# Disseny,implementació i avaluació d'algorismes Max-SAT multivaluats

**David Barroso Iglesias**
Enginyeria en Informàtica

**Vicenç Torra Reventós**

17 de Juny de 2004

# Índex

# Índex de figures

# 1

# Introducció

## 1.1 Context

El problema de la satisfactibilitat proposicional de les fórmules booleanes, conegut com a problema SAT, consisteix en decidir si una fórmula en forma normal conjuntiva (CNF) és satisfactible, es a dir, si existeix una assignació de valors a variables que satisfà totes les clàusules de la fórmula.

Una de les àrees de recerca actives de la Intel·ligència Artificial és la utilització de fórmules booleanes proposicionals com a llenguatge de programació amb restriccions per a solucionar problemes NP-complets.

Aquest mètode de resolució consisteix en:

1. Reduir el problema que volem resoldre al problema SAT.
2. Solucionar la instància obtinguda amb un algorisme de satisfactibilitat.
3. Construir una solució per al problema original a partir de la solució obtinguda per a la instància SAT.

Aquest enfocament ha demostrat ser molt competitiu en camps com la verificació de circuits electrònics, planificació i scheduling, i la seva utilitat ha provocat el desenvolupament de nombrosos resoledors SAT complerts i ràpids com Chaff[8], Grasp[11], RelSat[2] i Satz[6], tots ells basats en el procediment Davis-Putnam-Logemann-Loveland (DPLL)[4].

El problema Max-SAT, d'altra banda, consisteix en trobar una assignació de valors de veritat que maximitzi el nombre de clàusules satisfetes d'una fórmula CNF. Alguns del algorismes exactes de resolució del problema Max-SAT són variants del procediment DPLL. Un és el Wallace i Freuder[9] i l'altre és el Borchers i Furman[3]. Els dos són algorismes de Branch&Bound

amb profunditat prioritària. Malgrat que actualment estan apareixent algorismes cada cop més sofisticats en la literatura especialitzada, el disseny, implementació i avaluació d'algorismes per a resoldre el problema Max-SAT no està tan avançat com el desenvolupament d'algorismes per a resoldre el problema SAT

## 1.2 Justificació

Una de les línies de recerca dins del camp de la satisfactibilitat proposicional és la resolució de problemes codificats amb lògica multivaluada, donat que històricament s'ha utilitzat la lògica booleana per a representar els problemes i el formalisme multivaluat presenta alguns avantatges:

- La codificació amb lògica multivaluada preserva millor la seva estructura original dels problemes, de forma que aquests es poden codificar de forma més natural i compacta.
- És més proper al paradigma CSP.
- Els algorismes desenvolupats per a la resolució del problema SAT amb codificació multivaluada han demostrat ser més competitius, en un ampli espectre de problemes, que els que utilitzen codificació booleana.

Actualment no existeixen algorismes de resolució de problemes Max-SAT codificats amb lògica multivaluada, de forma que resulta força interessant fer una primera exploració en aquest sentit per avaluar la seva viabilitat i assentar les bases que facin possible la futura construcció d'algorismes competitius. Des d'un punt de vista aplicat, l'objectiu és definir un mètode genèric de resolució de problemes amb restriccions suaus (overconstrained problems)

## 1.3 Objectius

La finalitat d'aquest projecte final de carrera és definir el problema Max-SAT amb codificació multivaluada, implementar algorismes exactes de resolució del problema i construir un generador aleatori de problemes que permeti avaluar aquests algorismes.

Per a fer-ho, haurem de:

- Estudiar els algorismes Max-SAT amb codificació booleana existents, per tal de conèixer el seu funcionament i les millores introduïdes amb el temps.

- Definir un format de representació de fórmules clausals multivaluades.
- Dissenyar i implementar un generador de problemes per avaluar els algorismes a construir.
- Dissenyar i implementar un algorisme general de resolució del problema Max-SAT multivaluat.
- Estudiar, dissenyar i implementar millores i variants heurístiques a l'algorisme.
- Fer una avaluació experimental per extreure conclusions.

L'avaluació dels algorismes haurà de permetre fixar les línies de treball futur, prioritzant les variants algorísmiques que es demostrin òptimes.

## 1.4 Punt de partida i aportacions del PFC

Aquest projecte parteix de la implementació de l'algorisme per a la resolució del problema Max-SAT codificat amb lògica booleana feta per Borchers&Furman i millorat posteriorment per Alsinet,Manyà i Planes[1], donat que estan disponibles públicament.

Per a la construcció del generador de problemes, es partirà d'un generador de problemes CSP que es codificaran amb lògica multivaluada segons un format a definir.

Les aportacions d'aquest TFC han sigut:

- Definició d'un format multivaluat de representació de formes clausals.
- Implementació d'un generador de formes clausals multivaluades a partir d'un generador de CSP's.
- Adaptació de l'algorisme de Borchers&Furman a fórmules multivaluades, amb la incorporació d'estructures lazy.
- Incorporació de variants heurístiques de sel·lecció de variable per nombre d'aparicions.
- Incorporació de branching binari.
- Incorporació d'una estimació per sota per millorar la qualitat de la cota inferior.
- Estudi del rendiment dels algorismes, extracció de conclusions i establiment de línies de treball futur.

## 1.5 Enfocament i mètode seguit

Tots els productes generats en aquest treball final de carrera seran implementats amb Java, per diversos motius:

- La seva orientació a objectes, que permet estructurar el codi de forma modular i reutilitzable. La varietat d'algorismes a implementar fa aconsellable utilitzar aquestes característiques.
- La seva portabilitat: El projecte es desenvoluparà sota una plataforma GNU/Linux, però ha de poder funcionar sobre la plataforma estàndard recomanada per la UOC, sota sistema operatiu Windows.
- La penalització de rendiment que pot suposar l'ús d'un llenguatge interpretat no és important en aquest treball donat que el seu objectiu no és comparar-se amb algorismes competitius.

En tot el treball es seguirà el mètode clàssic de desenvolupament de programari, consistent en l'aplicació reiterada dels següents passos:

- Estudi
- Disseny
- Implementació
- Avaluació
- Millora

## 1.6 Productes

Els productes a lliurar al finalitzar el treball són:

- Els algorismes implementats.
- El generador de problemes com a instàncies de Max-SAT multivaluat.
- Un exemple de fitxer amb el format de representació de clàusules multivaluades.

## 1.7 Breu descripció de la resta de capítols

- Capítol 2: Algorismes Branch&Bound per a Max-SAT

  En aquest capítol descriurem els algorismes Branch&Bound per a Max-SAT; en particular descriurem el algorisme de Borchers i Furman i les millores aportades per Alsinet,Manyà i Planes.

- Capítol 3: Disseny d'algorismes Branch&Bound per a Max-SAT multivaluat

  En aquest capítol descriurem la forma com es dissenyen algorismes Branch&Bound per a Max-SAT multivaluat, que són les fórmules clausals multivaluades i com es representen, així com la forma com es generen problemes representats en aquest format.

- Capítol 4: Resultats experimentals

  En aquest capítol mostrarem els resultats obtinguts en l'avaluació dels diferents algorismes, amb la finalitat de poder extreure conclusions sobre quins són els millors algorismes per a cada tipus de problema

- Capítol 5: Conclusions i treballs futurs

  En aquest capítol farem un resum de conclusions del treball final de carrera i s'identificaran les principals línies de treball futur.

# 2

# Algorismes Branch&Bound per a Max-SAT

## 2.1 Introducció

En aquest capítol partim de varies definicions amb les que es pretén mostrar en que consisteix el paradigma de codificació anomenat forma normal conjuntiva (CNF), com s'aplica a la representació de problemes i com funcionen els algorismes de resolució del problema Max-SAT codificat mitjançant CNF que utilitzen alguns dels resoledors existents actualment, així com les millores i variants introduïdes en aquests al llarg del temps.

## 2.2 Definicions preliminars

**Definició 1** *Sigui* $V = \{x_1, x_2, \ldots, x_n\}$ *un conjunt de variables proposicionals. Un literal booleà és una variable proposicional (literal amb polaritat positiva) o una variable proposicional precedida del símbol de de negació (literal amb polaritat negativa). El complement* $\bar{L}$ *d'un literal* $L$ *és* $\neg x$ *si* $L = x$ *ó* $x$ *si* $L = \neg x$.

**Definició 2** *Una clàusula booleana és una disjunció de literals booleans i és unitària si conté un únic literal. La clàusula buida es representa per* $\square$.

**Definició 3** *Una fórmula booleana en forma normal conjuntiva (CNF) és una conjunció de clàusules booleanes. La fórmula buida es representa per* $\emptyset$.

**Definició 4** *Una interpretació booleana assigna a cada variable proposicional el valor veritat o fals. Una interpretació satisfà un literal* $p(\neg p)$ *si assigna a* $p(\neg p)$ *el valor veritat(fals). Una interpretació satisfà una clàusula si satisfà*

*almenys un dels seus literals. Una interpretació satisfà una fórmula CNF si satisfà totes les seves clàusules.*

L'espai composat per totes les possibles assignacions de valors de veritat a les variables d'una fórmula CNF pot ser representat com a un arbre de cerca, on els nodes interns representen assignacions parcials (no s'ha assignat valor a totes les variables) i les fulles representen assignacions complertes.

