	42	957	2
AIJ14L-1MJC.matrix.28p.17aa.usingEref_self_digit2.wcsp	5	136	16	2
AIJ14L-1NXB.matrix.34p.9aa.usingEref_self_digit2.wcsp	24	35	253	2
AIJ14L-1PGB.matrix.31p.17aa.usingEref_self_digit2.wcsp	31	179	496	2
AIJ14L-1PIN.matrix.28p.20aa.usingEref_self_digit2.wcsp	26	185	320	2
AIJ14L-1POH.matrix.46p.17aa.usingEref_self_digit2.wcsp	31	168	287	2
AIJ14L-1RIS.matrix.56p.17aa.usingEref_self_digit2.wcsp	55	177	1321	2
AIJ14L-1SHF.matrix.30p.9aa.usingEref_self_digit2.wcsp	20	47	193	2
AIJ14L-1SHG.matrix.28p.17aa.usingEref_self_digit2.wcsp	18	54	154	2
AIJ14L-1STN.matrix.120p.18aa.usingEref_self_digit2.wcsp	120	180	7166	2
AIJ14L-1TEN.matrix.39p.9aa.usingEref_self_digit2.wcsp	21	43	160	2
AIJ14L-1UBI.matrix.40p.17aa.usingEref_self_digit2.wcsp	35	141	543	2

Taula 5: Característiques principals de les instàncies (part 1/2).

Nom de la instància	Núm. variables	Mida màx. dominis	Núm. funcions	Aritat màx.
AIJ14L-2CI2.matrix.51p.18aa.usingEref_self_digit2.wcsp	48	180	1078	2
AIJ14L-2DRI.matrix.37p.19aa.usingEref_self_digit2.wcsp	34	179	484	2
AIJ14L-2PCY.matrix.46p.9aa.usingEref_self_digit2.wcsp	31	47	367	2
AIJ14L-2RN2.matrix.69p.9aa.usingEref_self_digit2.wcsp	52	43	1029	2
AIJ14L-2TRX.matrix.61p.19aa.usingEref_self_digit2.wcsp	56	179	990	2
AIJ14L-3CHY.matrix.74p.9aa.usingEref_self_digit2.wcsp	63	56	1506	2
AIJ14L-3HHR.matrix.115p.19aa.usingEref_self_digit2.wcsp	115	175	5296	2
Bio13-1BK2.matrix.24p.17aa.usingEref_self_digit8.wcsp	22	145	254	2
Bio13-1BRS.matrix.44p.20aa.usingEref_self_digit8.wcsp	38	178	742	2
Bio13-1C9O.matrix.43p.17aa.usingEref_self_digit8.wcsp	35	177	631	2
Bio13-1CDL.matrix.40p.19aa.usingEref_self_digit8.wcsp	38	170	742	2
Bio13-1CM1.matrix.42p.19aa.usingEref_self_digit8.wcsp	42	167	904	2
Bio13-1CSE.matrix.97p.18aa.usingEref_self_digit8.wcsp	40	114	821	2
Bio13-1CSK.matrix.30p.9aa.usingEref_self_digit8.wcsp	18	41	172	2
Bio13-1CSP.matrix.30p.17aa.usingEref_self_digit8.wcsp	14	150	106	2
Bio13-1CTF.matrix.39p.9aa.usingEref_self_digit8.wcsp	36	43	667	2
Bio13-1DKT.matrix.46p.18aa.usingEref_self_digit8.wcsp	45	168	1036	2
Bio13-1ENH.matrix.36p.17aa.usingEref_self_digit8.wcsp	36	166	667	2
Bio13-1FNA.matrix.38p.8aa.usingEref_self_digit8.wcsp	29	46	436	2
Bio13-1FYN.matrix.23p.19aa.usingEref_self_digit8.wcsp	21	184	232	2
Bio13-1GVP.matrix.52p.17aa.usingEref_self_digit8.wcsp	52	170	1379	2
Bio13-1HNG.matrix.85p.17aa.usingEref_self_digit8.wcsp	65	168	2139	2
Bio13-1L63.matrix.83p.17aa.usingEref_self_digit8.wcsp	72	175	2628	2
Bio13-1LZ1.matrix.59p.9aa.usingEref_self_digit8.wcsp	53	43	1432	2
Bio13-1MJC.matrix.28p.17aa.usingEref_self_digit8.wcsp	5	136	16	2
Bio13-1NXB.matrix.34p.9aa.usingEref_self_digit8.wcsp	25	35	326	2
Bio13-1PGB.matrix.31p.17aa.usingEref_self_digit8.wcsp	31	179	497	2
Bio13-1PIN.matrix.28p.20aa.usingEref_self_digit8.wcsp	26	185	352	2
Bio13-1POH.matrix.46p.17aa.usingEref_self_digit8.wcsp	29	168	436	2
Bio13-1RIS.matrix.56p.17aa.usingEref_self_digit8.wcsp	54	177	1486	2
Bio13-1SHF.matrix.30p.9aa.usingEref_self_digit8.wcsp	20	47	211	2
Bio13-1SHG.matrix.28p.17aa.usingEref_self_digit8.wcsp	17	49	154	2
UAI17-1dvo-full.wcsp	147	389	2354	2
UAI17-1f00-full.wcsp	269	390	5453	2
UAI17-1is1-full.wcsp	184	431	3507	2
UAI17-1ng2-full.wcsp	167	396	2770	2
UAI17-1xaw-full.wcsp	107	412	1556	2
UAI17-1ytq-full.wcsp	173	401	2996	2
UAI17-1yz7-full.wcsp	168	418	3133	2
UAI17-2af5-full.wcsp	280	403	5012	2
UAI17-2gee-full.wcsp	180	393	3348	2
UAI17-2x8x-full.wcsp	224	407	4261	2
UAI17-3e3v-full.wcsp	151	436	2694	2
UAI17-3lf9-full.wcsp	116	404	1950	2
UAI17-3r8q-full.wcsp	188	395	2945	2
UAI17-3sz7-full.wcsp	147	443	2433	2
UAI17-4bxp-full.wcsp	158	439	2341	2
UAI17-4uos-full.wcsp	185	377	3915	2
UAI17-5dbl-full.wcsp	121	384	1475	2
UAI17-5e0z-full.wcsp	127	393	2023	2
UAI17-5e10-full.wcsp	125	400	2002	2
UAI17-5eqz-full.wcsp	135	433	2348	2
UAI17-5jdd-full.wcsp	260	406	4904	2

Taula 6: Característiques principals de les instàncies (part 2/2).

7.2 Condicions dels experiments

Tots els experiments han estat executats en un ordinador de la universitat dedicat exclusivament a aquesta tasca durant el projecte. Té 4 processadors i 16 GB de RAM. El temps límit de cada execució és d'una hora. Cada execució té disponibles tots els recursos de l'ordinador: les 16 GB de RAM i els 4 processadors. El solver *cplex* utilitza paral·lelisme, per tant, els prototips que incloguin *cplex* inclouran paral·lelisme. En aquests cases comptarem el temps real (temps transcorregut entre l'inici i el final del procés) i no el temps de CPU (suma dels temps gastats per cada processador).

7.3 Experiments amb MaxSAT

7.3.1 Experiment preliminar

L'experiment preliminar de MaxSAT té com a objectiu descartar els prototips clarament ineficients. Hem executat els 8 prototips per les 16 instàncies CP13, que són les més senzilles. Tal com veiem a la taula 7, les implementacions manuals de MHS (*dynamic* i *satatic*) no han sigut capaces de resoldre cap instància MaxSAT, per tant, les descartarem. En altres experiments (no inclosos a la memòria) hem vist que poden resoldre MHS correctament, però pel que es veu en el context de MaxSAT no és prou eficient. Per tant, descartem aquestes versions i ens quedem amb els prototips *lb-hitman*, *lb-cplex*, *lub-hitman* i *lub-cplex*.

	lb-hitman	lb-cplex	lb-static	lb-dynamic	lub-hitman	lub-cplex	lub-static	lub-dynamic
CP13-404	0,987	196,360	TO	TO	0,479	127,684	TO	TO
CP13-503	0,218	26,459	TO	TO	0,052	1,002	TO	TO
CP13-505	TO	TO	TO	TO	TO	TO	TO	TO
CP13-pedigree13	TO	1440,486	TO	TO	26,070	10,963	TO	TO
CP13-pedigree18	TO	140,196	TO	TO	262,505	2,127	TO	TO
CP13-pedigree20	TO	TO	TO	TO	257,422	10,460	TO	TO
CP13-pedigree25	TO	197,282	TO	TO	0,105	0,885	TO	TO
CP13-pedigree30	TO	415,921	TO	TO	409,841	2,111	TO	TO
CP13-pedigree31	TO	TO	TO	TO	TO	1279,405	TO	TO
CP13-pedigree33	TO	TO	TO	TO	TO	5,368	TO	TO
CP13-pedigree39	TO	TO	TO	TO	TO	3,557	TO	TO
CP13-pedigree41	TO	TO	TO	TO	TO	420,920	TO	TO
CP13-pedigree44	TO	TO	TO	TO	TO	62,510	TO	TO
CP13-pedigree51	TO	TO	TO	TO	TO	30,456	TO	TO
CP13-pedigree7	TO	2592,495	TO	TO	0,236	1,520	TO	TO
CP13-pedigree9	TO	TO	TO	TO	TO	23,667	TO	TO
Instàncies resoltes	2	7	0	0	8	15	0	0

Taula 7: Resultats de l'experiment preliminar de MaxSAT. Temps (en segons) de resolució de cada instància per cada prototip. TO = temps esgotat (3600 segons).

7.3.2 Experiment final

Els MaxSAT solvers de referència amb què hem comparat el nostre solver són *MaxHS* [20] i *RC2* [19]. Com hem vist a 2.2.4 hi h moltes MaxSAT solvers competitius. Hem escollit *MaxHS* perquè, igual que nosaltres, utilitza la tècnica d'Implicit Hitting Set [21]. Hem escollit *RC2* perquè, igual que el nostre sovler, està programat en Python. De fet, és un dels pocs solvers en aquest llenguatge, ja que Python no destaca per la seva eficiència.

Les taules 8 i 9 mostren els resultats de l'experiment.

	maxHS	rc2	lb-hitman	lb-cplex	lub-hitman	lub-cplex
CP13-404	0,071	0,051	0,820	196,926	0,455	127,986
CP13-503	0,081	0,009	0,255	26,535	0,050	1,008
CP13-505	741,276	0,167	TO	TO	TO	TO
CP13-pedigree13	0,496	4,583	TO	1439,491	19,976	10,883
CP13-pedigree18	0,582	44,434	TO	140,605	182,331	2,155
CP13-pedigree20	0,201	98,854	TO	TO	226,502	10,473
CP13-pedigree25	0,395	0,039	TO	197,520	0,100	0,908
CP13-pedigree30	1,829	1,360	TO	415,593	381,138	2,110
CP13-pedigree31	22,492	TO	TO	TO	TO	1278,874
CP13-pedigree33	3,853	0,048	TO	TO	TO	5,381
CP13-pedigree39	0,926	1559,917	TO	TO	TO	3,563
CP13-pedigree41	6,346	TO	TO	TO	TO	422,566
CP13-pedigree44	19,242	TO	TO	TO	TO	62,416
CP13-pedigree51	3,389	2881,022	TO	TO	TO	30,410
CP13-pedigree7	0,130	29,021	TO	2593,237	0,210	1,527
CP13-pedigree9	9,292	2617,439	TO	TO	TO	23,670
AIJ14F-1C9O.55p.19aa...	TO	MO	MO	MO	MO	MO
AIJ14F-1CM1.17p.19aa...	3,019	1,952	3,507	19,496	3,450	20,638
AIJ14F-1HZ5.12p.19aa...	TO	128,165	TO	TO	TO	TO
AIJ14F-1HZ5.12p.9aa...	2,399	2,631	TO	TO	266,691	1127,563
AIJ14F-1PGB.11p.19aa...	TO	MO	TO	TO	TO	TO
AIJ14F-1PGB.11p.9aa...	0,166	0,118	0,228	1,744	0,217	2,119
AIJ14F-1PIN.28p.19aa...	TO	MO	TO	TO	TO	TO
AIJ14F-1UBI.13p.19aa...	TO	MO	TO	TO	TO	TO
AIJ14F-1UBI.13p.9aa...	783,344	TO	TO	TO	TO	TO
AIJ14F-2DHC.14p.19aa...	TO	MO	TO	TO	TO	TO
AIJ14F-2PCY.18p.8aa...	7,335	1,891	TO	TO	17,658	1132,413
AIJ14F-2TRX.11p.8aa...	0,645	0,260	0,550	26,198	0,406	6,718
AIJ14L-1BK2.matrix.24p.17aa...	12,423	6,042	TO	TO	478,217	2564,192
AIJ14L-1BRS.matrix.44p.20aa...	TO	MO	TO	TO	TO	TO
AIJ14L-1C9O.matrix.43p.17aa...	560,740	40,188	TO	TO	TO	TO
AIJ14L-1CDL.matrix.40p.19aa...	TO	MO	TO	TO	TO	TO
AIJ14L-1CM1.matrix.42p.19aa...	TO	TO	TO	TO	TO	TO
AIJ14L-1CSE.matrix.97p.18aa...	0,141	0,093	0,226	1,064	0,188	1,119
AIJ14L-1CSK.matrix.30p.9aa...	1,132	0,469	1882,769	TO	1,255	73,357
AIJ14L-1CSP.matrix.30p.17aa...	TO	TO	TO	TO	TO	TO
AIJ14L-1CTF.matrix.39p.9aa...	TO	117,929	TO	TO	TO	TO
AIJ14L-1DKT.matrix.46p.18aa...	TO	104,028	TO	TO	TO	TO
AIJ14L-1ENH.matrix.36p.17aa...	TO	MO	TO	TO	TO	TO
AIJ14L-1FNA.matrix.38p.8aa...	TO	15,113	TO	TO	TO	TO
AIJ14L-1FYN.matrix.23p.19aa...	TO	186,371	TO	TO	TO	TO
AIJ14L-1GVP.matrix.52p.17aa...	TO	MO	TO	TO	TO	TO
AIJ14L-1HNG.matrix.85p.17aa...	TO	524,082	TO	TO	TO	TO
AIJ14L-1L63.matrix.83p.17aa...	217,123	5,653	TO	TO	670,381	TO
AIJ14L-1LZ1.matrix.59p.9aa...	TO	TO	TO	TO	TO	TO
AIJ14L-1MJC.matrix.28p.17aa...	0,060	0,026	0,066	0,355	0,066	0,360
AIJ14L-1NXB.matrix.34p.9aa...	0,053	0,039	0,073	0,513	0,072	0,535
AIJ14L-1PGB.matrix.31p.17aa...	TO	MO	TO	TO	TO	TO
AIJ14L-1PIN.matrix.28p.20aa...	TO	MO	TO	TO	TO	TO
AIJ14L-1POH.matrix.46p.17aa...	1,045	0,141	0,323	10,131	0,310	9,951
AIJ14L-1RIS.matrix.56p.17aa...	TO	MO	TO	TO	TO	TO
AIJ14L-1SHF.matrix.30p.9aa...	0,087	0,083	0,167	0,934	0,158	0,984
AIJ14L-1SHG.matrix.28p.17aa...	18,787	2302,124	TO	TO	TO	TO
AIJ14L-1STN.matrix.120p.18aa...	MO	MO	MO	MO	MO	MO
AIJ14L-1TEN.matrix.39p.9aa...	5,852	0,576	3398,934	TO	15,175	261,315
AIJ14L-1UBI.matrix.40p.17aa...	TO	TO	TO	TO	TO	TO

Taula 8: Resultats de l'experiment de MaxSAT (part 1/2). Temps (en segons) de resolució de cada instància per cada prototip. TO = temps esgotat (3600 segons). MO = memòria esgotada (16 GB).

	maxHS	rc2	lb-hitman	lb-cplex	lub-hitman	lub-cplex
AIJ14L-2CI2.matrix.51p.18aa...	TO	MO	TO	TO	TO	TO
AIJ14L-2DRI.matrix.37p.19aa...	TO	MO	TO	TO	TO	TO
AIJ14L-2PCY.matrix.46p.9aa...	0,410	0,180	0,355	2,120	0,425	2,666
AIJ14L-2RN2.matrix.69p.9aa...	TO	35,582	TO	TO	TO	TO
AIJ14L-2TRX.matrix.61p.19aa...	211,456	11,151	TO	TO	325,114	TO
AIJ14L-3CHY.matrix.74p.9aa...	TO	TO	TO	TO	TO	TO
AIJ14L-3HHR.matrix.115p.19aa...	TO	MO	TO	MO	TO	TO
Bio13-1BK2.matrix.24p.17aa...	146,377	MO	TO	TO	TO	TO
Bio13-1BRS.matrix.44p.20aa...	TO	MO	MO	TO	MO	TO
Bio13-1C9O.matrix.43p.17aa...	1128,965	MO	TO	TO	MO	TO
Bio13-1CDL.matrix.40p.19aa...	TO	MO	MO	TO	TO	TO
Bio13-1CM1.matrix.42p.19aa...	TO	MO	MO	TO	MO	TO
Bio13-1CSE.matrix.97p.18aa...	0,192	0,128	0,311	1,511	0,282	1,619
Bio13-1CSK.matrix.30p.9aa...	0,929	MO	TO	TO	1,284	59,547
Bio13-1CSP.matrix.30p.17aa...	TO	MO	TO	TO	MO	TO
Bio13-1CTF.matrix.39p.9aa...	TO	MO	TO	TO	MO	TO
Bio13-1DKT.matrix.46p.18aa...	TO	MO	TO	TO	MO	TO
Bio13-1ENH.matrix.36p.17aa...	TO	MO	MO	TO	TO	TO
Bio13-1FNA.matrix.38p.8aa...	TO	MO	TO	TO	MO	TO
Bio13-1FYN.matrix.23p.19aa...	TO	MO	MO	TO	MO	TO
Bio13-1GVP.matrix.52p.17aa...	TO	MO	TO	TO	TO	TO
Bio13-1HNG.matrix.85p.17aa...	TO	MO	TO	TO	MO	TO
Bio13-1L63.matrix.83p.17aa...	91,253	MO	TO	TO	MO	TO
Bio13-1LZ1.matrix.59p.9aa...	TO	MO	TO	TO	MO	TO
Bio13-1MJC.matrix.28p.17aa...	0,015	0,027	0,088	0,359	0,066	0,423
Bio13-1NXB.matrix.34p.9aa...	0,026	0,080	0,150	0,870	0,148	0,929
Bio13-1PGB.matrix.31p.17aa...	TO	MO	TO	TO	TO	TO
Bio13-1PIN.matrix.28p.20aa...	TO	MO	MO	TO	MO	TO
Bio13-1POH.matrix.46p.17aa...	0,241	0,124	0,327	1,442	0,304	1,590
Bio13-1RIS.matrix.56p.17aa...	TO	MO	TO	TO	TO	TO
Bio13-1SHF.matrix.30p.9aa...	0,026	0,102	0,206	1,178	0,190	1,223
Bio13-1SHG.matrix.28p.17aa...	19,044	MO	TO	TO	MO	TO
UAI17-1dvo-full	MO	MO	MO	MO	MO	MO
UAI17-1f00-full	MO	MO	MO	MO	MO	MO
UAI17-1is1-full	MO	MO	MO	MO	MO	MO
UAI17-1ng2-full	MO	TO	MO	MO	MO	MO
UAI17-1xaw-full	MO	MO	MO	MO	MO	MO
UAI17-1ytq-full	MO	MO	MO	MO	MO	MO
UAI17-1yz7-full	MO	MO	MO	MO	MO	MO
UAI17-2af5-full	MO	MO	MO	MO	MO	MO
UAI17-2gee-full	MO	MO	MO	MO	MO	MO
UAI17-2x8x-full	MO	MO	MO	MO	MO	MO
UAI17-3e3v-full	MO	MO	MO	MO	MO	MO
UAI17-3lf9-full	MO	MO	MO	MO	MO	MO
UAI17-3r8q-full	MO	MO	MO	MO	MO	MO
UAI17-3sz7-full	MO	MO	MO	MO	MO	MO
UAI17-4bxp-full	MO	MO	MO	MO	MO	MO
UAI17-4uos-full	MO	MO	MO	MO	MO	MO
UAI17-5dbl-full	MO	MO	MO	MO	MO	MO
UAI17-5e0z-full	MO	MO	MO	MO	MO	MO
UAI17-5e10-full	MO	MO	MO	MO	MO	MO
UAI17-5eqz-full	MO	TO	MO	MO	MO	MO
UAI17-5jdd-full	MO	MO	MO	MO	MO	MO
Instàncies resoltes	45	43	18	21	30	35

Taula 9: Resultats de l'experiment de MaxSAT (part 2/2). Temps (en segons) de resolució de cada instància per cada prototip. TO = temps esgotat (3600 segons). MO = memòria esgotada (16 GB).

La figura 32 ens permet comparar l'eficiència dels 4 prototips amb els 2 solvers de referència. Veiem que el prototip més eficient del nostre solver és *lub-cplex*, que ha resolt 35 instàncies en menys d'una hora. Tot així, no supera als solvers *MaxHS* i *RC2* que han resolt 45 i 43 instàncies, respectivament.

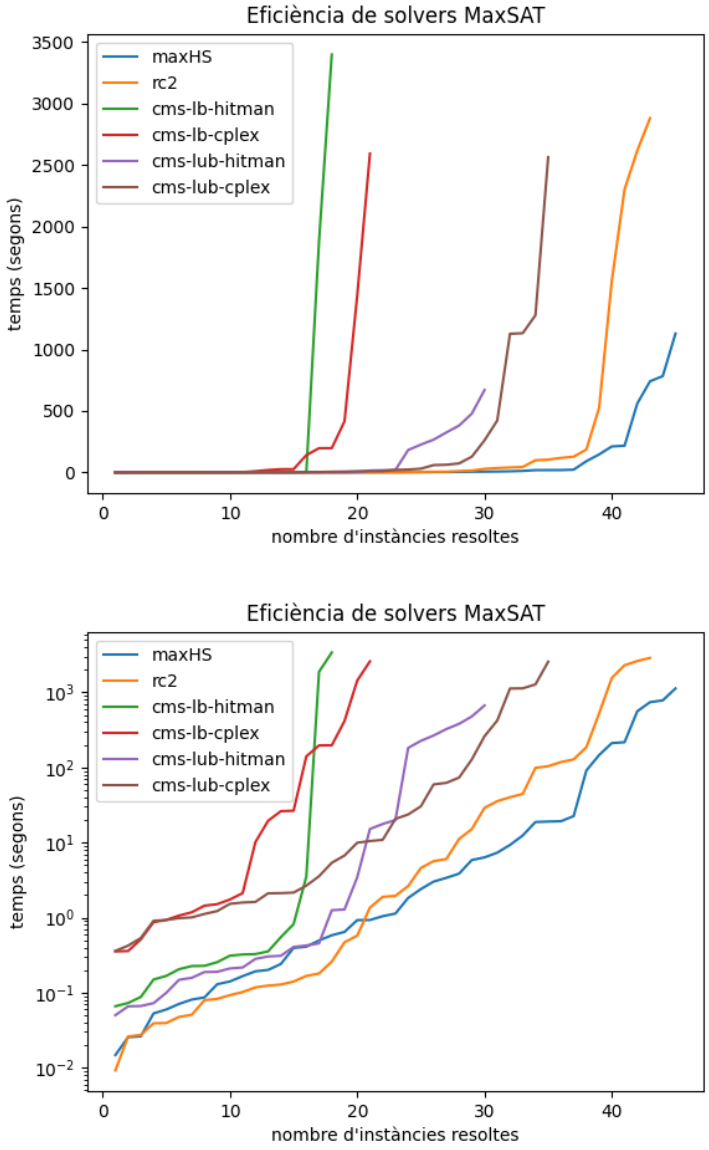

Figura 32: Resultats de l'experiment de MaxSAT. El nostre solver és *cms* (CoreMaxSAT). A dalt, temps en escala lineal; a baix, temps en escala logarítmica.