**Definició 5** *Donada una fórmula $\phi$ amb una clàusula unitària formada pel literal $L$, la regla del literal unitari (*one-literal rule*[7]) deriva una clàusula $\phi'$ eliminant de $\phi$ totes les clàusules que continguin el literal $L$ i eliminant el literal $\bar{L}$ de totes les clàusules que el continguin.*

**Definició 6** *Donada una fórmula $\phi$, la propagació unitària consisteix en l'aplicació reiterada de la regla del literal unitari sobre $\phi$ fins a derivar un estat de saturació, es a dir, fins eliminar totes les clàusules unitàries de $\phi$, derivar $\phi = \emptyset$ o derivar la clàusula buida.*

**Definició 7** *Donada una fórmula booleana $\phi$, el problema Max-SAT consisteix en trobar una interpretació que maximitzi el nombre de clàusules de $\phi$ satisfetes*

## 2.3 Algorismes

Un algorisme Branch&Bound per a Max-SAT booleà explora l'arbre de cerca en profunditat prioritària, cercant l'assignació complerta que satisfà el màxim nombre de clàusules de la fórmula. En cada node, l'algorisme compara el nombre de clàusules no satisfetes per la millor assignació complerta trobada fins el moment - anomenat *upper bound* ($UP$) - amb el nombre de clàusules no satisfetes per l'assignació parcial actual (*unsat*) més una estimació per sota del nombre de clàusules que deixarien de satisfer-se si completéssim l'assignació parcial actual (*understimation*). La suma de *unsat* + *understimation* s'anomena *lower bound* ($LB$). Si $UB \leq LB$, no es podrà trobar una assignació millor en aquest punt de la cerca, de forma que l'algorisme ignora el subarbre per davall del node actual i fa un salt enrera (*backtraking*) cap a un nivell superior de l'arbre de cerca. Si $UB > LB$, estén l'assignació instanciant una nova variable i creant dos noves branques a l'arbre de cerca: la branca esquerra correspon a la nova variable instanciada a fals ($\neg p$)i la branca dreta correspon a la nova variable instanciada a cert ($p$). Les fórmules corresponents a cada branca s'obtenen aplicant la

regla del literal unitari (*one-literal rule*). La fórmula corresponent a la branca esquerra (dreta) s'obté de la fórmula del node actual esborrant totes les clàusules que contenen la variable $\neg p(p)$ i esborrant totes les ocurrències del literal $p(\neg p)$. La solució de Max-SAT és el valor que pren $UB$ al finalitzar l'exploració de tot l'arbre de cerca.

Els aspectes clau en la implementació d'un algorisme branch&bound per a Max-SAT són:

- El mètode de càlcul de la fita inferior (*lower bound*). Exemples:
  - **LB1** = *unsat*: Aquest mètode no contempla *understimation*, de forma que la fita inferior correspon al nombre de clàusules no satisfetes per l'assignació parcial actual.
  - **LB2** = $unsat + \sum_{p \in \phi'}$ mín(ic(p),ic($\neg$ p)): on $\phi'$ és la fórmula corresponent a l'assignació parcial actual i ic(p) (ic($\neg$p)) és el comptador de inconsistències de p ($\neg$p), es a dir, el nombre de clàusules que es deixen de satisfer si s'estén l'assignació parcial actual fixant p ($\neg$p) a fals. ic(p) (ic($\neg$p)) coincideix amb el nombre de clàusules unitàries de $\phi'$ que contenen $\neg$p (p).
    Contant, per a cada variable de la fórmula, el nombre de clàusules unitàries que contenen la variable en literals positius i negatius, sabrem el nombre mínim de clàusules unitàries que no es satisfaran quan instanciem la variable a qualsevol valor. Sumant aquest valor per a totes les variables al nombre de clàusules no satisfetes per l'assignació parcial actual (*unsat*), tindrem una fita inferior del nombre de clàusules que es deixen de satisfer si estenem aquesta assignació parcial
- L'heurística de sel·lecció de la variable a instanciar. Exemples:
  - **MOMS**: Sel·lecciona una variable entre aquelles que apareixen amb més freqüència en clàusules de menor longitud.
  - **Jeroslow-Wang[5]**: Donada una fórmula $\phi$, per a cada literal $L$ es defineix la següent funció:

    $$J(L) = \sum_{L \in C \in \phi} 2^{-|C|}$$

    on $C$ és la longitud de la clàusula $C$. JW sel·lecciona una variable $p$ de $\phi$ entre aquelles que maximitzen $J(p) + J(\neg p)$.

Borchers&Furman han implementat un algorisme utilitzant LB1 i MOMS, alhora que incorporen dos millores significatives:

**Input:** **max-sat**$(\phi, ub)$ : A boolean CNF formula $\phi$ and an upper bound $ub$
1: **if** $\phi = \emptyset$ or $\phi$ only contains empty clauses **then**
2: return **empty-clauses**$(\phi)$
3: **endif**
4: **if** lower-bound$(\phi) = ub - 1$ **then**
5: $\phi \leftarrow$ unit-propagation$(\phi)$
6: **endif**
7: **if** lower-bound$(\phi) \geq ub$ **then**
8: return $\infty$
9: **endif**
10: $p \leftarrow$ select-variable$(\phi)$
11: $p \leftarrow \min(ub, \text{max-sat}(\phi_{\neg p}, ub))$
12: $ub \leftarrow \min(ub, \text{max-sat}(\phi_p, ub))$
**Output:** The maximum number of clauses of $\phi$ than can be satisfied

Figura 2.1: Algorisme Branch&Bound per a Max-SAT

- Abans de començar l'exploració de l'arbre de cerca, obtenen una fita superior del nombre de clàusules no satisfetes en una solució òptima utilitzant el procediment GSAT[10] de cerca local. Aquesta fita s'utilitza com a *upper bound* inicial de l'algorisme branch&bound. Aquesta millora permet resoldre instàncies del problema fins a set vegades més ràpid.

- Apliquen propagació unitària segura. Els resoledors SAT apliquen la propagació unitària per a simplificar la fórmula inicial i les intermedies, però els algorismes Max-SAT podrien retornar resultats no òptims en cas de fer-ho. Borchers&Furman es van adonar que en el cas que la diferència entre el *lower bound* i l'*upper bound* sigui d'una unitat, es pot aplicar propagació unitària de forma segura, donat que fixant a fals qualsevol literal de qualsevol de les clàusules unitàries s'arriba al *upper bound*

Recentment Alsinet,Manyà i Planes han desenvolupat algorismes derivats del Borchers&Furman aplicant les quatre possibles combinacions LB1+MOMS, LB1+JW, LB2+MOMS i LB2+JW, demostrant que LB2+MOMS i LB2+JW milloren els resultats de LB1+MOMS i LB1+JW.

La figura 2.1 mostra el pseudocodi del nucli dels algorismes descrits amb anterioritat, utilitzant la següent notació:

- **empty-clauses($\phi$)** és una funció que retorna el nombre de clàusules existents a la fórmula $\phi$.

- **lower-bound($\phi$)** és la funció que calcula el lower-bound, ja sigui LB1 o LB2.
- **ub** és una cota superior del nombre de clàusules no satisfetes per una solució òptima. Inicialment pot prendre com a valor el nombre de clàusules de la fórmula o el resultat d'aplicar GSAT.
- **select-variable($\phi$)** és la funció que retorna la pròxima variable a instanciar, segons l'heurística aplicada (MOMS o JW).
- $\phi_p$ ($\phi_{\neg p}$) és la fórmula obtinguda d'aplicar la regla del literal unitari a $\phi$ utilitzant el literal p ($\neg p$).

## 2.4 Estructures de dades per a representar fórmules CNF

Les estructures de dades que s'utilitzen per a representar fórmules són decisives per realitzar eficientment les operacions que constitueixen un algorisme de satisfactibilitat.

### 2.4.1 Estructures basades en llistes d'adjacència i comptadors

En aquest tipus d'estructura de dades, les clàusules es representen com a llistes de literals i, per a cada variable, es manté una llista de totes les clàusules que contenen aquesta variable. Per a saber si es satisfà, s'associa a cada clàusula un comptador de literals que es satisfan i un de literals que no es satisfan amb l'assignació parcial actual. Una clàusula no es satisfà si el comptador de literals que no es satisfan és igual al nombre de literals de la clàusula i es satisfà si el comptador de literals que es satisfan és major o igual a un. La clàusula és unitària si el comptador de literals que no es satisfan és igual al nombre de literals de la clàusula menys un i queda un literal sense assignar. Quan una clàusula es declara unitària, es fa un recorregut de la llista dels seus literals per a identificar aquell que encara no té assignat valor. Quan es detecta un conflicte i es fa *backtracking*, s'han d'actualitzar els comptadors de totes les clàusules en que apareix la variable o variables que passen a no estar assignades.

L'inconvenient d'aquest tipus d'estructures de dades és que cada cop que s'assigna o des-assigna valor a una variable, s'ha d'analitzar totes les clàusules que contenen aquesta variable.

### 2.4.2 Estructures mandroses (lazy)

Les estructures de dades mandroses tracten de retardar l'avaluació d'una clàusula fins el moment en que és imprescindible fer-ho, que coincideix amb el moment en que s'instancia l'últim literal. D'aquesta forma la clàusula no s'avalua fins que s'han instanciat tots els literals que hi ocorren. Si la interpretació actual no satisfà cap literal, derivem la clàusula buida.

En el cas de voler efectuar càlculs de fites inferiors com LB2 (vist a l'apartat 2.2), podem avançar l'avaluació al moment en que s'instancia el penúltim literal, de forma que si la interpretació actual no satisfà el penúltim i anteriors literals, obtenim una clàusula unitària.

La decisió del literal a utilitzar està en funció del càlcul que es vulgui fer de la cota inferior. En aquest treball, els algorismes sense understimation utilitzen apuntadors a l'última literal, mentre que els algorismes amb understimation utilitzen el penúltim. D'altra banda, les estructures utilitzades en aquest treball estan pensades per a heurístiques de sel·lecció de variable estàtiques, es a dir, determinades amb anterioritat a l'inici de l'algorisme. Podríem definir estructures de dades mandroses per a heurístiques dinàmiques, amb el risc de perdre eficiència.

L'estructura de dades utilitzada és, per tant, la següent:

- Cada clàusula està formada per una llista dels literals que conté.
- Cada clàusula té un apuntador a l'últim (penúltim) literal segons l'ordre estàtic de sel·lecció de variables.
- Per a cada literal positiu i negatiu mantenim una llista de les clàusules en les que el literal és l'últim (penúltim) en l'ordre estàtic de sel·lecció de variables. D'aquesta forma, quan assignem valor a una variable $p$, únicament avaluem les clàusules que apareixen a la llista del literal $\neg p$.

En la figura 2.2 podem veure una mostra de la construcció d'aquestes estructures per al cas en que es manté l'apuntador a l'últim literal en l'ordre estàtic de sel·lecció de variable.

Amb aquestes estructures de dades aconseguim

- Reduir a constant el cost del backtracking, donat que no s'han de desfer comptadors.
- Tractar en cada pas únicament aquelles clàusules que poden fer variar la situació de l'algorisme.

Recentment, Alsinet,Manyà i Planes han utilitzat estructures d'aquest tipus en els seus algorismes derivats del Borchers&Furman i han demostrat que el seu rendiment és superior als algorismes que no les implementen.

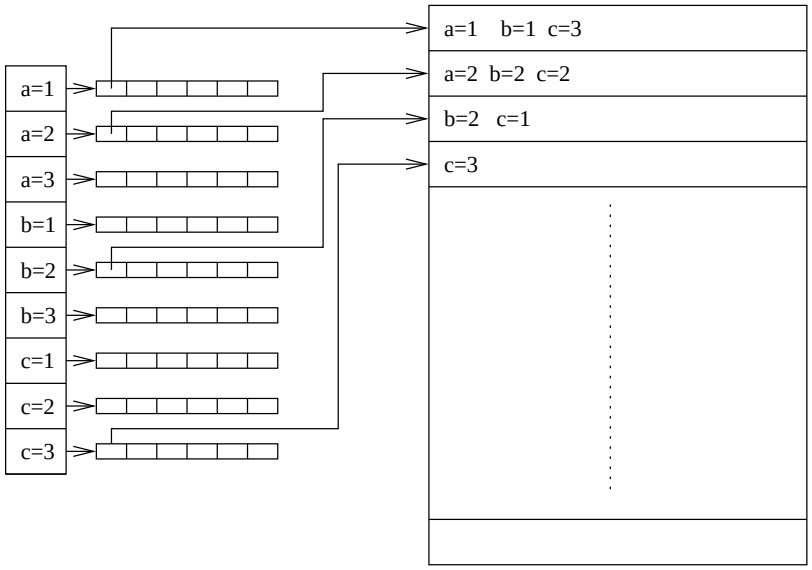


Figura 2.2: Exemple de construcció d'estructures Lazy

# 3

# Disseny d'algorismes Branch&Bound per a Max-SAT multivaluat

## 3.1 Introducció

En aquest capítol partim de varies definicions amb les que es pretén mostrar en que consisteix el paradigma de codificació amb formes clausals multivaluades, com s'aplica a la representació de problemes i quines diferències existeixen amb el paradigma CNF. El canvi de paradigma de representació dels problemes suposa definir el seu format, la forma d'obtenir-ne instàncies i com afecta als algorismes resoledors. En definitiva, la primera part del capítol fixa les bases de definició dels algorismes a implementar en aquest treball final de carrera.

## 3.2 Definicions preliminars

**Definició 8** *Sigui $V = \{x_1, x_2, \ldots, x_n\}$ un conjunt de variables proposicionals i $N$ un conjunt totalment ordenat de valors de veritat, anomenat domini, tal que $|N| \geq 2$. Un literal multivaluat és una expressió de la forma $x_i = k$ o $x_i \neq k$, on $x_i$ és una variable proposicional i $k$ és un valor de veritat pertanyent a $N$.*

**Definició 9** *Una clàusula multivaluada és una disjunció de literals multivaluats i és unitària si conté un únic literal. Una clàusula buida es representa amb $\square$.*

**Definició 10** *Una fórmula multivaluada en forma normal conjuntiva (CNF) és una conjunció de clàusules multivaluades. La fórmula buida es representa per* $\emptyset$.

**Definició 11** *Una interpretació multivaluada assigna un valor de veritat a cada variable proposicional. Una interpretació multivaluada satisfà un literal de la forma* $x_i = k$ *si assigna a* $x_i$ *el valor de veritat* $k$*, i satisfà un literal de la forma* $x_i \neq k$ *si assigna a* $x_i$ *un valor de veritat diferent de* $k$*. Una interpretació multivaluada satisfà una clàusula si satisfà almenys un dels seus literals i satisfà una fórmula CNF multivaluada si satisfà totes les seves clàusules.*

A l'igual que en el cas booleà, l'espai composat per totes les possibles assignacions de valors de veritat a les variables d'una fórmula CNF multivaluada pot ser representat com a un arbre de cerca, on els nodes interns representen assignacions parcials (no s'ha assignat valor a totes les variables) i les fulles representen assignacions complertes.

**Definició 12** *Donada una fórmula* $\phi$ *amb una clàusula unitària, la regla del literal unitari (*one-literal rule*) deriva una clàusula* $\phi'$ *de la següent forma:*

- *Si la clàusula està formada per un literal de la forma* $x_i = k$*, s'elimina de* $\phi$ *totes les clàusules que continguin aquest literal i aquelles que continguin literals de la forma* $x_i \neq l$*, on* $l \neq k$*. Alhora, s'eliminen de les clàusules que els continguin els literals* $x_i \neq k$ *i els literals de la forma* $x_i = l$*, on* $l \neq k$*.*
- *Si la clàusula està formada per un literal de la forma* $x_i \neq k$*, s'elimina de* $\phi$ *totes les clàusules que continguin aquest literal. Alhora, s'elimina el literal* $x_i = k$ *de les clàusules que el continguin.*

**Definició 13** *Donada una fórmula multivaluada* $\phi$*, el problema Max-SAT consisteix en trobar una interpretació que maximitzi el nombre de clàusules de* $\phi$ *satisfetes*

## 3.3 Diferències entre els algorismes booleans i multivaluats

Les diferències entre els algorismes Branch&Bound per als casos booleà i multivaluat es poden trobar en:

- La forma de realitzar la ramificació de l'arbre de cerca. En el cas booleà, la ramificació sempre és binaria, de forma que una branca correspon a l'assignació a cert de la variable i l'altra a l'assignació a fals. En el cas multivaluat, aquesta es pot fer segons tres mètodes:
  - Es fan $N$ branques, fixant $x_i = k, \forall k \in N$. Aquest mètode expandeix l'arbre en tantes branques com valors de veritat existeixen al domini, de forma que cada branca representa l'assignació a la variable d'un valor de veritat diferent. Aquest mètode de ramificació s'anomena *branching n-ari*.
  - Es fan dos branques, una per a $x_i = k$ i l'altra per a $x_i \neq k$. L'anomenarem *branching binari*.
  - Es fan dos branques, una per a $x_i < k$ i l'altra per a $x_i \geq k$. L'anomenarem *branching regular*.
- L'heurística de sel·lecció de variable. En el nostre cas, serà estàtica, donat que els algorismes implementats utilitzen estructures Lazy dissenyades específicament per a ordenacions estàtiques. Dins d'aquest tipus de sel·lecció de variable, implementarem dos mètodes:
  - Ordenació lexicogràfica: Consisteix en instanciar les variables segons el seu ordre lexicogràfic o numèric.
  - Ordenació Most Often: Consisteix en instanciar les variables segons el seu grau d'aparició.
- El mètode de càlcul de la fita inferior. Recordem que la fita inferior es composa de *unsat* més una estimació per sota. Les estimacions a desenvolupar són:
  - **LB1** = *unsat*. No hi ha *understimation*, de forma que la fita inferior correspon al nombre de clàusules no satisfetes per l'assignació parcial actual.
  - **LB2** = $unsat + \sum_{x_i \in \phi'} \text{mín}(\text{ic}(x_i = 1), \text{ic}(x_i = 2), \ldots, \text{ic}(x_i = n))$: on $\phi'$ és la fórmula corresponent a l'assignació parcial actual i $\text{ic}(x_i = k)$ és el nombre de clàusules que es deixen de satisfer si s'estén l'assignació parcial actual amb $x_i = k$, que coincideix amb el nombre de clàusules unitàries de $\phi'$ que contenen $x_i \neq k$ o $x_i = l$ on $l \neq k$.

En aquest PFC, per tant, s'implementaran els següents algorismes:

- Lazy MV-Max-SAT amb branching n-ari i ordenació lexicogràfica: Algorisme branch&bound per a la resolució de Max-SAT multivaluat utilitzant estructures de dades Lazy, ramificació del tipus n-ari i sel·lecció de variables segons una ordenació lexicogràfica.
- Lazy MV-Max-SAT amb branching n-ari i ordenació most often.
- Lazy MV-Max-SAT amb branching binari ($x_i = k, x_i \neq k$) i ordenació lexicogràfica.
- Lazy MV-Max-SAT amb branching binari ($x_i = k, x_i \neq k$) i ordenació most often.
- Lazy MV-Max-SAT amb branching n-ari, ordenació lexicogràfica i *understimation*.
- Lazy MV-Max-SAT amb branching n-ari, ordenació most often i *understimation*.

## 3.4 Representació de fórmules CNF multivaluades

Els resoledors SAT utilitzen un format anomenat DIMACS per a representar fórmules CNF booleanes que llegeixen com a entrada. Un fitxer amb aquest format té l'aspecte que mostra la figura 3.1, on podem observar que:

- Al principi del fitxer pot haver un nombre indefinit de línies de comentaris, que començaran amb la lletra c.
- Seguidament, hi ha una línia de paràmetres, que comença amb la lletra p, seguida del nom del format (cnf), el nombre de variables i el nombre de clàusules que intervenen en el problema.
- Les línies següents descriuen les clàusules. Cada valor indica el número de variable, indicant un zero el final de la definició de la clàusula. Si el número que indica la variable va precedit d'un signe menys, significa que aquesta està negada.

L'exemple de la figura 3.1 correspondria a la fórmula següent:

$$(\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_5) \wedge (\neg x_3 \vee \neg x_3 \vee \neg x_5)$$

```
c Exemple de fitxer .cnf amb 5 variables i 4 clàusules
p cnf 5 4
-1 -2 3 0
-1 -2 4 0
-1 -2 5 0
-3 -4 -5 0
```

Figura 3.1: Representació de fórmules CNF booleanes en format DIMACS

Per a representar fórmules multivaluades, hem de redefinir aquest format per afegir el valor de veritat que s'assigna a la variable, donat que aquest ja no és binari. D'altra banda, s'ha d'afegir als paràmetres de definició del problema informació sobre el domini dels valors de veritat.

El format a utilitzar per a fórmules multivaluades serà el mostrat a la figura 3.2, on podem observar que:

- Al principi del fitxer pot haver un nombre indefinit de línies de comentaris, que començaran amb la lletra c
- Seguidament, hi ha una línia de paràmetres, que comença amb la lletra p, seguida del nom del format (mvcnf), el nombre de variables, el nombre de clàusules que intervenen en el problema i el domini de les variables.
- Les línies següents descriuen les clàusules. El primer valor indica la variable i el segon indica el valor de veritat. Si el número que indica el valor de veritat va precedit d'un signe menys, significa que la variable no pot prendre aquell valor.