7.4 Experiments amb WCSP

7.4.1 Experiment preliminar

L'experiment preliminar de WCSP té com a objectiu descartar els prototips clarament ineficients. Hem executat els 8 prototips per les 16 instàncies CP13, que són les més senzilles. Tal com veiem a la taula 10, la implementació manuals de CSP amb Forward Checking (*fc*) no ha sigut capaç de resoldre cap instància WCSP, per tant, la descartem. Ens quedem, doncs amb la implementació de CSP com a SAT, que inclou els prototips *sat-lb*, *sat-lub-greedy*, *sat-lub-model* i *sat-lub-callback*.

	sat-lb	sat-lub-greedy	sat-lub-model	sat-lub-callback	fc-lb	fc-lub-greedy	fc-lub-model	fc-lub-callback
CP13-404	6,834	3,495	5,755	3,784	TO	TO	TO	TO
CP13-503	0,923	0,438	0,864	0,473	TO	TO	TO	TO
CP13-505	TO	899,681	312,010	259,663	TO	TO	TO	TO
CP13-pedigree13	10,119	2,432	9,035	4,986	TO	TO	TO	TO
CP13-pedigree18	2,361	0,597	18,424	9,426	TO	TO	TO	TO
CP13-pedigree20	32,837	0,813	3,615	12,547	TO	TO	TO	TO
CP13-pedigree25	1,962	0,178	5,057	1,961	TO	TO	TO	TO
CP13-pedigree30	5,383	0,415	55,105	12,807	TO	TO	TO	TO
CP13-pedigree31	TO	339,025	2014,490	565,319	TO	TO	TO	TO
CP13-pedigree33	38,377	0,129	23,109	30,869	TO	TO	TO	TO
CP13-pedigree39	18,216	1,359	34,712	13,562	TO	TO	TO	TO
CP13-pedigree41	TO	36,511	TO	1626,870	TO	TO	TO	TO
CP13-pedigree44	647,593	13,020	656,296	252,333	TO	TO	TO	TO
CP13-pedigree51	13,169	2,099	151,306	13,806	TO	TO	TO	TO
CP13-pedigree7	0,982	0,181	1,984	0,851	TO	TO	TO	TO
CP13-pedigree9	412,308	22,353	731,763	172,704	TO	TO	TO	TO
Instàncies resoltes	13	16	15	16	0	0	0	0

Taula 10: Resultats de l'experiment preliminar de WCSP. Temps (en segons) de resolució de cada instància per cada prototip. TO = temps esgotat (3600 segons).

7.4.2 Experiment final

El solver de referència amb què compararem els prototips, és *toulbar2* que, com hem vis és pràcticament l'únic solver de WCSP.

Les taules 11 i 12 mostren els resultats de l'experiment final, que és l'execució de *toulbar2* i els 4 prototips seleccionats a l'experiment preliminar amb les 109 instàncies.

	toulbar2	sat-lb	sat-lub-greedy	sat-lub-model	sat-lub-callback
CP13-404	53,889	6,834	3,495	5,755	3,784
CP13-503	2994,918	0,923	0,438	0,864	0,473
CP13-505	TO	TO	899,681	312,010	259,663
CP13-pedigree13	1,170	10,119	2,432	9,035	4,986
CP13-pedigree18	124,167	2,361	0,597	18,424	9,426
CP13-pedigree20	0,477	32,837	0,813	3,615	12,547
CP13-pedigree25	1262,337	1,962	0,178	5,057	1,961
CP13-pedigree30	187,498	5,383	0,415	55,105	12,807
CP13-pedigree31	2525,973	TO	339,025	2014,490	565,319
CP13-pedigree33	7,646	38,377	0,129	23,109	30,869
CP13-pedigree39	8,486	18,216	1,359	34,712	13,562
CP13-pedigree41	187,981	TO	36,511	TO	1626,870
CP13-pedigree44	2578,925	647,593	13,020	656,296	252,333
CP13-pedigree51	TO	13,169	2,099	151,306	13,806
CP13-pedigree7	6,952	0,982	0,181	1,984	0,851
CP13-pedigree9	3379,366	412,308	22,353	731,763	172,704
AIJ14F-1C9O.55p.19aa...	TO	TO	TO	TO	TO
AIJ14F-1CM1.17p.19aa...	0,261	0,405	0,397	0,399	0,399
AIJ14F-1HZ5.12p.19aa...	0,318	TO	TO	TO	TO
AIJ14F-1HZ5.12p.9aa...	0,015	TO	46,274	TO	TO
AIJ14F-1PGB.11p.19aa...	1,821	TO	TO	TO	TO
AIJ14F-1PGB.11p.9aa...	0,017	0,088	0,085	0,071	0,069
AIJ14F-1PIN.28p.19aa...	TO	TO	TO	TO	TO
AIJ14F-1UBI.13p.19aa...	TO	TO	TO	TO	TO
AIJ14F-1UBI.13p.9aa...	0,076	TO	TO	TO	TO
AIJ14F-2DHC.14p.19aa...	22,814	TO	TO	TO	TO
AIJ14F-2PCY.18p.8aa...	0,028	TO	78,913	TO	TO
AIJ14F-2TRX.11p.8aa...	0,016	3,298	2,185	0,764	837,751
AIJ14L-1BK2.matrix.24p.17aa...	0,028	TO	603,227	TO	TO
AIJ14L-1BRS.matrix.44p.20aa...	742,833	TO	TO	TO	TO
AIJ14L-1C9O.matrix.43p.17aa...	0,324	TO	1540,550	TO	TO
AIJ14L-1CDL.matrix.40p.19aa...	136,020	TO	TO	TO	TO
AIJ14L-1CM1.matrix.42p.19aa...	1,253	TO	TO	TO	TO
AIJ14L-1CSE.matrix.97p.18aa...	0,019	0,039	0,024	0,024	0,024
AIJ14L-1CSK.matrix.30p.9aa...	0,014	666,978	53,749	TO	TO
AIJ14L-1CSP.matrix.30p.17aa...	0,220	TO	TO	TO	TO
AIJ14L-1CTF.matrix.39p.9aa...	0,234	TO	TO	TO	TO
AIJ14L-1DKT.matrix.46p.18aa...	0,396	TO	TO	TO	TO
AIJ14L-1ENH.matrix.36p.17aa...	TO	TO	TO	TO	TO
AIJ14L-1FNA.matrix.38p.8aa...	0,065	TO	TO	TO	TO
AIJ14L-1FYN.matrix.23p.19aa...	0,560	TO	2068,140	TO	TO
AIJ14L-1GVP.matrix.52p.17aa...	2028,209	TO	TO	TO	TO
AIJ14L-1HNG.matrix.85p.17aa...	0,679	TO	TO	TO	TO
AIJ14L-1L63.matrix.83p.17aa...	0,201	TO	774,798	TO	TO
AIJ14L-1LZ1.matrix.59p.9aa...	0,570	TO	TO	TO	TO
AIJ14L-1MJC.matrix.28p.17aa...	0,004	0,027	0,011	0,011	0,011
AIJ14L-1NXB.matrix.34p.9aa...	0,011	0,014	0,015	0,015	0,015
AIJ14L-1PGB.matrix.31p.17aa...	TO	TO	TO	TO	TO
AIJ14L-1PIN.matrix.28p.20aa...	0,968	TO	TO	TO	TO
AIJ14L-1POH.matrix.46p.17aa...	0,027	1,302	0,718	9,924	1,149
AIJ14L-1RIS.matrix.56p.17aa...	110,358	TO	TO	TO	TO
AIJ14L-1SHF.matrix.30p.9aa...	0,017	0,045	0,028	0,027	0,028
AIJ14L-1SHG.matrix.28p.17aa...	0,038	TO	TO	TO	TO
AIJ14L-1STN.matrix.120p.18aa...	TO	TO	TO	TO	TO
AIJ14L-1TEN.matrix.39p.9aa...	0,015	TO	20,236	TO	TO
AIJ14L-1UBI.matrix.40p.17aa...	1,559	TO	TO	TO	TO

Taula 11: Resultats de l'experiment de WCSP (part 1/2). Temps (en segons) de resolució de cada instància per cada prototip. TO = temps esgotat (3600 segons). MO = memòria esgotada (16 GB).

	toulbar2	sat-lb	sat-lub-greedy	sat-lub-model	sat-lub-callback
AIJ14L-2CI2.matrix.51p.18aa...	TO	TO	TO	TO	TO
AIJ14L-2DRI.matrix.37p.19aa...	272,372	TO	TO	TO	TO
AIJ14L-2PCY.matrix.46p.9aa...	0,035	0,166	0,208	0,170	0,148
AIJ14L-2RN2.matrix.69p.9aa...	0,134	TO	TO	TO	TO
AIJ14L-2TRX.matrix.61p.19aa...	0,190	TO	1297,410	TO	TO
AIJ14L-3CHY.matrix.74p.9aa...	62,652	TO	TO	TO	TO
AIJ14L-3HHR.matrix.115p.19aa...	TO	TO	TO	TO	TO
Bio13-1BK2.matrix.24p.17aa...	0,054	TO	TO	TO	TO
Bio13-1BRS.matrix.44p.20aa...	1955,844	TO	TO	TO	TO
Bio13-1C9O.matrix.43p.17aa...	0,288	TO	TO	TO	TO
Bio13-1CDL.matrix.40p.19aa...	260,247	TO	TO	TO	TO
Bio13-1CM1.matrix.42p.19aa...	2,320	TO	TO	TO	TO
Bio13-1CSE.matrix.97p.18aa...	0,023	0,193	0,197	0,196	0,200
Bio13-1CSK.matrix.30p.9aa...	0,013	TO	116,998	TO	TO
Bio13-1CSP.matrix.30p.17aa...	0,275	TO	TO	TO	TO
Bio13-1CTF.matrix.39p.9aa...	0,482	TO	TO	TO	TO
Bio13-1DKT.matrix.46p.18aa...	0,550	TO	TO	TO	TO
Bio13-1ENH.matrix.36p.17aa...	TO	TO	TO	TO	TO
Bio13-1FNA.matrix.38p.8aa...	0,065	TO	TO	TO	TO
Bio13-1FYN.matrix.23p.19aa...	0,851	TO	TO	TO	TO
Bio13-1GVP.matrix.52p.17aa...	TO	TO	TO	TO	TO
Bio13-1HNG.matrix.85p.17aa...	0,786	TO	TO	TO	TO
Bio13-1L63.matrix.83p.17aa...	0,227	TO	TO	TO	TO
Bio13-1LZ1.matrix.59p.9aa...	1,265	TO	TO	TO	TO
Bio13-1MJC.matrix.28p.17aa...	0,004	0,052	0,054	0,053	0,052
Bio13-1NXB.matrix.34p.9aa...	0,012	0,067	0,072	0,072	0,071
Bio13-1PGB.matrix.31p.17aa...	TO	TO	TO	TO	TO
Bio13-1PIN.matrix.28p.20aa...	1,384	TO	TO	TO	TO
Bio13-1POH.matrix.46p.17aa...	0,025	0,288	0,297	0,294	0,296
Bio13-1RIS.matrix.56p.17aa...	95,812	TO	TO	TO	TO
Bio13-1SHF.matrix.30p.9aa...	0,015	0,108	0,114	0,116	0,114
Bio13-1SHG.matrix.28p.17aa...	0,045	TO	TO	TO	TO
UAI17-1dvo-full	969,746	TO	TO	TO	TO
UAI17-1f00-full	2694,083	TO	TO	TO	TO
UAI17-1is1-full	TO	TO	TO	TO	TO
UAI17-1ng2-full	TO	TO	TO	TO	TO
UAI17-1xaw-full	597,994	TO	TO	TO	TO
UAI17-1ytq-full	TO	TO	TO	TO	TO
UAI17-1yz7-full	TO	TO	TO	TO	TO
UAI17-2af5-full	TO	TO	TO	TO	TO
UAI17-2gee-full	281,364	TO	TO	TO	TO
UAI17-2x8x-full	TO	TO	TO	TO	TO
UAI17-3e3v-full	TO	TO	TO	TO	TO
UAI17-3lf9-full	310,493	TO	TO	TO	TO
UAI17-3r8q-full	965,012	TO	TO	TO	TO
UAI17-3sz7-full	TO	TO	TO	TO	TO
UAI17-4bxp-full	330,398	TO	TO	TO	TO
UAI17-4uos-full	TO	TO	TO	TO	TO
UAI17-5dbl-full	63,592	TO	TO	TO	TO
UAI17-5e0z-full	34,892	TO	TO	TO	TO
UAI17-5e10-full	39,073	TO	TO	TO	TO
UAI17-5eqz-full	TO	TO	TO	TO	TO
UAI17-5jdd-full	TO	TO	TO	TO	TO
Instàncies resoltes	85	28	40	29	30

Taula 12: Resultats de l'experiment de WCSP (part 2/2). Temps (en segons) de resolució de cada instància per cada prototip. TO = temps esgotat (3600 segons). MO = memòria esgotada (16 GB).

La figura 33 ens permet comparar l'eficiència dels 4 prototips amb els 2 solvers de referència. Veiem que el prototip més eficient del nostre solver és *sat-lub-greedy*, que ha resolt 40 instàncies en menys d'una hora. Tot així, no supera *toulbar* que ha resolt 85 instàncies, més del doble. Per tant, *toulbar2* és clarament més eficient que el nostre solver pel global d'instàncies. La situació canvia si ens fixem en un subconjunt d'instàncies.

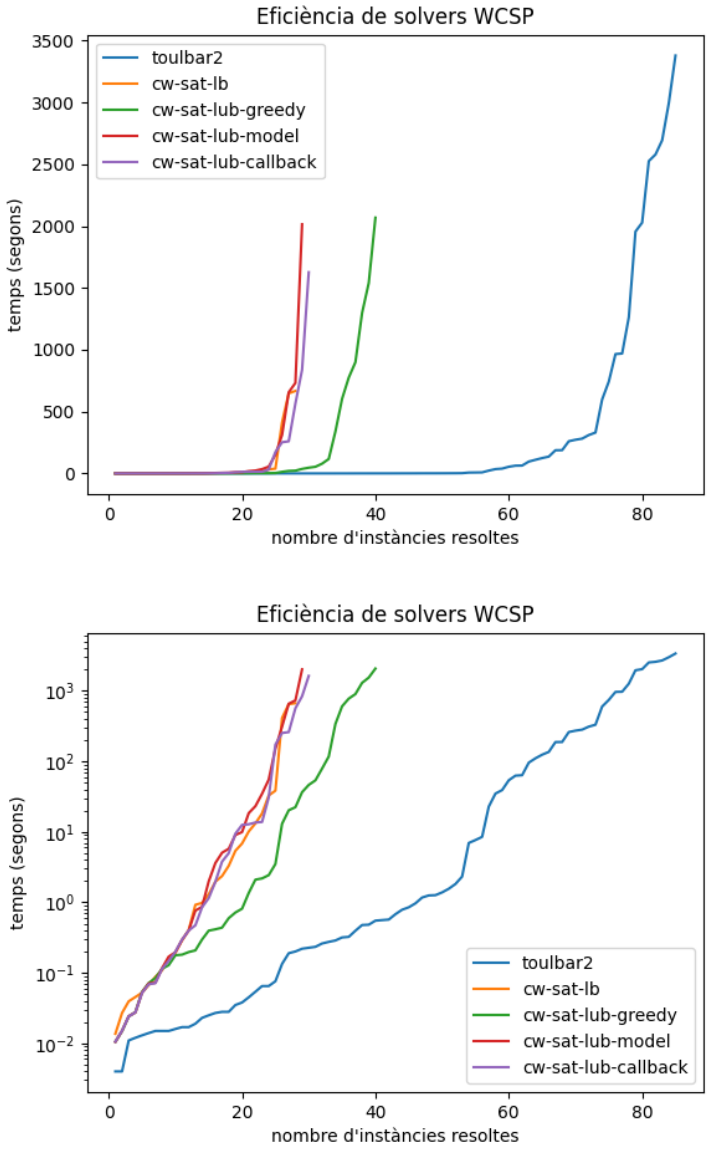

Figura 33: Resultats de l'experiment de WCSP. El nostre solver és *cw* (CoreWCSP). A dalt, temps en escala lineal; a baix, temps en escala logarítmica.

La figura 34 mostra els resultats de l'experiment per les 16 instàncies del benchmark CP13. Són precisament les que no representen disseny de proteïnes, sinó gestió de satèl·lits i lligament genètic. També són les més petites, a causa que els dominis són molt petits. Per aquestes instàncies els 4 prototips del nostre solver són més eficients que *toulbar2*. El prototip més eficient és *sat-lub-greedy*, igual que en el cas anterior. Per posar un exemple, *cw-sat-lub-greedy* pot resoldre 14 instàncies en menys d'un minut i *toulbar2* només 6.

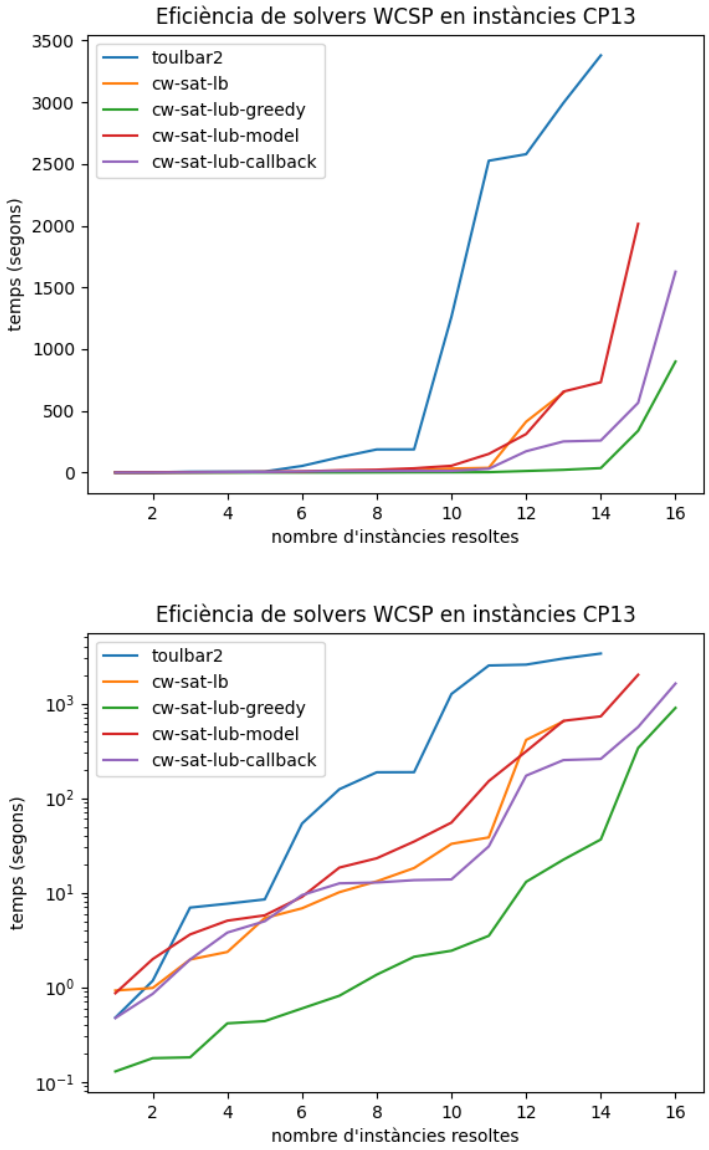

Figura 34: Resultats de l'experiment de WCSP pel benchmark CP13. El nostre solver és *cw* (CoreWCSP). A dalt, temps en escala lineal; a baix, temps en escala logarítmica.

7.4.3 Subproblemes

Tal com hem explicat a 5.1.3 hem programat que a cada iteració es resolgui CSP diverses vegades per tal d'obtenir millors cores. Ho hem justificat al·legant que el solver dedica més temps a MHV i HV que a CSP. A la taula 13 veiem que, tot i aquesta millora, se segueix dedicant més temps a MHV i HV que a CSP.

	CSP	MHV	HV
sat-lb	0,76	102,68	-
sat-lub-greedy	79,95	117,98	-
sat-lub-model	0,44	14,37	124,06
sat-lub-callback	0,89	1,75	124,60

Taula 13: Mitjana de temps (en segons) dedicat a cada subproblema (CSP, MHV i HV) per cada prototip de WCSP. Només es tenen en compte les instàncies resoltes.

7.5 MaxSAT vs WCSP

La figura 35 mostra els resultats dels 3 solvers de referència i el nostre millor prototip per cada problema. Veiem que els nostres prototips no aconsegueixen superar cap solver. Pel que fa els solvers de referència, *toulbar2*, de WCSP, és molt més eficient que els de MaxSAT. Això reforça la nostra hipòtesi inicial que no és bona idea resoldre CPD amb MaxSAT perquè la traducció des de WCSP resulta amb moltes variables a causa de la mida dels dominis.

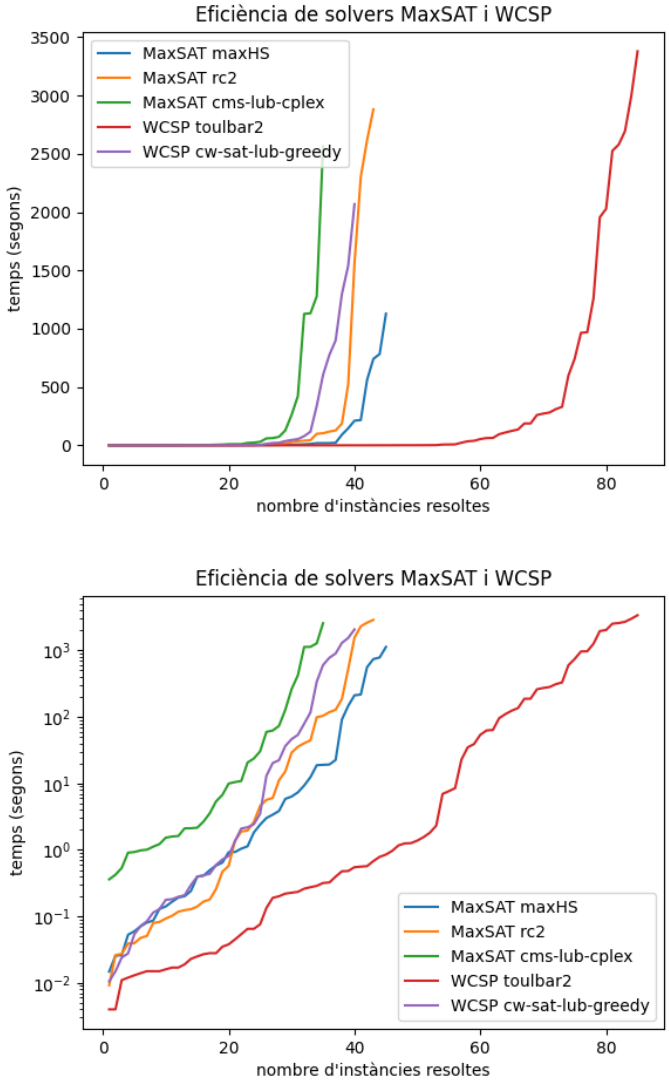

Figura 35: Resultats dels experiments per WCSP i MaxSAT. A dalt, temps en escala lineal; a baix, temps en escala logarítmica.