L'exemple de la figura 3.2 correspondria a la fórmula següent:

$$(x_1 \neq 1 \vee x_2 = 1) \wedge (x_1 \neq 1 \vee x_2 = 2) \wedge (x_1 \neq 1 \vee x_2 = 3)$$

## 3.5 Generació de problemes

Per a generar fitxers mvcnf per a avaluar els algorismes implementats, utilitzarem un generador uniforme i aleatori de problemes CSP binaris, codificant els problemes resultants amb MV-SAT.

Un problema de satisfacció de restriccions (CSP) es defineix per una tupla $\langle X, D, R \rangle$ on:

```
c Exemple de fitxer .mvcnf multivaluat amb 2 variables
c de domini 3 i 4 clàusules
p mvcnf 2 4 3
1 -1 2 1 0
1 -1 2 2 0
1 -11 2 3 0
```

Figura 3.2: Representació de fórmules CNF multivaluades (MVCNF)

- $X = \{X_1, X_2, \ldots, X_n\}$ és un conjunt finit de variables,
- $D = \{D_1, D_2, \ldots, D_n\}$ és un conjunt finit de dominis i
- $R = \{R_1, R_2, \ldots, R_r\}$ és un conjunt finit de restriccions.

Cada variable $X_i$ pren valors en el seu corresponent domini $D_i$. Una restricció entre un subconjunt de variables $\{X_{i1}, \ldots, X_{ik}\}$ de $X$ és un subconjunt del producte cartesià $D_{i1} \times \cdots \times D_{ik}$ que està format per les tuples de valors que satisfan la restricció, es a dir, les combinacions d'assignacions de valors a variables que no violen la restricció. Una restricció en la que intervé una única variable és una restricció unària, i una restricció en la que intervenen dos variables és una restricció binària. Un CSP binari és un CSP en que únicament es donen restriccions unàries o binàries.

El generador utilitzat ha estat implementat per D. Frost, C. Bessière, R. Dechter, i J.C. Régin, i es troba disponible a http://www.lirmm.fr/∼bessiere /generator.html. El seu format de sortida és el mostrat a la figura 3.3, on podem observar que els dos primers valors, anteriors al símbol dos punts, indiquen entre quines variables es dona la restricció. Les parelles de valors que segueixen són els *nogoods*, aquells valors incompatibles entre les variables, de forma que si la variable x pren el valor a, la variable y no pot prendre el valor b.

```
x y: (a b) (c d)
x z: (e f) (g h)
```

Figura 3.3: Format de representació de CSP

La transformació es realitza agefint, per a cada *nogood* x y: (a b), una clàusula de la forma

$$x_i \neq a \vee x_j \neq b$$

ja que:

**Input: max-sat**($\phi$,x,ub) : A number of variable $x$, a boolean CNF formula $\phi$ and an upper bound ub
1: **if** $\phi = \emptyset$ or $\phi$ only contains empty clauses **then**
2: return **empty-clauses**($\phi$)
3: **endif**
4: $p \leftarrow$ x + 1
5: **foreach** v:value of domain
6: $\phi \leftarrow$ one-literal-rule($\phi, p = v$)
8: **if** unsat($\phi$) $\geq$ ub **then**
9: return $\infty$
10: **endif**
11: $ub = min$(ub,max-sat($\phi, p, ub$))
10: **endforeach**
11: **Output:** The maximum number of clauses of $\phi$ than can be satisfied

Figura 3.4: Esquelet algorisme Lazy MV-MAX-Sat amb ordenació lexicogràfica i branching n-ari

- x y: (a b) és equivalent a $x = a \rightarrow y \neq b$
- $x = a \rightarrow y \neq b$ és equivalent a $x \neq a \vee y \neq b$

## 3.6 Algorisme LazyMVMaxSatLexNari

L'algorisme LazyMVMaxSatLexNari és un algorisme de resolució de problemes Max-SAT multivaluats mitjançant utilització d'estructures Lazy, ordenació lexicogràfica de les variables i branching N-ari. El seu esquelet es mostra a la figura 3.4

Els literals de les clàusules de la fórmula estan en ordre lexicogràfic ascendent, es a dir, en primer lloc es troben els de menor ordre i en últim els de major ordre. Per a cada clàusula, es manté un apuntador al seu literal de major ordre.

Per a cada literal, es manté una llista de clàusules on aquest és el literal de major ordre. D'aquesta forma, processem únicament aquelles clàusules en que han estat instanciades totes les variables que hi ocorren.

L'algorisme efectua els següents passos:

1. Llegir la fórmula d'un fitxer MV-CNF.
2. Ordenar les seves clàusules en ordre lexicogràfic ascendent i enregistrar quin és el literal de major ordre.

3. Enregistrar, per a cada literal, en quines clàusules aquest literal és el de major ordre.

4. Instanciar les variables per ordre ascendent amb els valors del domini també per ordre ascendent.

5. Aplicar la regla del literal unitari pel literal instanciat, obtenint el nombre de clàusules no satisfetes per aquella instanciació, es a dir, el nombre de clàusules que esdevenen buides.

6. Si aquest nombre és inferior al *upper bound*, cridar a l'algorisme amb la nova fórmula, de la que s'han eliminat totes les clàusules que han esdevingut buides o satisfetes.

7. Si el valor que retorna aquesta crida recursiva és inferior al *upper bound*, actualitzar-lo a aquest nou valor

8. Tornar a incloure les clàusules eliminades de la fórmula i retornar.

Cal destacar que la avantatge de les estructures Lazy és que per incloure de nou les clàusules eliminades de la fórmula únicament cal fer una suma.

## 3.7 Algorisme LazyMVMaxSatMONari

L'algorisme LazyMVMaxSatMONari és un algorisme de resolució de problemes Max-SAT multivaluats mitjançant utilització d'estructures Lazy, ordenació most often de les variables i branching N-ari. El seu esquelet es mostra a la figura 3.5

La funció $select - variable(\phi)$ retorna la variable següent en l'ordre most often determinat.

Els literals de les clàusules de la fórmula estan ordenats segons la freqüència d'aparició de la variable. Per a cada clàusula, es manté un apuntador al seu literal de major ordre.

Per a cada literal, es manté una llista de clàusules on aquest és el literal de major ordre. D'aquesta forma, processem únicament aquelles clàusules en que han estat instanciades totes les variables que hi ocorren.

L'algorisme efectua els següents passos:

1. Llegir la fórmula d'un fitxer MV-CNF.

2. Fer un recompte d'aparicions de cada literal de la fórmula, agrupats per variables.

**Input:** **max-sat**($\phi$,x,ub) : A number of variable $x$, a boolean CNF formula $\phi$ and an upper bound ub
1: **if** $\phi = \emptyset$ or $\phi$ only contains empty clauses **then**
2: return **empty-clauses**($\phi$)
3: **endif**
4: $p \leftarrow$ select-variable($\phi$)
5: **foreach** v:value of domain
6: $\phi \leftarrow$ one-literal-rule($\phi, p = v$)
8: **if** unsat($\phi$) $\geq$ ub **then**
9: return $\infty$
10: **endif**
11: $ub = min$(ub,max-sat($\phi, p, ub$))
10: **endforeach**
11: **Output:** The maximum number of clauses of $\phi$ than can be satisfied

Figura 3.5: Esquelet algorisme Lazy MV-MAX-Sat amb ordenació most often i branching n-ari

3. Ordenar les variables segons la suma d'aparicions dels seus literals, en l'ordre escollit (ascendent o descendent).

4. Ordenar les clàusules en l'ordre determinat pels passos anteriors, enregistrant quin és el literal de major ordre.

5. Enregistrar, per a cada literal, en quines clàusules aquest literal és el de major ordre.

6. Instanciar les variables per l'ordre determinat en el pas 4, assignant-los-hi valors del domini en l'ordre determinat en el pas 3.

7. Aplicar la regla del literal unitari pel literal instanciat, obtenint el nombre de clàusules no satisfetes per aquella instanciació, es a dir, el nombre de clàusules que esdevenen buides.

8. Si aquest nombre és inferior al *upper bound*, cridar a l'algorisme amb la nova fórmula, de la que s'han eliminat totes les clàusules que han esdevingut buides o satisfetes.

9. Si el valor que retorna aquesta crida recursiva és inferior al *upper bound*, actualitzar-lo a aquest nou valor.

10. Tornar a incloure les clàusules eliminades de la fórmula i retornar.

Veiem un exemple de funcionament:

- Suposem que la fórmula llegida en el pas 1 conté les següents clàusules:

$$x_1 \neq 1 \vee x_2 \neq 2$$

$$x_1 \neq 2 \vee x_2 \neq 1$$

$$x_1 \neq 1 \vee x_3 \neq 1$$

- Fem un recompte d'aparicions de les variables:
  - aparicions($x_1$) = 3 (aparicions($x_1 \neq 1$) = 2 + aparicions($x_1 \neq 2$) = 1)
  - aparicions($x_2$) = 2 (aparicions($x_2 \neq 1$) = 1 + aparicions($x_2 \neq 2$) = 1)
  - aparicions($x_3$) =1 (aparicions($x_3 \neq 1$) = 1)
- Ordenem les variables de forma ascendent (descendent) segons el nombre d'aparicions
  - $x_3$,$x_2$,$x_1$
- Ordenem les clàusules per l'ordre determinat en el pas 4

$$x_2 \neq 2 \vee x_1 \neq 1$$

  Literal de major ordre: $x_1 \neq 1$

$$x_2 \neq 1 \vee x_1 \neq 2$$

  Literal de major ordre: $x_1 \neq 2$

$$x_3 \neq 1 \vee x_1 \neq 1$$

  Literal de major ordre: $x_1 \neq 1$

- Ordenem les variables a instanciar per a cada clàusula
  - Primer instanciarem $x_3$
  - Després instanciarem $x_2$
  - Finalment instanciarem $x_1$
- Instanciem les variables en l'ordre determinat, començant pel primer valor del domini i acabant per l'últim

En aquest TFC s'avaluaran les diferents alternatives d'ordenació i s'analitzaran els resultats.

**Input: max-sat**($\phi$,l,ub) : A literal $l$, a boolean CNF formula $\phi$
and an upper bound ub
1: **if** $\phi = \emptyset$ or $\phi$ only contains empty clauses **then**
2: return **empty-clauses**($\phi$)
3: **endif**
4: $l \leftarrow$ select-literal($\phi$,l)
5: $\phi \leftarrow$ one-literal-rule($\phi, l$)
8: **if** unsat($\phi$) $\geq$ ub **then**
9: return $\infty$
10: **endif**
11: $ub = min$(ub,max-sat($\phi, l, ub$))
4: $l \leftarrow \bar{l}$
5: $\phi \leftarrow$ one-literal-rule($\phi, l$)
8: **if** unsat($\phi$) $\geq$ ub **then**
9: return $\infty$
10: **endif**
11: $ub = min$(ub,max-sat($\phi, l, ub$))
11: **Output:** The maximum number of clauses of $\phi$ than can be satisfied

Figura 3.6: Esquelet algorisme Lazy MV-MAX-Sat amb ordenació lexicogràfica i branching binari

## 3.8 Algorisme LazyMVMaxSatLexBinari

L'algorisme LazyMVMaxSatLexBinari és un algorisme de resolució de problemes Max-SAT multivaluats mitjançant utilització d'estructures Lazy, ordenació lexicogràfica de les variables i branching binari. El seu esquelet es mostra a la figura 3.6

La funció $select - literal(\phi, l)$ es comporta de forma diferent segons el sentit del literal que rep com a paràmetre. Si aquest literal és positiu ($x_i = k$) retorna la següent variable instanciada al primer valor del domini. Si és negatiu ($x_i \neq k$) retorna el següent valor de la mateixa variable, o el mateix que en el cas positiu si la variable ja ha pres tots els possibles valors del seu domini.

Amb aquesta consideració, l'algorisme efectua els següents passos:

1. Llegir la fórmula d'un fitxer MV-CNF.

2. Ordenar les seves clàusules en ordre lexicografic ascendent i enregistrar quin és el literal de major ordre.

3. Enregistrar, per a cada literal, en quines clàusules aquest literal és el de major ordre.

4. Cridar a la funció select-literal, que instancia les variables per ordre ascendent amb els valors del domini també per ordre ascendent, tenint en compte que si el literal instanciat anteriorment era positiu ($x_i = k$) haurà d'instanciar la següent variable ($x_{i+1} = 1$), mentre que si rep un literal negatiu ($x_i \neq k$), haurà d'instanciar la mateixa variable amb el següent valor del domini ($x_i = k + 1$).

5. Aplicar la regla del literal unitari pel literal instanciat, obtenint el nombre de clàusules no satisfetes per aquella instanciació, es a dir, el nombre de clàusules que esdevenen buides.

6. Si aquest nombre és inferior al *upper bound*, cridar a l'algorisme amb la nova fórmula, de la que s'han eliminat totes les clàusules que han esdevingut buides o satisfetes.

7. Si el valor que retorna aquesta crida recursiva és inferior al *upper bound*, actualitzar-lo a aquest nou valor.

8. Tornar a incloure les clàusules eliminades de la fórmula.

9. Canviar el sentit del literal instanciat.

10. Aplicar la regla del literal unitari pel literal instanciat, obtenint el nombre de clàusules no satisfetes per aquella instanciació, es a dir, el nombre de clàusules que esdevenen buides.

11. Si aquest nombre és inferior al *upper bound*, cridar a l'algorisme amb la nova fórmula, de la que s'han eliminat totes les clàusules que han esdevingut buides o satisfetes.

12. Si el valor que retorna aquesta crida recursiva és inferior al *upper bound*, actualitzar-lo a aquest nou valor.

13. Tornar a incloure les clàusules eliminades de la fórmula i retornar.

## 3.9 Algorisme LazyMVMaxSatMoBinari

L'algorisme LazyMVMaxSatMoBinari és un algorisme de resolució de problemes Max-SAT multivaluats mitjançant utilització d'estructures Lazy, ordenació most often de les variables i branching binari.

L'algorisme actua de forma similar a l'anterior, tenint en compte que en aquest cas la funció select-literal té en compte l'ordre d'aparició calculat en la fase de preprocessament de la fórmula.

## 3.10 Algorisme LazyMVMaxSatLexNariUS

L'algorisme LazyMVMaxSatLexNari és un algorisme de resolució de problemes Max-SAT multivaluats mitjançant utilització d'estructures Lazy, ordenació lexicogràfica de les variables i branching N-ari, aplicant understimation.

El seu funcionament és molt similar al funcionament de l'algorisme LazyMV-MaxSatLexNari, amb la diferència que s'aplica una estimació que implica modificar les estructures Lazy amb les que es representa el problema. Recordem que l'estimació aplicada és la suma de mínimes inconsistències de les variables de la fórmula. Aquesta estimació fa un recompte de clàusules unitàries que es deixen de satisfer si s'instancia una variable amb cadascun dels valors del seu domini, i calcula el mínim. Finalment calcula la suma dels mínims per a cada variable del sistema.

Per a fer aquest càlculs, cal disposar de clàusules unitàries. Per aquest motiu l'estructura Lazy que representa el problema ha de passar a emmagatzemar, per a cada literal, apuntadors a les clàusules on aquest literal és el *penúltim* de màxim ordre, en comptes d'emmagatzemar el de màxim ordre com en la resta d'algorismes.

D'aquesta forma, retardem l'avaluació d'una clàusula fins el moment en que s'ha instanciat el seu literal de penúltim ordre, moment en que de l'avaluació de la clàusula resulta la clàusula buida (no satisfactible) o una clàusula unitària.

L'algorisme, per tant, efectua els següents passos:

1. Llegir la fórmula d'un fitxer MV-CNF.

2. Ordenar les seves clàusules en ordre lexicogràfic ascendent i enregistrar quin és el penúltim literal en aquest ordre.

3. Enregistrar, per a cada literal, en quines clàusules aquest literal és el penúltim en ordre.

4. Instanciar les variables per ordre ascendent amb els valors del domini també per ordre ascendent.

5. Aplicar la regla del literal unitari pel literal instanciat, obtenint el nombre de clàusules eliminades per aquella instanciació, es a dir, el nombre

de clàusules que esdevenen satisfetes, així com les clàusules que esdevenen unitàries, es a dir, que deixen de satisfer-se avaluant-les fins el penúltim literal.

6. Fer el càlcul de l'estimació. Per a cada variable, calcular el mínim de clàusules que es deixen de satisfer si instanciem la variable amb cadascun dels valors del domini. Després suma els mínims de cada variable.

7. Si la suma del nombre de clàusules que esdevenen buides i la estimació és inferior al *upper bound*, cridar a l'algorisme amb la nova fórmula, de la que s'han eliminat totes les clàusules que han esdevingut buides o satisfetes.

8. Si el valor que retorna aquesta crida recursiva és inferior al *upper bound*, actualitzar-lo a aquest nou valor.

9. Tornar a incloure les clàusules eliminades de la fórmula i retornar.

## 3.11 Algorisme LazyMVMaxSatMONariUS

L'algorisme LazyMVMaxSatMONari és un algorisme de resolució de problemes Max-SAT multivaluats mitjançant utilització d'estructures Lazy, ordenació most often de les variables i branching N-ari, aplicant understimation.

El seu funcionament es pot deduir fàcilment de la informació donada per l'algorisme LazyMVMaxSatMONari i per l'algorisme LazyMVMaxSatLexNariUS, i es pot resumir en:

1. Llegir la fórmula d'un fitxer MV-CNF.

2. Fer un recompte d'aparicions de cada literal de la fórmula, agrupats per variables.

3. Ordenar les variables segons la suma d'aparicions dels seus literals, en l'ordre escollit.

4. Ordenar les clàusules en l'ordre determinat pels passos anteriors, enregistrant quin és el penúltim literal en aquest ordre .

5. Enregistrar, per a cada literal, en quines clàusules aquest literal és el penúltim en ordre.

6. Instanciar les variables per l'ordre determinat en el pas 3, assignant-los-hi valors del domini en ordre ascendent.

7. Aplicar la regla del literal unitari pel literal instanciat, obtenint el nombre de clàusules eliminades per aquella instanciació, es a dir, el nombre de clàusules que esdevenen satisfetes, així com les clàusules que esdevenen unitàries.

8. Fer el càlcul de l'estimació. Per a cada variable, calcular el mínim de clàusules que es deixen de satisfer si instanciem la variable amb cadascun dels valors del domini. Després suma els mínims de cada variable.

9. Si la suma del nombre de clàusules que esdevenen buides i la estimació és inferior al *upper bound*, cridar a l'algorisme amb la nova fórmula, de la que s'han eliminat totes les clàusules que han esdevingut buides o satisfetes.

10. Si el valor que retorna aquesta crida recursiva és inferior al *upper bound*, actualitzar-lo a aquest nou valor.

11. Tornar a incloure les clàusules eliminades de la fórmula i retornar.

# 4

# Avaluació experimental

## 4.1 Introducció

L'avaluació experimental dels algorismes implementats s'ha portat a terme amb instàncies de CSP binaris generats aleatòriament. El generador d'instàncies pren com a paràmetres la tupla $\langle n, d, p_1, t\rangle$ on $n$ és el nombre de variables, $d$ és la cardinalitat del domini, $p_1$ és el ratio de restriccions existents i $t$ és el ratio de parells de valors no permesos (*nogood*). Les variables de les restriccions i els parells de valors no permesos s'escullen de forma aleatòria i uniforme segons l'anomenat model B[12] de generació de CSP's.

Tots els experiments han estat realitzats en ordinadors Pentium IV a 3GHz amb 512MB de RAM. Per a cada tipus de problema , valor del paràmetre $t$ i algorisme s'han generat mostres amb 100 instàncies.

Analitzarem el comportament dels diferent algorismes a mesura que augmentem el paràmetre $t$. Quan aquest valor és baix, hi ha molts valors permesos a les restriccions, el nombre de clàusules no satisfetes és baix i el problema és fàcil de resoldre. A mesura que aquest valor augmenta, augmenta el nombre de clàusules no satisfetes i la dificultat del problema.

## 4.2 Experiment 1

En aquest experiment mostrem, primer per separat i finalment en conjunt, el comportament dels algorismes implementats per a resoldre la instància $\langle 10, 5, 45, t\rangle$, on $t$ pot variar entre 1 i 24.