Si ens centrem en les instàncies CP13, les diferències d'eficiència entre WCSP i MaxSAT s'esvaeixen. Els dos solvers més eficients són *MaxHS* i el nostre millor prototip de WCSP, *cw-sat-lub-greedy*, que queden bastant empatats, quedant *MaxHS* lleugerament per davant.

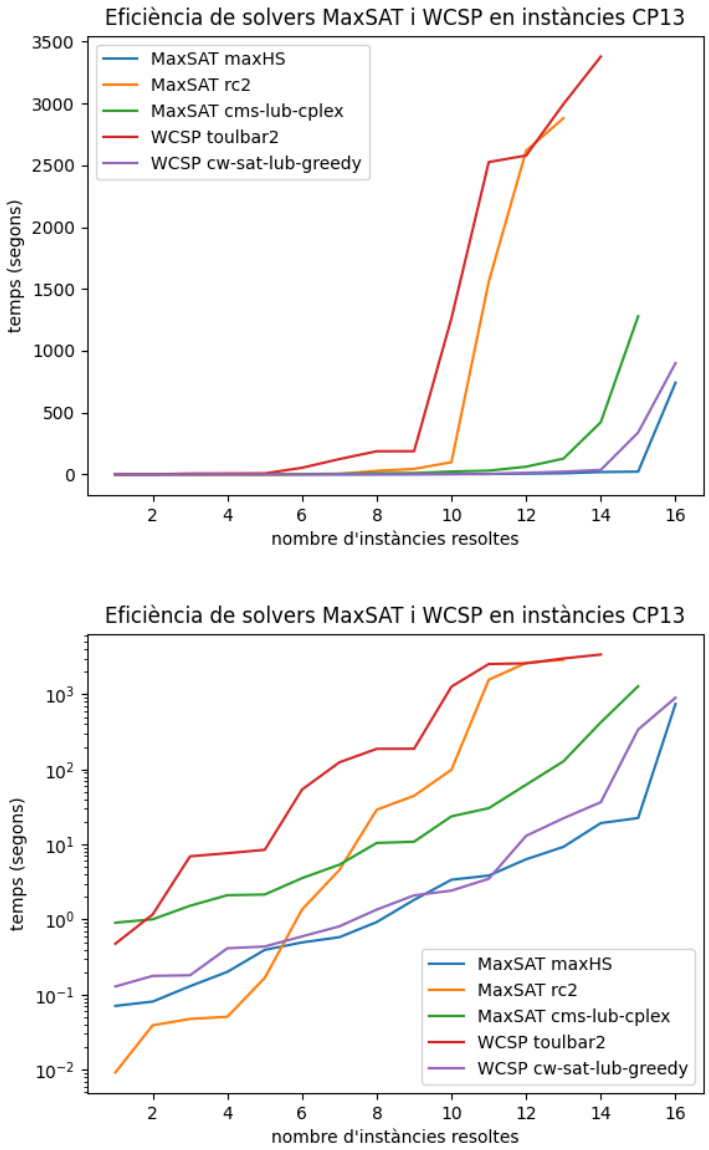

Figura 36: Resultats dels experiments per WCSP i MaxSAT pel benchmark CP13. A dalt, temps en escala lineal; a baix, temps en escala logarítmica.

8 Conclusions

Quan vam començar el projecte crèiem que la nostra implementació de WCSP seria eficient per resoldre disseny computacional de proteïnes (CPD), en comparació al solver de referència *toulbar2*. Curiosament, de moment no hem millorat l'eficiència de CPD però sí d'altres problemes representables a WCSP: gestió de satèl·lits i inferència probabilística pel lligament genètic. A diferència de CPD, aquests problemes solen tenir dominis petits.

Segurament caldria experimentar més amb instàncies d'aquest tipus ja que en aquest projecte només ho hem fet amb 3 de satèl·lits i 13 de lligament genètic. També ho podríem provar amb altres aplicacions de WCSP com l'assignació de radiofreqüències. Amb les proves fetes fins ara l'eficiència del nostre WCSP solver queda empatada amb la del MaxSAT solver *MaxHS*.

A part d'això, el solver encara té marge de millora. Se'ns han acudit altres optimitzacions (com ara agrupar funcions per reduir la mida dels vectors a hitejar) que poden fer millorar l'eficiència i potser eventualment superar *toulbar2* en CPD. De fet, no és estrany que fins ara no l'haguem superat ja que porta vint anys desenvolupant-se i nosaltres portem menys d'un any.

Pel que fa el solver de MaxSAT, no és gaire eficient. La idea d'Implicit Hitting Set ja s'havia posat en pràctica al solver *MaxHS* i el nostre no l'ha superat en cap aspecte. Per això i perquè ja hi ha molts solvers de MaxSAT el més sensat és descartar-lo.

Per altra banda, comparant les diferents versions pels subproblemes, hem constatat que la programació amb restriccions sol ser més eficient que un algoritme específic. Ha estat el cas de MHS, on la reducció a programació lineal i resolució amb *cplex* ha estat molt més eficient que un algoritme recursiu específic. També ha passat amb CSP, que ha estat molt més eficient reduir-lo a SAT i resoldre'l amb *CaDiCaL* que no implementat l'algoritme de Forward Checking.

En resum, el nostre solver WCSP és eficient per un tipus característic d'instàncies, cosa que s'hauria de validar amb un benchmark més gran. Per CPD, el solver encara necessita millores.

A Annex: Planificació

En aquest annex veurem la planificació del treball feta el setembre de 2021, seguint les directrius del curs de Gestió de Projectes (GEP) vinculat al TFG. Noteu que quan es va escriure la planificació, estava previst presentar el TFG a la lectura del gener de 2022, tot i que finalment s'ha optat per presentar-lo a la lectura del juny de 2022.

A.1 Extensió temporal

Aquest treball va iniciar-se a mitjans de juny de 2021. Prendrem com a data d'inici el 15 de juny de 2021.

La lectura està prevista per la setmana del 24 al 29 de gener de 2022. D'acord amb la normativa [65], la memòria s'ha d'entregar almenys una setmana abans de la lectura. Per tant, haurà d'estar acabada el 17 de gener de 2022. Fixarem com a data final el 31 de desembre de 2021 i així deixem 17 dies de marge per possibles imprevistos.

Per tant, l'extensió temporal prevista és de 6 mesos i mig. Preveiem que l'autor hi dedicarà unes 500 hores.

A.2 Personal i material

El personal és:

- *Director del treball.* S'encarrega de dissenyar els algoritmes i supervisar el programador.
- *Programador.* S'encarrega d'implementar algoritmes, optimitzar-los, realitzar experiments i escriure la memòria.
- *Col·laboradora.* Juntament amb el director, dissenya els algoritmes i ajuda al programador.

El material necessari és:

- Un ordinador personal per cada treballador.
- Un espai de treball amb connexió a internet i material d'oficina per cada treballador.
- Un servidor per realitzar experiments.

A.3 Tasques

Podem dividir les tasques en les fases següents:

- Fase prèvia (P)
- Implementació (I)
- Optimització (O)
- Experimentació (E)
- Anàlisi de resultats (A)
- Reunions de seguiment (R)
- Documentació (D)

- Presentació (PR)

A continuació es detallen les tasques de cada fase.

Fase Prèvia (P)

- P1. Dissenyar els algoritmes en pseudo-codi. Aquesta tasca la realitza el director i genera una documentació, on també defineix els problemes.
- P2. Obtenir instàncies WCSP pels futurs experiments. Aquesta tasca la fa el director, que té experiència prèvia en aquest problema. Les instàncies les obté de publicacions científiques.
- P3. Transformar les instàncies WCSP a MaxSAT a través d'un algoritme existent.
- P4. Comprendre els problemes i els algoritmes que el director ha proposat per resoldre'ls. Aquesta tasca la realitza el programador mitjançant la lectura de la documentació i reunions amb el director.
- P5. Descriure les instàncies. Programar i executar un script que analitzi les instàncies i en proporcioni característiques.

Implementació (I)

En aquesta fase implementarem els algoritmes que resolen els problemes i els subproblemes. Cada algoritmes té variacions. En aquesta fase implementarem totes les variacions i en una fase posterior (optimització) determinarem quines variacions son les idònies per cada implementació. Cal tenir en compte que un problema hereta totes les variacions dels seus subproblemes.

- I1. Implementar un algoritme que resolgui Minimum Hitting Set (MHS). Ho farem en Python.
- I2. Implementar un algoritme que resolgui MaxSAT com a Implicit MHS. Ho farem en Python. Una de les variacions serà si utilitzar la nostra implementació de MHS o la que ens proporciona la llibreria PySAT.
- I3. Implementar un algoritme que resolgui Constraint Satisfaction Problem (CSP). Ho farem en C++.
- I4. Implementar un algoritme que resolgui Minimum Hitting Vector (MHV). Ho farem en C++.
- I5. Implementar un algoritme que resolgui Weighted Constraint Satisfaction Problem (WCSP) com a Implicit MHV. Ho farem en C++. CSP i MHV són subproblemes d'aquest; n'utilitzarem les nostres implementacions.

Optimització (O)

Aquesta fase consisteix en triar les variacions òptimes de les implementacions de la fase anterior. Volem que les variacions dels subproblemes (MHS, CSP, MHV) siguin òptimes per resoldre els problemes (MaxSAT, WCSP). Per això, no optimitzarem els subproblemes per si sols, sinó que experimentant amb cert problema triarem les variacions òptimes pel problema i els seus subproblema.

- O1. Optimitzar MaxSAT (i el seu subproblema MHS).
- O2. Optimitzar WCSP (i els seus subproblemes CSP i MHV).

Experimentació (E)

Cada experiment consisteix en què el servidor executa un solver amb cadascuna de les instàncies i anotarem el temps d'execució, així com altres indicadors. Com que no sabem quan trigarà cada execució (podria trigar dies o fins i tot anys), establirem un temps límit i si no es resol en aquest temps avortarem i anotarem que s'ha esgotat el temps.

- E1. Experimentar amb un solver de MaxSAT existent.
- E2. Experimentar amb el solver de MaxSAT que hem implementat.
- E3. Experimentar amb el solver de WCSP que té *toulbar2*.
- E4. Experimentar amb el solver de WCSP que hem implementat.

Anàlisi de resultats (A)

Compararem la nostra implementació de cada algoritme amb la de *toulbar2* a partir dels experiments E1 i E2. Determinarem per quines instàncies una implementació ha superat l'altra en temps d'execució.

- A1. Anàlisi de MaxSAT.
- A2. Anàlisi de WCSP.

Reunions de seguiment (R)

- R1. Reunions de plantejament. En aquestes reunions el director planteja els problemes i explica al programador el algoritmes que ha dissenyat per resoldre'ls. Aquestes reunions ajudaran al programador a realitzar la tasca P4 (comprensió).
- R2. Reunions de seguiment. En aquestes reunions el programador i el director posen en comú el progrés i els resultats obtinguts fins al moment. El director dona indicacions al programador per realitzar les futures tasques i resoldre possibles incidències. A vegades també assisteix la col·laboradora.
- R3. Reunió final. Abans de la data d'entrega el director i el programador es reuneixen per analitzar les conclusions i avaluar l'assoliment dels objectius.

Documentació (D)

- D1. Documentació de la gestió del projecte. Documentarem els aspectes relacionats amb la gestió: context, estat de l'art, abast, metodologia, planificació temporal, planificació econòmica, etc.
- D2. Memòria. Documentarem els aspectes relacionats amb el contingut del treball: implementació, experiments, resultats, etc.