Es mostren a continuació les gràfiques de temps corresponents als sis algorismes implementats

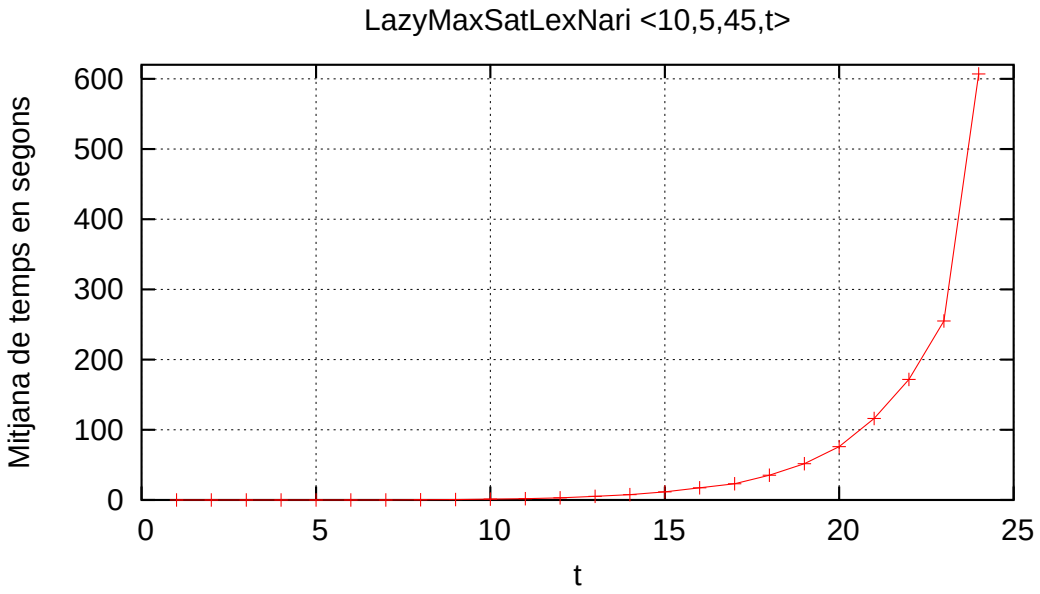


Figura 4.1: Temps de l'algorisme LazyMaxSatLexNari per a $\langle 10, 5, 45, t \rangle$

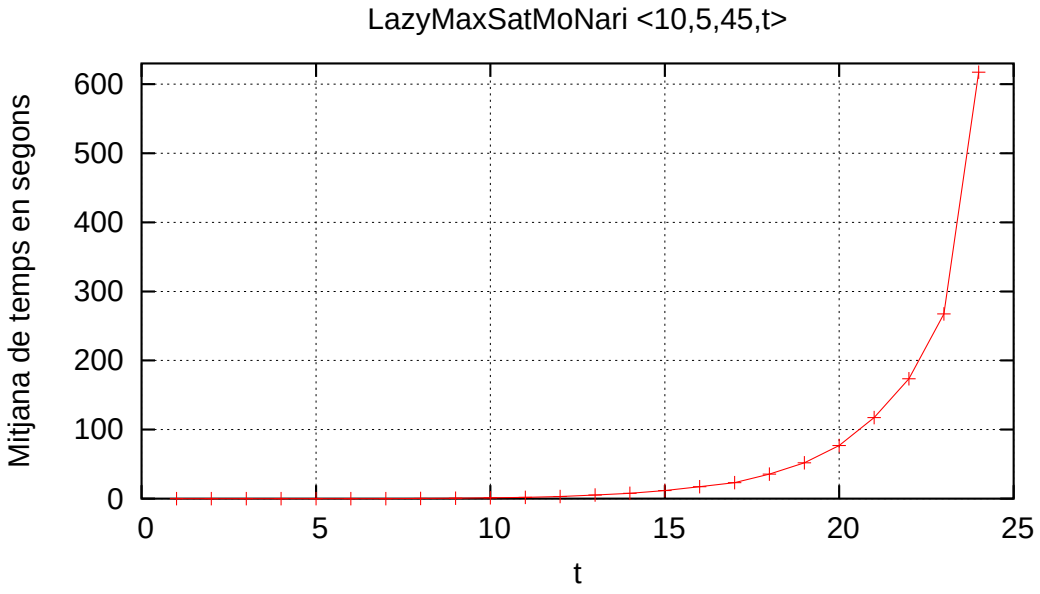


Figura 4.2: Temps de l'algorisme LazyMaxSatMoNari per a $\langle 10, 5, 45, t \rangle$

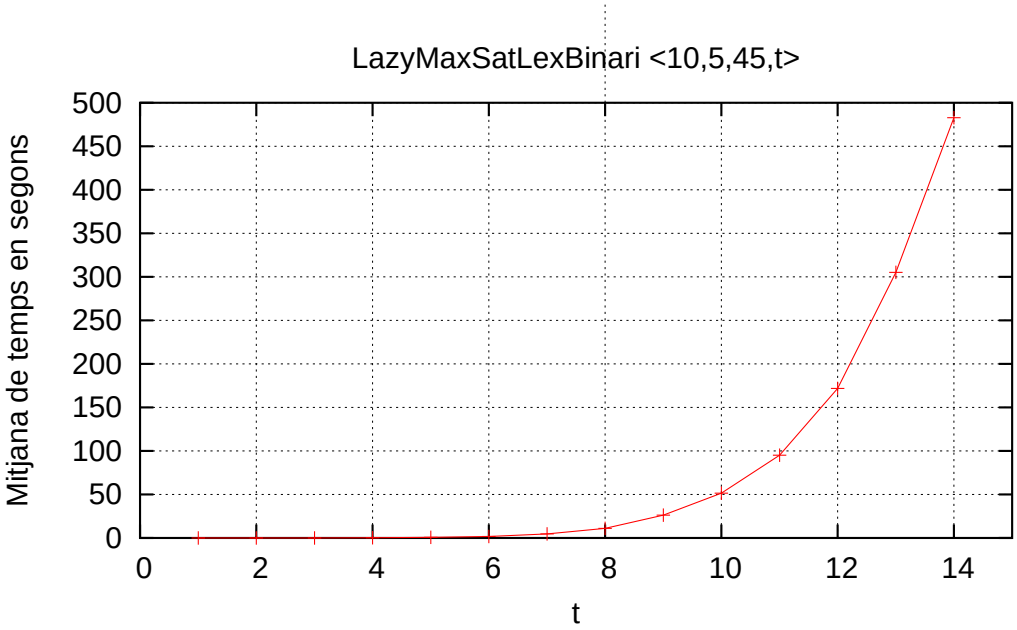


Figura 4.3: Temps de l'algorisme LazyMaxSatLexBinari per a $\langle 10, 5, 45, t \rangle$

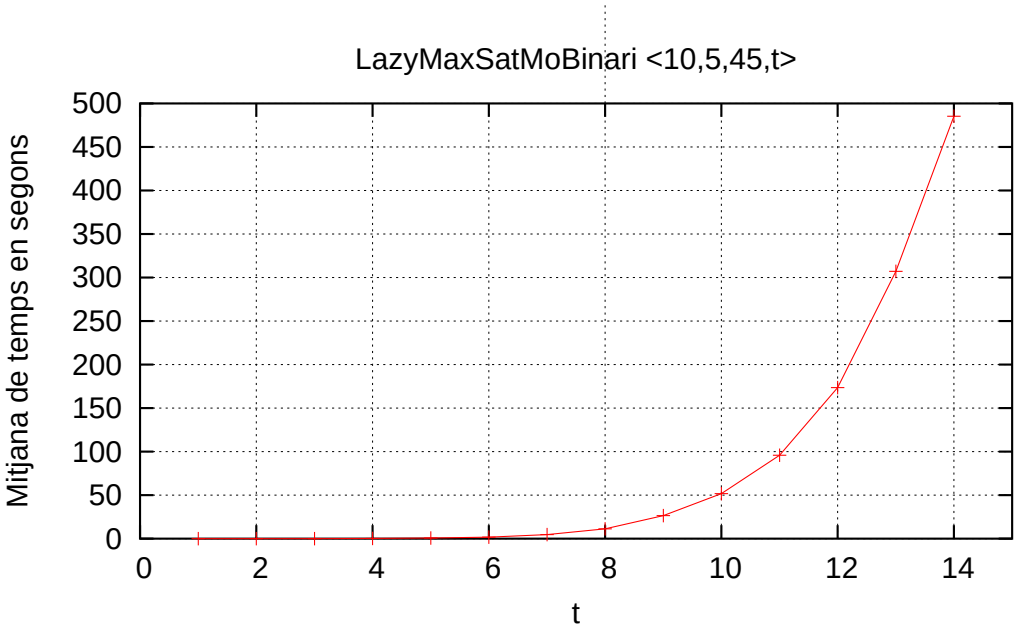


Figura 4.4: Temps de l'algorisme LazyMaxSatMoBinari per a $\langle 10, 5, 45, t \rangle$

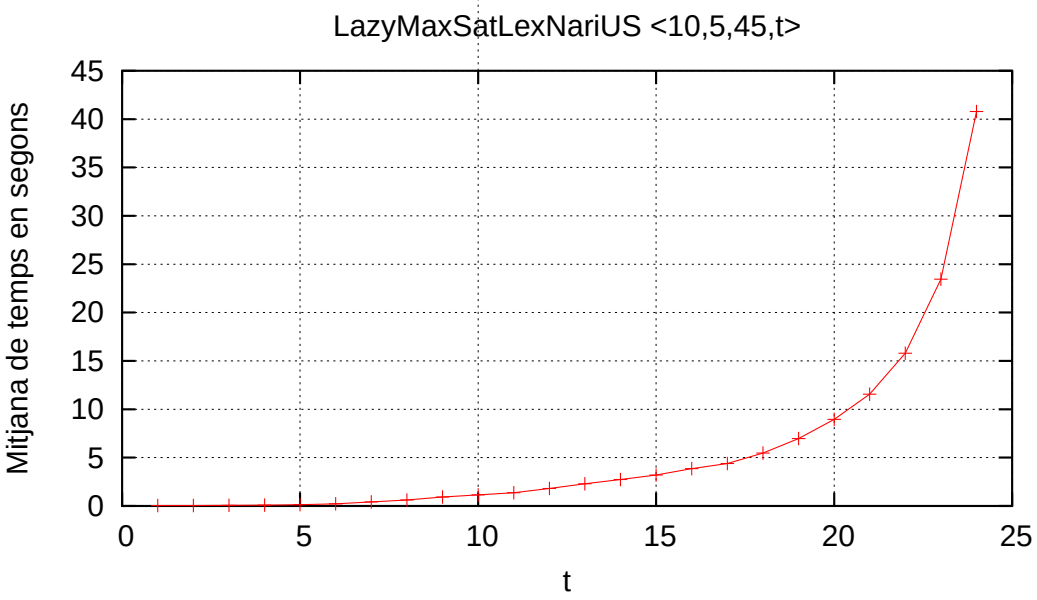


Figura 4.5: Temps de l'algorisme LazyMaxSatLexNariUS per a $\langle 10, 5, 45, t \rangle$