Presentació (PR)

- PR. Presentació. Preparació i execució de la defensa oral del treball davant del tribunal.

A.4 Estimacions temporals i diagrama de Gantt

A la taula 14 es mostra un resum de les tasques amb les dependències i l'estimació d'hores que es dedicarà a cadascuna.

La tasca més llarga és el disseny dels algoritmes per part del director (P1). Té sentit que sigui així ja que són l'essència del treball. Pensar els algoritmes i perfeccionar-ne el pseudo-codi és un esforç intel·lectual que comporta moltes hores. En ser una tasca exclusivament del director, va ser iniciada abans de l'inici del treball de fi de grau. Per això, les tasques exclusives del director (P1 i P2) no es comptabilitzen dins de les 500 hores del treball ni consten al diagrama de Gantt.

A part del disseny d'algoritmes les tasques més costoses són les d'implementació (I1, I2, I2, I3, I4, I5) i optimització (O1, O2). La implementació (i per extensió l'optimització) dels algoritmes

Codi	Tasca	Dependències	Hores	Agent
P1	Dissenyar els algoritmes en pseudo-codi		120+60	Director i col·laboradora
P2	Obtenir instàncies WCSP pels experiments		2	Director
P3	Transformar les instàncies WCSP a MaxSAT	P2	1	Programador
P4	Comprendre els problemes i algoritmes	R1*	20	Programador
P5	Descriure les instàncies	P2	40	Programador
I1	Implementar MHS	P4*	40	Programador
I2	Implementar MaxSAT	I1	40	Programador
I3	Implementar CSP	P4*	40	Programador
I4	Implementar MHV	P4*	40	Programador
I5	Implementar WCSP	I3, I4	60	Programador
O1	Optimitzar MaxSAT	I2	40	Programador
O2	Optimitzar WCSP	I5	40	Programador
E1	Experiment MaxSAT existent	P3	4	Programador
E2	Experiment MaxSAT nostre	P3, O1	4	Programador
E3	Experiment WCSP toulbar2	P2	4	Programador
E4	Experiment WCSP nostre	P2, O2	4	Programador
A1	Anàlisi MaxSAT	E1, E2	10	Programador
A2	Anàlisi WCSP	E3, E4	10	Programador
R1	Reunions plantejament	P1*	12	Director, prog. i col·lab.
R2	Reunions seguiment	R1*	24	Director, prog. i col·lab.
R3	Reunió final	A1, A2	2	Director, prog. i col·lab.
D1	Documentació sobre la gestió		16	Programador
D2	Documentació sobre el contingut		40	Programador
PR	Presentació	A1, A2	10	Programador

Taula 14: Resum de les tasques. Les dependències són final-inici (cal haver acabat la dependència abans de començar la tasca) excepte les marcades amb (*), que són inici-inici (cal haver començat la dependència abans de començar la tasca)

dissenyats pel director és el propòsit d'aquest treball i té sentit que és on el programador hi dediqui més temps.

Els experiments hem considerat el temps que hi dedicarà el personal (en aquest cas el programador), que és relativament poc. El servidor hi dedicarà més temps. El temps que hi dedica depèn del límit que establim i del resultat dels propis experiments (que és el temps d'execució). Establir el primer factor forma part del disseny de l'experiment i el segon no el coneixerem fins que s'hagin executat. Per això no hem previst el temps que hi dedicaran els servidors. Afecta més aviat poc a la planificació, ja que mentre el servidor està treballant nosaltres podem treballar en paral·lel.

A.4.1 Diagrama de Gantt

El diagrama de Gantt apareix a les figures 37, 38 i 39. Hi figuren els dies en què s'executa cada tasca.

Figura 37: Diagrama de Gantt (part 1 de 3).

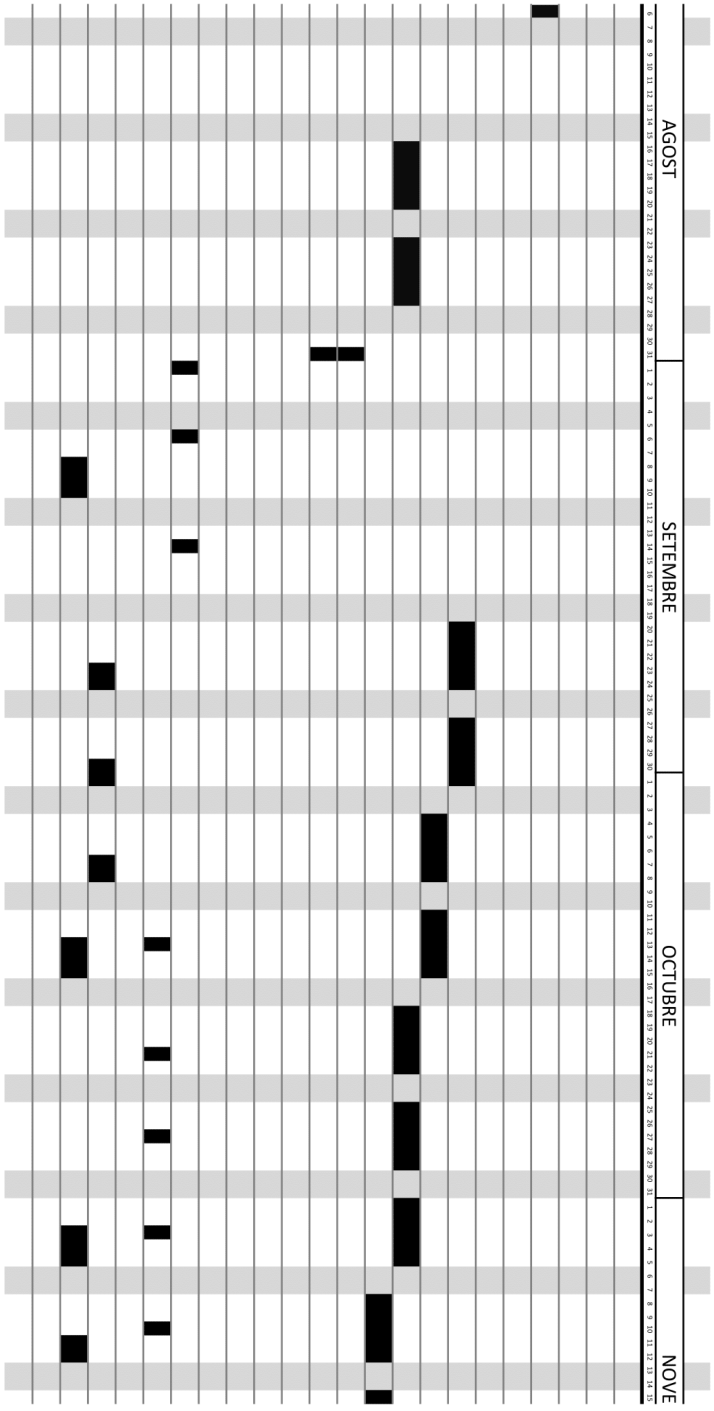

Figura 38: Diagrama de Gantt (part 2 de 3).

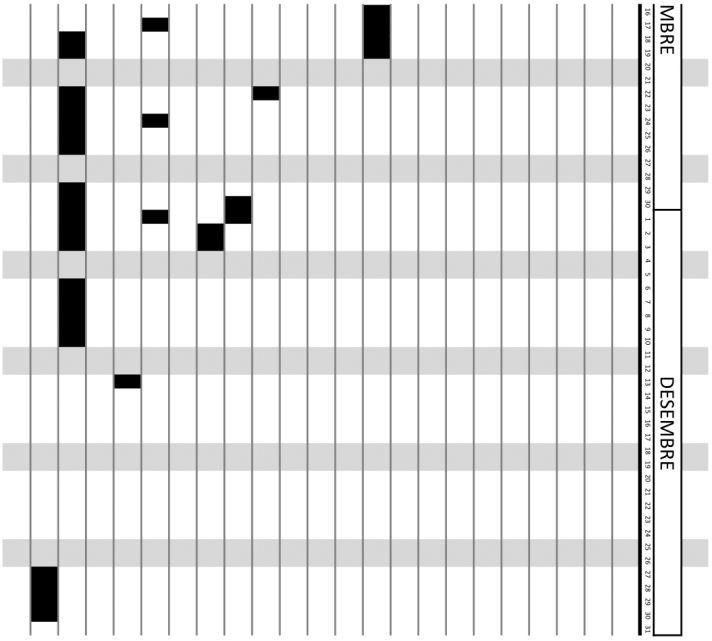

Figura 39: Diagrama de Gantt (part 3 de 3).

A.5 Pressupost

En aquesta secció calcularem un pressupost hipotètic pel projecte. Volem recalcar que es tracta d'un pressupost hipotètic a mode d'exercici i que les dades que conté no tenen per què correspondre amb la realitat.

Dividirem els costos en dos grans grups:

- *Costos de personal.* Salaris de les tres persones que intervenen al projecte.
- *Costos genèrics.* Amortitzacions de material, i quotes de lloguer, electricitat i connexió a internet.

A.5.1 Costos de personal

Hi ha tres persones que treballen en el projecte. L'autor o programador, el director i la col·laboradora.

Els tres treballen a la UPC. Considerarem que tenen les categories professionals que figuren a la taula 15.

	Categoria professional general	Cetegoria professional específica
Director	Personal Docent i Investigador (PDI)	Director/a d'investigació
Col·laboradora	Personal Docent i Investigador (PDI)	Investigador/a ordinari/a
Programador	Personal d'Administració i Serveis (PAS)	Personal de Suport a la Recerca (PSR)

Taula 15: Categories professionals dels desenvolupadors del projecte. (Hipotètic)

A partir de les categories professionals en podem calcular el sou, tal com es mostra a la taula 16.

Els salaris del PDI estan publicats a [66]. Al sou base dels dos PDI s'hi han de sumar els triennis, un per cada tres anys d'experiència en el càrrec. Suposarem que porten entre 9 i 12 anys al càrrec i que, per tant, cobren tres triennis.

El sou dels PSR varia en funció del projecte i les qualificacions de l'empleat. Consultarem les ofertes de feina per PSR, que es fan per concurs públic. [67] és una oferta compatible amb la feina del programador d'aquest projecte; n'emprarem el sou.

El cost del personal inclou el sou brut sumat a la cotització a la seguretat social, que actualment és el 28,3% del sou. [68]

Considerarem que un any laboral són 230 dies = 52 setmanes × 5 dies/setmana — 20 dies de vacances — 10 dies per assumptes propis. Per tant, un any laboral a jornada completa són 1480 hores = 230 dies × 8 hores/dia. Així obtenim el cost/hora.

Multipliquem el cost/hora pel nombre d'hores dedicades al projecte. Les hores les hem obtingut de la planificació de tasques.

	Anual a jornada completa						
	Sou base	Triennis	Sou	Sou+S.Social	Cost/hora	Hores	Cost
Director	38.352,90 €	2.065,98 €	40.418,88 €	51.857,42 €	28,18 €	162	4.565,71 €
Col·laboradora	34.440,46 €	2.065,98 €	36.506,44 €	46.837,76 €	25,46 €	98	2.494,62 €
Programador	25.802,74 €	0,00 €	25.802,74 €	33.104,92 €	17,99 €	501	9.013,89 €
Total						761	16.074,22 €

Taula 16: Costos de personal. (Hipotètic)

A la taula 17 desglossem el cost per tasca.

Fase / Tasca	Hores director	Hores programador	Hores col·laboradora	Cost
Prèvia	**124**	**61**	**60**	**6.119,56 €**
P1	120		60	4.909,32 €
P2	4			112,73 €
P3		1		17,99 €
P4		20		359,84 €
P5		40		719,67 €
Implementació		**220**		**3.958,20 €**
I1		40		719,67 €
I2		40		719,67 €
I3		40		719,67 €
I4		40		719,67 €
I5		60		1.079,51 €
Optimització		**80**		**1.439,34 €**
O1		40		719,67 €
O2		40		719,67 €
Experimentació		**16**		287,87 €
E1		4		71,97 €
E2		4		71,97 €
E3		4		71,97 €
E4		4		71,97 €
Anàlisi		**20**		**359,84 €**
A1		10		179,92 €
A2		10		179,92 €
Reunions	38	38	38	2.721,96 €
R1	12	12	12	859,57 €
R2	24	24	24	1.719,13 €
R3	2	2	2	143,26 €
Documentació		**56**		**1.007,54 €**
D1		16		287,87 €
D2		40		719,67 €
Presentació		**10**		**179,92 €**
PR		10		179,92 €
Total CPA	**162**	**501**	**98**	**16.074,22 €**

Taula 17: Costos de personal per tasca. (Hipotètic)

A.5.2 Costos genèrics

A continuació exposarem els costos genèrics. Es tracta de recursos necessaris per realitzar el projecte durant els sis mesos i mig de desenvolupament. Els utilitzaran els tres desenvolupadors: director, col·laboradora i programador.

Cal tenir en compte que el director i la col·laboradora no dedicaran la totalitat de la seva jornada laboral al projecte. Utilitzaran els mateixos recursos tant quan treballin pel projecte, com quan treballin en altres tasques. Comptabilitzarem com a costos del projecte la part proporcional en funció de les hores dedicades al projecte i les hores no dedicades.

Els dos treballen a jornada completa. Com que abans hem considerat que un any laboral són 1480 hores, sis mesos i mig correspondran a 802 hores.

	Hores totals	Hores projecte	Proporció dedicada al projecte
Director	802	160	20,0%
Col·laboradora	802	38	4,7%
Programador	501	501	100,0%
Mitjana			41,6%

A.5.2.1 Quotes

Suposarem que lloguem un despatx de 40 m^2 a Barcelona perquè hi puguem cabre els tres. A [69] hi figuren nombroses ofertes de lloguer d'oficines a Barcelona, i també se'n indica el preu mitjà per unitat de superfície, que al moment d'escriure aquestes línies, és 16,67 €/mes/m^2. A partir d'aquesta mitjana calculem la mensualitat que pagaríem: 40 m^2 × 16,67 €/mes/m^2 = 666,80 €/mes

Pel que fa la connexió a internet podríem contractar la tarifa [70].

Per calcular el consum elèctric enlloc desglossar-lo per aparell consumidor, ens fixarem en la factura de la llum d'un despatx sililar: hi treballen dues persones a jornada completa i en dos mesos van gastar 293 kWh. Amb això podem estimar que el nostre despatx gastarà 293 / 2 persones × 3 persones = 439,5 kWh en dos mesos, és a dir, 2.673 kWh/any. Considerant un marge d'error de 50%, podríem contractar la tarifa plana de [71] que cobreix fins a 4.000 kWh/any.

	Cost mensual	Mesos	Cost
Lloguer despatx	666,80 €	6,5	4.334,20 €
Connexió a internet	38,00 €	6,5	247,00 €
Electricitat	61,98 €	6,5	402,87 €
Total			4.984,07 €

A.5.2.2 Amortitzacions de material personal

A continuació veurem les amortitzacions del material que necessita cada empleat: un ordinador, una taula i una cadira. Les referències pels preus són [72], [73] i [73], respectivament. Se'n necessiten tres de cada.

	Preu	Vida útil (anys)	ús (anys)	Amortització
Ordinador personal	649 €	5	0,54	70,09 €
Taula	69 €	20	0,54	1,86 €
Cadira	69 €	10	0,54	3,73 €
Subtotal				75,68 €
Unitats			3	
Total				227,04 €

A.5.2.3 Amortitzacions de recursos exclusius del projecte

En aquesta secció tenim en compte els recursos no humans que durant el desenvolupament del projecte seran dedicats exclusivament pel projecte. Es tracta del programari i un servidor per executar experiments. El servidor podria ser un servidor com [74]. El programari serà programari lliure, és a dir, gratuït.

	Preu	Vida útil (anys)	ús (anys)	Amortització
Servidor	1.949 €	8	0,54	131,56 €
Programari lliure	0 €			0 €
Total				131,56 €

A.5.2.4 Suma de costos genèrics

A continuació sumem els costos genèrics que acabem d'esmentar.

A.5.3 Suma de costos

La taula següent suma els costos que hem calculat anteriorment. El subtotal correspon a la suma de costos de personal i costos genèrics. Estimem un 10% addicional per fer front a possibles obstacles.

	Cost total	Dedicació al projecte	Cost pel projecte
Costos quota mensuals	4.984,07 €	41,6%	2.071,52 €
Amortitzacions material personal	227,04 €	41,6%	94,37 €
Amortitzacions exclusives projecte	131,56 €	100%	131,56 €
Total			2.297,44 €

Concepte		Cost
Costos de personal		16.074,22 €
Costos genèrics		2.297,44 €
Subtotal		18.371,66 €
Desviació per obstacles	10%	1.837,16 €
Total		20.604,82 €

A.6 Sostenibilitat

Cal recordar que el projecte consisteix a resoldre un problema abstracte. La resolució per si mateixa no té cap impacte econòmic, social ni ambiental més enllà dels costos econòmics i la creació d'un lloc de treball. Els beneficis directes seran únicament dins de la comunitat científica i aquesta podrà aportar eventualment beneficis econòmics, socials i ambientals a llarg termini. Per avaluar-los caldria analitzar què en farà la comunitat científica, i això no és objecte d'aquest treball. Malgrat tot, ho farem per aquesta anàlisi. Haurem de tenir en compte que les aplicacions del problema són incertes: no sabem per quines serviran el nostre solver funcionarà millor. Partim de la hipòtesi que funcionarà bé pel disseny computacional de proteïnes (CPD), però el mateix CPD no és el producte final sinó que té nombroses aplicacions que no són objecte d'aquest treball i no sabem per quines funcionarà bé el nostre solver. Una d'elles és la creació de medicaments. Per aquesta anàlisi ens centrarem en aquesta.

La incertesa sobre el resultat i les aplicacions de la nostra recerca farà perdre precisió a l'anàlisi de sostenibilitat.

A.6.1 Sostenibilitat econòmica

El cost econòmic del projecte l'hem vist a l'apartat de pressupost. El posarà l'erari públic amb l'esperança que reverteixi positivament a la societat. Els beneficis econòmics probablement seran de les farmacèutiques que produeixin medicaments.

A.6.2 Sostenibilitat social

S'espera que els beneficis socials de la recerca siguin en l'àmbit de la salut. S'espera que el nostre solver ajudi al CPD i aquest ajudi en la creació de medicaments que reverteixin positivament en la salut de la societat.

A.6.3 Sostenibilitat ambiental

Els costos ambientals del treball s'han limitat al consum elèctric, que el podem considerar relativament baix. No s'espera que el resultat de la recerca produeixi beneficis o perjudicis al medi ambient.

Referències

1. Hallen, M. A. & Donald, B. R. Protein Design by Provable Algorithms. *Communications of the ACM* **62,** 76 - 84. https://cacm.acm.org/magazines/2019/10/239678-protein-design-by-provable-algorithms/fulltext# (2019).
2. Hurley, B., O'Sullivan, B., Allouche, D. *et al.* Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints* **21,** 413 - 434. https://doi.org/10.1007/s10601-016-9245-y (2016).
3. Sanchez-Fibla, M., de Givry, S. & Schiex, T. Mendelian Error Detection in Complex Pedigrees Using Weighted Constraint Satisfaction Techniques. *Constraints* **13,** 130 - 154. https://doi.org/10.1007/s10601-007-9029-5 (2007).
4. Bensana, E., Lemaitre, M. & Verfaillie, G. Earth Observation Satellite Management. *Constraints* **4,** 293 - 299. ISSN: 1383-7133. https://doi.org/10.1023/A:1026488509554 (1999).
5. Cabon, B., de Givry, S., Lobjois, L., Schiex, T. & Warners, J. Radio Link Frequency Assignment. **4.** https://doi.org/10.1023/A:1009812409930 (1999).
6. Argelich, J., Le Berre, D., Lynce, I., Silva, J. & Rapicault, P. *Solving Linux Upgradeability Problems Using Boolean Optimization* a *Electronic Proceedings in Theoretical Computer Science* **29** (2010), 11 - 22. https://doi.org/10.4204/EPTCS.29.2.
7. Chen, Y., Safarpour, S., Veneris, A. & Marques-Silva, J. *Spatial and Temporal Design Debug Using Partial MaxSAT* a *Proceedings of the 19th ACM Great Lakes Symposium on VLSI* (Association for Computing Machinery, 2009), 345 - 350. ISBN: 9781605585222. https://doi.org/10.1145/1531542.1531621.
8. Li, C.-M., Jiang, H. & Xu, R.-C. *Incremental MaxSAT Reasoning to Reduce Branches in a Branch-and-Bound Algorithm for MaxClique* a *Learning and Intelligent Optimization* (ed. Dhaenens, C., Jourdan, L. & Marmion, M.-E.) (Springer International Publishing, 2015), 268 - 274. ISBN: 978-3-319-19084-6. https://doi.org/10.1007/978-3-319-19084-6_26.
9. Park, J. *Using weighted MAX-SAT engines to solve MPE* a *Proceedings of the National Conference on Artificial Intelligence* (2002), 682 - 687. https://www.aaai.org/Papers/AAAI/2002/AAAI02-102.pdf.
10. Saikko, P. *Implicit Hitting Set Algorithms for Constraint Optimization* tesi doct. (University of Helsinki, 2019). http://urn.fi/URN:ISBN:978-951-51-5669-3.
11. *Proteins: Definition, Properties, Structure, Classification, Functions* Microbe Notes. https://microbenotes.com/proteins-properties-structure-classification-and-functions/.
12. Delisle, E. & Bacchus, F. *Solving Weighted CSPs by Successive Relaxations* a *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden* (ed. Schulte, C.) **8124** (Springer, 2013), 273 - 281. https://doi.org/10.1007/978-3-642-40627-0_23.
13. Allouche, D., André, I., Barbe, S., Davies, J., de Givry, S. *et al.* Computational protein design as an optimization problem. *Artificial Intelligence* **212,** 59 - 79. https://www.sciencedirect.com/science/article/pii/S0004370214000332 (2014).
14. Traoré, S. *et al.* A new framework for computational protein design through cost function network optimization. *Bioinformatics* **29,** 2129 - 2136. ISSN: 1367-4803. https://doi.org/10.1093/bioinformatics/btt374 (2013).
15. Ouali, A. *et al. Iterative Decomposition Guided Variable Neighborhood Search for Graphical Model Energy Minimization* a *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017* (ed. Elidan, G., Kersting, K. & Ihler, A. T.) (AUAI Press, 2017). http://auai.org/uai2017/proceedings/papers/197.pdf.
16. *wcsp2sat* http://genoweb.toulouse.inra.fr/~degivry/evalgm/scripts/wcsp2sat.cc.
17. *toulbar2* https://toulbar2.github.io/toulbar2.
18. *toulbar2* GitHub. https://github.com/toulbar2/toulbar2.
19. *RC2 MaxSAT solver* PySAT. https://pysathq.github.io/docs/html/api/examples/rc2.html.