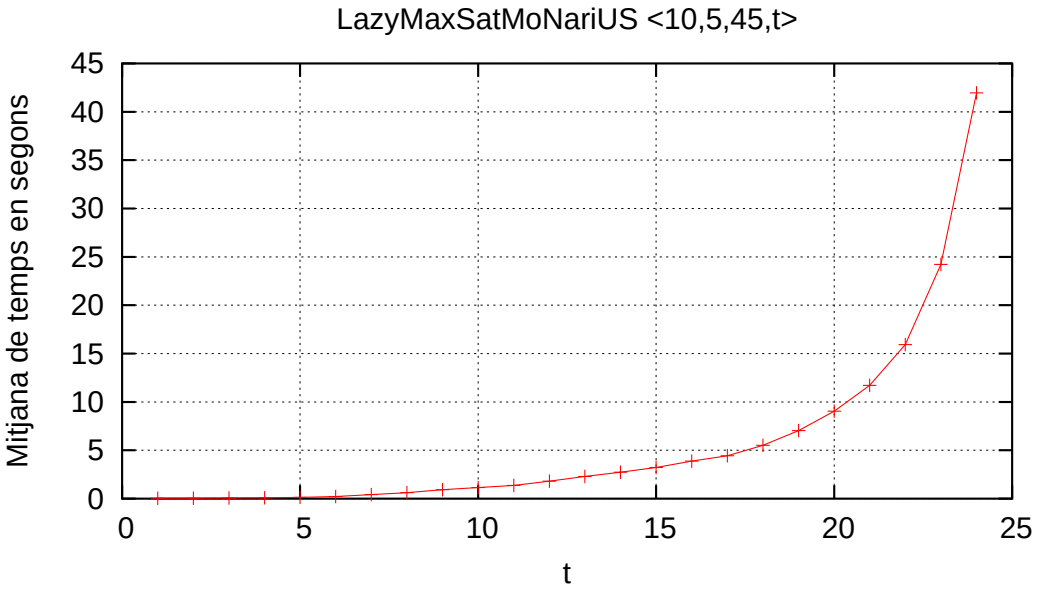


Figura 4.6: Temps de l'algorisme LazyMaxSatMoNariUS per a $\langle 10, 5, 45, t \rangle$

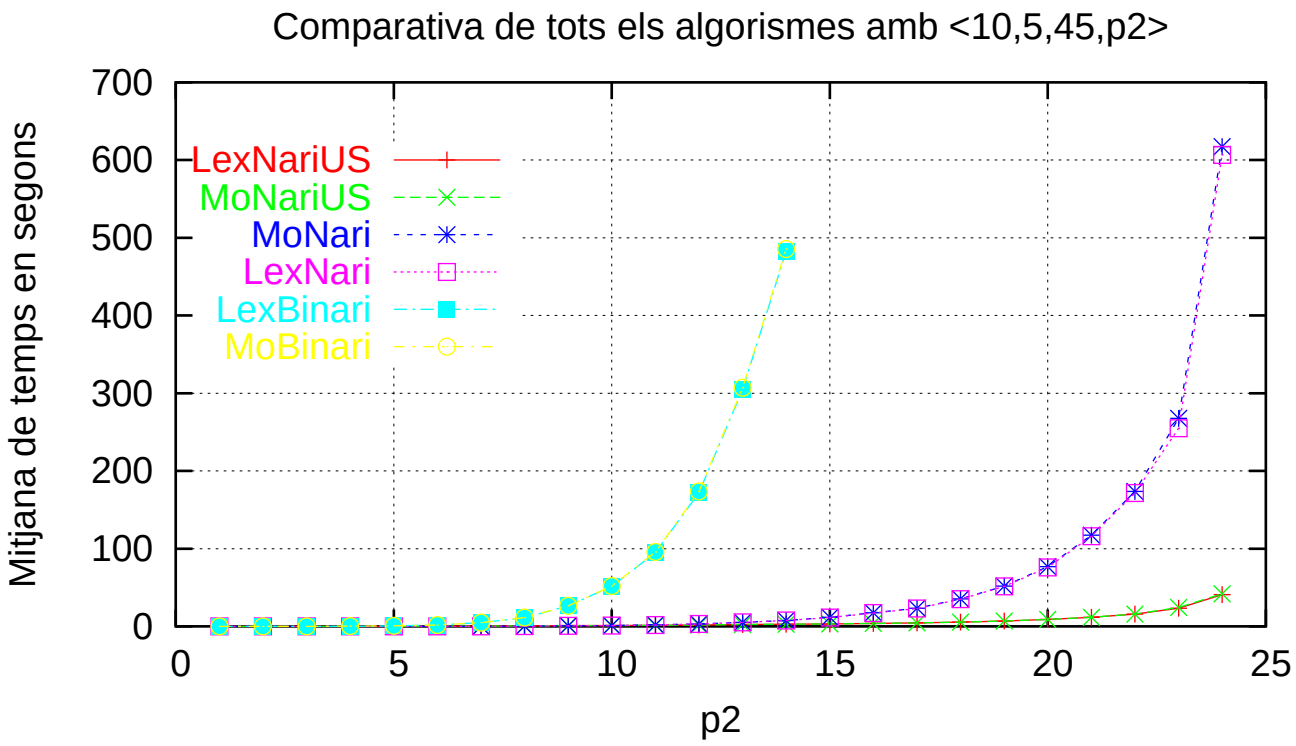


Figura 4.7: Comparativa de tots els algorismes per a $\langle 10, 5, 45, t \rangle$

L'observació de la gràfica comparativa de la figura 4.7 ens permet extreure diverses conclusions:

1. Com era d'esperar, els algorismes amb estimació superen en molt el rendiment dels algorismes que no la incorporen.

2. Contràriament al que dicta la lògica i les experiències booleanes, els algorismes que tenen en compte la freqüència d'aparició de les variables no superen en rendiment, en els problemes generats en aqust experiment, als d'ordenació lexicogràfica. Estudiant detingudament els problemes generats, e observat que aquests són molt uniformes, de forma que les diferències en el nombre d'aparicions de les diferents variables i dels diferents valors d'una mateixa variable són quasi nul·les.i D'aquesta forma, l'ordenació únicament suposa un temps extra de processament que no aporta cap millora en el rendiment.

3. Els algorismes binaris són els que tenen pitjor rendiment. El branching binari és natural als problemes codificats amb logica booleana, pero no en els casos multivaluats, donat que instanciar la variable amb negatiu ens proporciona menys propagació. En el capítol 3 hem explicat el funcionament de la regla del literal unitari per a fórmules multivaluades, i hem vist que en el cas que el literal sigui negatiu, s'eliminen menys clàusules que en el cas que sigui positiu.

## 4.3 Experiment 2

En aquest experiment tractarem de veure quina ordenació és òptima per a l'algorisme amb heurística de sel·lecció de variable per ordre d'aparició. Hi ha dues possibilitats:

- Ordenació ascendent: Instanciem en primer lloc les variables que ocorren menys vegades a la fórmula, acabant per instanciar les que més ocorren

- Ordenació descendent: Instanciem en primer lloc les variables que ocorren més vegades

En la figura 4.8 podem veure una comparativa de les dos ordenacions en el problema $\langle 12, 4, 50, t\rangle$, on $t$ varia entre 1 i 14.

D'aquest experiment es pot concloure que la millor ordenació és la descendent, com calia esperar, donat que instanciar en primer lloc les variables que mes ocorren a la fórmula ens aporta més propagació.

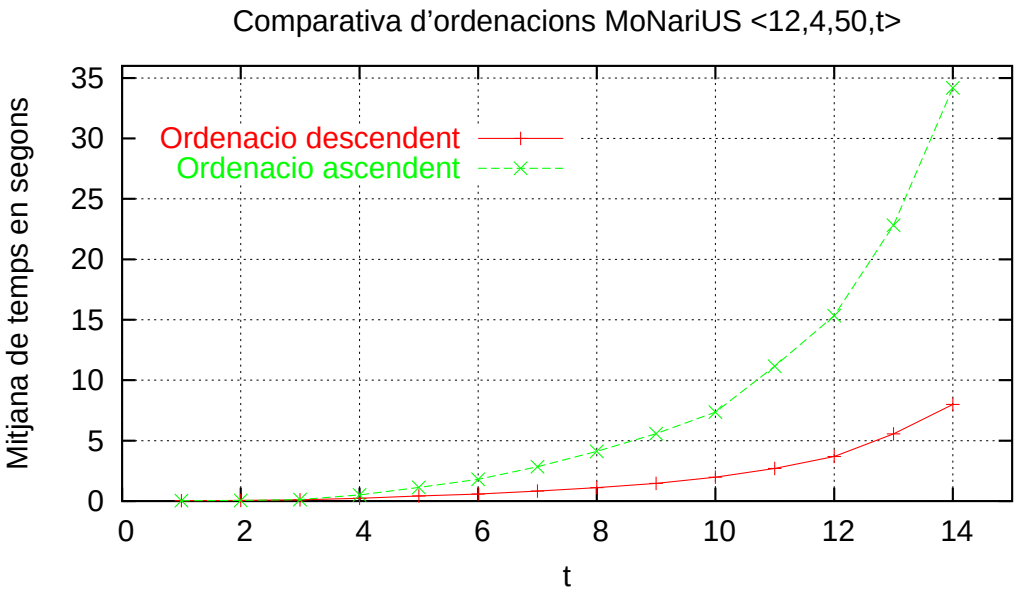


Figura 4.8: Comparativa d'ordenacions MoNariUS per a $\langle 12, 4, 50, t \rangle$

## 4.4 Experiment 3

En aquest experiment es pretén mostrar la diferència entre els algorismes amb ordenació lexicogràfica i amb ordenació most often per a un problema no uniforme, es a dir, on la diferència entre el nombre d'aparicions de les variables és considerable.

Per aconseguir un problema d'aquest tipus, haurem de generar un CSP amb poques restriccions. D'aquesta forma, no totes les variables apareixeran a les restriccions, de la mateixa forma que no ho faran tots els valors. El problema de l'experiment 1 té totes les possibles restriccions que permet el nombre de variables. Si $n$ és el nombre de variables, el nombre de restriccions serà com a màxim $\frac{n(n-1)}{2}$.

Hem generat el problema $\langle 12, 4, 50, t \rangle$, amb $t$ variant entre 1 i 14. El màxim nombre de restriccions d'aquest problema és 66, i únicament s'han generat 50.

En la figura 4.9 podem veure que l'algorisme amb ordenació per nombre d'aparicions és, en aquest cas, més ràpid

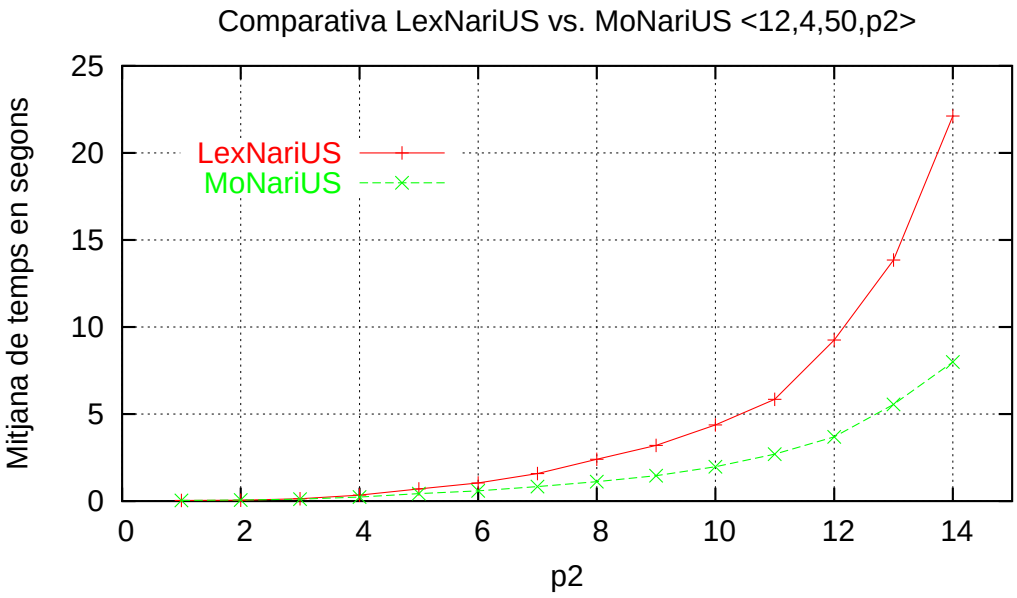


Figura 4.9: Comparativa LexNariUS vs. MoNariUS per a $\langle 12, 4, 50, t \rangle$

## 4.5 Experiment 4

En aquest experiment volem mostrar el creixement de la diferència entre els algorismes d'ordenació lexicogràfica i els d'ordenació per ordre d'aparició en problemes poc uniformes. Per a fer-ho em generat un problema que únicament genera aproximadament meitat de les restriccions possibles, tot i ser un problema força complexe i costós de resoldre.

El problema generat és el $\langle 15, 4, 70, t \rangle$. Aquest problema podria tenir un màxim de 105 restriccions, de les que únicament en generem 70.

A la figura 4.10 podem veure que l'algorisme amb ordenació lexicogràfica té un rendiment força pitjor que l'algorisme amb ordenació per ordre d'aparició, que a més escala millor.

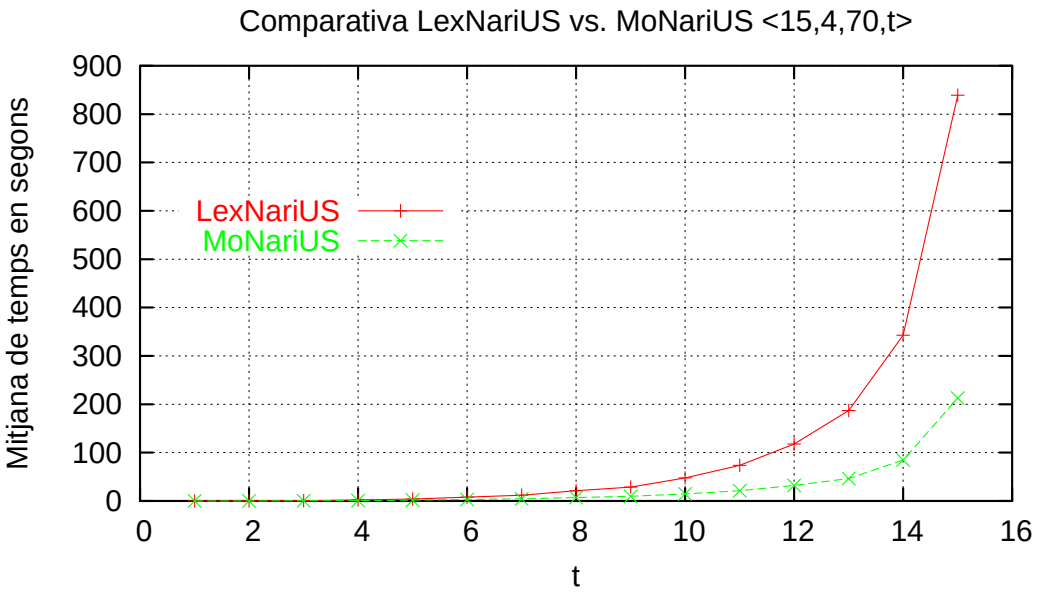


Figura 4.10: Comparativa LexNariUS vs. MoNariUS per a $\langle 15, 4, 70, t \rangle$

# 5

# Conclusions i treballs futurs

En aquest treball final de carrera hem estudiat la resolució del problema Max-SAT per a fórmules CNF multivaluades. Per a fer-ho, hem partit de l'estudi dels resoledors existents per a fórmules CNF booleanes i hem hagut de:

- Definir un format multivaluat per a fórmules CNF, partint del format CNF booleà.
- Estudiar la forma de generar fórmules CNF multivaluades per avaluar els algorismes a implementar, optant per la codificació multivaluada de instàncies CSP obtingudes d'un generador aleatori de lliure distribució.
- Implementar aquest codificador.
- Estudiar el funcionament de les estructures de dades Lazy i la seva incorporació als algorismes de resolució de Max-SAT multivaluat.
- Dissenyar i implementar un algorisme bàsic de resolució del problema Max-SAT multivaluat a partir de l'algorisme de Borchers&Furman, incorporant estructures de dades Lazy i utilitzant ordenació lexicogràfica.
- Definir i implementar noves heurístiques de sel·lecció de variable, basades en la freqüència d'aparició.
- Implementar una variant de l'algorisme amb branching binari per estudiar la seva utilitat en la resolució de problemes.
- Estudiar i implementar tècniques d'estimació per a obtenir millors cotes inferiors.
- Realitzar una avaluació experimental dels algorismes implementats.

De forma resumida, podem concloure que:

- Les estructures de dades Lazy tenen una aplicació directa en els algorismes Max-SAT multivaluat.
- Els millors algorismes per a resoldre el problema max-sat multivaluat són aquells que incorporen estimacions per sota.
- Els algorismes de branching n-ari superen clarament en rendiment als de branching binari.
- Els algorismes amb heurístiques de sel·lecció de variable basades en el nombre d'aparicions superen amb claredad als algorismes lexicogràfics en la resolució de problemes amb estructura, es a dir, no generats aleatòriament. Els algorismesi lexicogràfics poden ser utilitzats, per la seva simplicitat d'implementació, en problemes molt uniformes i poc costosos.

Com a futures línies de treball futur apuntem les següents:

- Realitzar una avaluació experimental dels algorismes desenvolupats amb problemes amb estructura, es a dir, no generats aleatòria i uniformement, sinó partint de codificacions amb lògica multivaluada de problemes reals.
- Implementar els algorismes més competitius en un llenguatge de programació compilat com C++, per tal de millorar el seu rendiment. Com a punt de partida caldria prendre l'algorisme LazyMaxSatMoNariUS, es a dir, l'algorisme amb estructures lazy, ordenació most often, branching n-ari i amb understimation.
- Experimentar l'assignació de valors a variables ordenada per nombre d'aparicions.
- Experimentar el branching regular ($x_i \leq k, x_i > k$).
- Estudiar la incorporació d'estructures de dades Lazy que no requereixin ordenació estàtica de variables, per tal de poder implementar heurístiques de sel·lecció de variable més avançades.
- Estudiar la incorporació de cotes inferiors de major qualitat a partir de les existents per a problemes booleans o definir-ne de noves.
- Optimitzar les implementacions fetes dels algorismes.

# Bibliografia


[1] Alsinet, T., Manya, F., and Planes, J. Improved branch and bound algorithms for max-sat. In *SAT Conference* (2003).

[2] Bayardo, R. J., and Schrag, R. C. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA* (1997), AAAI Press, pp. 203–208.

[3] Borchers, B., and Furman, J. A two fase exact algorithm for max-sat and weighted max-sat problems. *Journal of Combinatorial Optimization 2* (1999), 299–306.

[4] Davis, M., and Putnam, H. A computing procedure for quantification theory. *Journal of the ACM 7*, 3 (1960), 201–215.

[5] Jeroslow, R. J., and Wang, J. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence 1* (1990), 167–188.

[6] Li, C. M., and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'97, Nagoya, Japan* (1997), Morgan Kaufmann, pp. 366–371.

[7] Loveland, D. W. Automated theorem proving. a logical basis. *Fundamental studies in computer science 6* (1978).

[8] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)* (2001).

[9] Richard J Wallace, E. C. F. Comparative studies of constraint satisfaction and davis-putnam algorithms for maximum satisfiability problems.

[10] Selman, B., Levesque, H., and Mitchell, D. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI'92, San Jose/CA, USA* (1992), AAAI Press, pp. 440–446.

[11] Silva, J. P. M., and Sakallah, K. A. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers 48*, 5 (1999), 506–521.

[12] Smith, B., and Dyer, M. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence 81* (1996), 155–181.