20. *MaxHS* http://www.maxhs.org/.
21. Davies, J. *Solving MaxSAT by decoupling optimization and satisfaction* tesi doct. (University of Toronto, 2013). http://www.maxhs.org/docs/Davies_Jessica_E_201311_PhD_thesis.pdf.
22. Rossi, F., van Beek, P. & Walsh, T. *Handbook of Constraint Programming* ISBN: 9780080463803. https://books.google.es/books?id=Kjap9ZWcKOoC&q=handbook+of+constraint+programming&pg=PP1&redir_esc=y#v=onepage&q&f=false (Elsevier, 2006).
23. Jaffar, J. & Lassez, J. L. Constraint logic programming. *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages,* 111 - 119. https://doi.org/10.1145/41625.41635 (1987).
24. *Constraint programming* Wikipedia. https://en.wikipedia.org/wiki/Constraint_programming.
25. *CP Conference Series* Association for Constraint Programming. https://www.a4cp.org/events/cp-conference-series.
26. Borchers, B. & Furman, J. A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT Problems. *Journal of Combinatorial Optimization,* 299 - 306. https://doi.org/10.1023%5C%2FA%5C%3A1009725216438 (1998).
27. *Clique problem* Wikipedia. https://en.wikipedia.org/wiki/Clique_problem.
28. *MaxSAT Evaluations* https://maxsat-evaluations.github.io/.
29. Chandrasekaran, K., Karp, R., Moreno-Centeno, E. & Vempala, S. *Algorithms for Implicit Hitting Set Problems* a *Proceedings of the 2011 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2011), 614 - 629. https://doi.org/10.1137/1.9781611973082.48.
30. Saikko, P., Wallner, J. P. & Järvisalo, M. *Implicit hitting set algorithms for reasoning beyond NP* a *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning* (2016). https://www.aaai.org/ocs/index.php/KR/KR16/paper/viewPaper/12812.
31. *Graph coloring* Wikipedia. https://en.wikipedia.org/wiki/Graph_coloring.
32. *Second International CSP Solver Competition* http://www.cril.univ-artois.fr/CPAI06/.
33. *Third International CSP Solver Competition* http://www.cril.univ-artois.fr/CPAI08/.
34. *Fourth International CSP Solver Competition* http://www.cril.univ-artois.fr/CSC09/.
35. Dawn Teare, M. & Barrett, J. H. Genetic linkage studies. *The Lancet* **366,** 1036 - 1044. ISSN: 0140-6736. https://doi.org/10.1016/S0140-6736(05)67382-5 (2005).
36. Latham, K. *Protein Structure* Biology Dictionary. https://biologydictionary.net/protein-structure/.
37. Harder, T. *et al.* Beyond rotamers: a generative, probabilistic model of side chains in proteins. *BMC Bioinformatics.* https://doi.org/10.1186/1471-2105-11-306 (2010).
38. Gainza, P., Roberts, K. E. & Donald, B. R. Protein design using continuous rotamers. *PLoS computational biology.* https://doi.org/10.1371/journal.pcbi.1002335 (2012).
39. Allouche, D., Barbe, S., Givry, S. *et al. Cost Function Networks to Solve Large Computational Protein Design Problems* https://www.researchgate.net/publication/334387860_Cost_Function_Networks_to_Solve_Large_Computational_Protein_Design_Problems (2019).
40. Baran, D. *et al.* Principles for computational design of binding antibodies. *Proceedings of the National Academy of Sciences* **114,** 10900 - 10905. https://doi.org/10.1073/pnas.1707171114 (2017).
41. Norman, R. A. *et al.* Computational approaches to therapeutic antibody design: established methods and emerging trends. *Briefings in Bioinformatics* **21,** 1549 - 1567. ISSN: 1477-4054. https://doi.org/10.1093/bib/bbz095 (2019).
42. Vojcic, L. *et al.* Advances in protease engineering for laundry detergents. *New Biotechnology* **32.** European Congress of Biotechnology - ECB 16, 629 - 634. ISSN: 1871-6784. https://doi.org/10.1016/j.nbt.2014.12.010 (2015).
43. Khan, M. F., Kundu, D., Hazra, C. & Patra, S. A strategic approach of enzyme engineering by attribute ranking and enzyme immobilization on zinc oxide nanoparticles to attain thermostability in mesophilic Bacillus subtilis lipase for detergent formulation. *International journal of biological macromolecules.* https://doi.org/10.1016/j.ijbiomac.2019.06.042.

44. Li, Q. *et al.* Computational design of a cutinase for plastic biodegradation by mining molecular dynamics simulations trajectories. *Computational and Structural Biotechnology Journal* **20,** 459 - 470. ISSN: 2001-0370. https://doi.org/10.1016/j.csbj.2021.12.042 (2022).
45. Cohen, D., Cooper, M. & Jeavons, P. *A Complete Characterization of Complexity for Boolean Constraint Optimization Problems* a *Lecture Notes in Computer Science - 10th International Conference CP 2004 - Proceedings* (2004), 212 - 226. ISBN: 978-3-540-23241-4. http://doi.org/10.1007/978-3-540-30201-8_18.
46. Ausiello, G., Crescenzi, P. & Protasi, M. Approximate solution of NP optimization problems. *Theoretical Computer Science* **150,** 1 - 55. ISSN: 0304-3975. https://www.sciencedirect.com/science/article/pii/030439759400291P (1995).
47. Pierce, N. A. & Winfree, E. Protein Design is NP-hard. *Protein Engineering, Design and Selection* **15,** 779 - 782. ISSN: 1741-0126. https://doi.org/10.1093/protein/15.10.779 (2002).
48. *Set cover problem* Wikipedia. https://en.wikipedia.org/wiki/Set_cover_problem.
49. Moreno-Centeno, E. & Karp, R. M. The Implicit Hitting Set Approach to Solve Combinatorial Optimization Problems with an Application to Multigenome Alignment. *Oper. Res.* **61,** 453 - 468. https://doi.org/10.1287/opre.1120.1139 (2013).
50. Bacchus, F., Hyttinen, A., Järvisalo, M. & Saikko, P. *Reduced Cost Fixing in MaxSAT* a *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings* (ed. Beck, J. C.) **10416** (Springer, 2017), 641 - 651. https://doi.org/10.1007/978-3-319-66158-2%5C_41.
51. *DIMACS CNF file format* https://jix.github.io/varisat/manual/0.2.0/formats/dimacs.html.
52. *PySAT documentation* https://pysathq.github.io/docs/pysat.pdf.
53. *CPLEX Python API Reference Manual* IBM. https://www.ibm.com/docs/en/icos/12.8.0.0?topic=cplex-python-api-reference-manual.
54. *Tutorial: Beyond Linear Programming (CPLEX Part2)* https://ibmdecisionoptimization.github.io/tutorials/html/Beyond_Linear_Programming.html#Integer-Optimization.
55. *CP and WCSP file formats* toulbar2. https://forgemia.inra.fr/thomas.schiex/toulbar2/-/blob/master/doc/CpWcspFormats.txt.
56. *CaDiCaL Simplified Satisfiability Solver* JKU. http://fmv.jku.at/cadical/.
57. *CaDiCaL Simplified Satisfiability Solver* Github. https://github.com/arminbiere/cadical.
58. Biere, A., Fazekas, K., Fleury, M. & Heisinger, M. *CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020* a *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions* (ed. Balyo, T. *et al.*) **B-2020-1** (University of Helsinki, 2020), 51 - 53. http://fmv.jku.at/papers/BiereFazekasFleuryHeisinger-SAT-Competition-2020-solvers.pdf.
59. *Constraint Propagation* Guide to Constraint Programming. https://ktiml.mff.cuni.cz/~bartak/constraints/propagation.html.
60. *SPOT5 instances (CP13)* http://genoweb.toulouse.inra.fr/~degivry/evalgm/CFN/SPOT5/.
61. *Linkage pedigree instances (CP13)* http://genoweb.toulouse.inra.fr/~degivry/evalgm/MRF/Linkage/.
62. *CPD instances (AIJ14)* http://genoweb.toulouse.inra.fr/~tschiex/CPD-AIJ/.
63. *CPD instances (Bio13)* http://genoweb.toulouse.inra.fr/~tschiex/CPD/.
64. *CPD instances (UAI17)* http://genoweb.toulouse.inra.fr/~degivry/evalgm/CFN/ProteinDesignUAI2017/.
65. *Normativa del treball final de grau del grau en Enginyeria Informàtica de la Facultat d'Informàtica de Barcelona* UPC. https://www.fib.upc.edu/sites/fib/files/documents/estudis/normativa-tfg-mencio-addicional-gei-br.pdf.
66. *Taules retributives del personal docent i investigador. Any 2020.* UPC. https://www.upc.edu/transparencia/ca/informacio-de-personal/sdg8_2_1_retribuciones_pdi_2020-1.pdf.

67. *Tècnic/a de Suport a la Recerca (Oferta de feina)*. UPC. https://rdi.upc.edu/ca/uaslr/vols-dedicar-te-a-la-recerca/ofertes-PSR/concursos-psr/concursos-actius/Documentacio%5C%20PSR/2021%5C%20PSR/res-00667-2021-de-13-de-maig/fitxa-150-723-115.pdf.

68. *Seguridad Social: Bases y tipos de cotización 2021* Ministerio de Inclusión, Seguridad Social y Migraciones. https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537#36538.

69. *Oficines de lloguer a Barcelona* idealista. https://www.idealista.com/ca/alquiler-oficinas/barcelona-barcelona/.

70. *Tarifas de internet* Movistar. https://www.movistar.es/particulares/tienda/comparador-tarifas-internet/.

71. *Tarifa plana de luz* Comparadorluz. https://comparadorluz.com/faq/conviene-tarifa-plana-luz-gas.

72. *Portátil ASUS* Mediamarkt. https://www.mediamarkt.es/es/product/_portatil-asus-d515ua-br279-156-hd-amd-ryzentm-7-5700u-8-gb-ram-512-gb-ssd-radeon-freedos-plata-1515772.html?gclid=Cj0KCQjwnoqLBhD4ARIsAL5JedIIdUsBz1x0fHjM4YwSfeSEvL7o8UPuhlN6045i_aO5GQlOvnz8YOQaArJKEALw_wcB&gclsrc=aw.ds&utm_campaign=MM_ES_SEARCH_GOOGLE_CATEGORIES_PLA_PLA-SMART-PH_INFORMATICA_ALL_ALL&utm_medium=cpc&utm_source=google.

73. *Escriptori per ordinador* IKEA. https://www.ikea.com/es/ca/cat/escriptoris-i-escriptoris-per-l-ordinador-20649/.

74. *XPS Tower* Dell. https://www.dell.com/es-es/shop/sobremesas-de-dell/xps-tower/spd/xps-8940-desktop/cdx89435co?gacd=9955562-5295-5761040-294411634-0&dgc=ST&gclid=Cj0KCQjwnoqLBhD4ARIsAL5JedLQjnQ7fX1sqeCpwuwUhMb14RnFFk5aC7BhDLCjAVwc5YlZ2RjCmzEaAka1wcB&gclsrc=aw.ds.

Glossari

benchmark Conjunt d'instàncies per mesurar el rendiment d'un programa informàtic. 6

constraint Restricció. 13, 17

core Subconjunt insatisfactible d'una instància insatisfactible (d'un problema de decisió). 14

CPD Disseny Computacional de Proteïnes (en anglès, *Computational Protein Design*). 2.4. 5, 7, 22

CSP *Constraint Satisfaction Problem*. 2.3.1. 13, 17

HS *Hitting Set*. 3.1.1. 25

HV *Hitting Vector*. 3.2.1. 29

IMHS *Implicit Minimum Cost Hitting Set Problem*. 3.1.2. 25

IMHV *Implicit Minimum Cost Hitting Vector Problem*. 3.2.2. 29

instància Conjunt de dades que conformen l'input d'un problema. 6

lower bound Cota inferior (a l'òptim). 26

MaxSAT *Weighted Boolean Maximum Satisfiability Problem*. 2.2.2. 5, 6, 13, 14

MHS *Minimum Cost Hitting Set (Problem)*. 3.1.1. 25

MHV *Minimum Cost Hitting Vector (Problem)*. 3.2.1. 29

SAT *Boolean Satisfiability Problem*. 2.2.1. 13, 14

upper bound Cota superior (a l'òptim). 26

WCSP *Weighted Constraint Satisfaction Problem* (algunes publicacions l'anomenen *Cost Function Networks*, CFN). 2.3.2. 5, 6, 13, 17

òptim Solució (d'un problema d'optimització) amb el mínim cost possible. 13