Generador d'Horaris d'Instituts

Cristòfor Nogueira Gascons

14 de juny de 2015

Índex

1 Introducció

La realització d'horaris d'instituts és un problema recorrent amb què es troben la majoria d'instituts. L'alta combinatòria que amaguen en dificulta molt la seva confecció de manera manual ja que a causa del munt de decisions cegues que s'han de prendre durant la seva realització és molt probable que es cometin errors. Amb l'aparició dels computadors, molts instituts van relegar aquesta feina a eines informàtiques capaces d'explorar centenars de milers de combinacions per segon. Tot i així, aquelles primeres eines no semblaven tenir millor fortuna ja que el nombre de combinacions possibles rere un problema d'aquestes característiques és exponencial al nombre d'assignatures. Per tant, també es tenien moltíssimes dificultats per generar horaris viables. Aquests horaris generalment eren de baixa qualitat i requerien d'un refinament manual posterior.

Un altre aspecte que afegeix complexitat al problema és la gran varietat de requeriments dels horaris, que varien d'un país a un altre. En particular dificulta l'abstracció necessària a l'hora de modelar una eina capaç de generar horaris de manera satisfactòria per instituts d'arreu. Aquest fet fa difícil la creació de generadors capaços d'explotar les peculiaritats inherents a cada instància concreta.

Encara avui dia, no es coneix cap eina capaç de resoldre el problema de manera determinista i en un temps raonable. Però l'estat de l'art ha avançat moltíssim i ja són pocs els instituts que opten per atacar el problema de manera manual.

Els objectius d'aquest treball són per una banda, aprofundir sobre el problema de la confecció d'horaris, veure com de dur és el problema, què és el que el fa tan dur i aprofundir en les tècniques deterministes de més èxit a l'actualitat. I per altra banda, implementar un generador determinista capaç de generar horaris per a instàncies de formulació específica utilitzant dues de les tecnologies amb més èxit i futur en el sí de l'estat de l'art actual, com són tècniques basades en SAT (Satisfactibilitat proposicional booleana) i tècniques basades en SMT (Satisfiability Modulo Theories).

1.1 Estructura del treball

A fi de complir els objectius marcats hem dividit el treball en tres parts:

1. Estudi de la duresa del problema (Secció 4)

   En aquesta secció describim el problema, n'analitzem la seva duresa i definim quina formulació específica tractarà el nostre generador. Tal com s'ha mencionat anteriorment, un dels problemes de la generació d'horaris és la diferència de criteris entre els instituts d'arreu.

2. Estudi de l'estat de l'art (Seccions 5 i 6)

Aspecte central del treball. Es considera fonamental entendre bé el raonament rere cada tècnica per poder-ne extreure el màxim rendiment a l'hora d'aplicar-la en l'implementació del generador.
A tal fi, s'han estudiat els fonaments de diverses tècniques deterministes molt efectives en la resolució de problemes similars. A més, s'han estudiat en profunditat vàries codificacions de restriccions de cardinalitat per a SAT i SMT.

3. Implementació del generador (Secció 8)

   En aquest apartat s'apliquen els coneixements apresos en el punt anterior en l'implementació de dos generadors d'horaris. Un basat en SAT i l'altre en SMT.

1.2 Grup de recerca LAP

Aquest treball s'enmarca dins el grup de recerca de Lògica i Programació (LAP)[1] de l'àmbit d'àrea tècnica de la Universitat de Girona.
El grup basa la seva recerca en l'estudi de la satisfactibilitat de formules proposicionals booleanes (SAT) i Satisfiability Modulo Theories (SMT) i les seves aplicació per a la resolució de problemes combinatoris com ara problemes de planificació i programació de tasques com ara la confecció d'horaris de tot tipus, seqüenciació de tasques en processos industrials, etc.

Durant la confecció del treball he rebut l'ajut i assessorament dels membres del grup, així com el del meu tutor de projecte el Dr. Josep Suy que també forma part del grup de recerca.

[1] `http://ima.udg.edu/Recerca/GrupESLIP.html`

2 Metodologia

Una part important del present treball és la confecció d'un generador d'horaris per a instituts. Per tal de crear-lo s'ha decidit adoptar una metodologia de prototipatge com a procés de desenvolupament.
Creiem que una estratègia de prototipatge és adient sobretot perquè l'altre part important d'aquest treball és l'estudi i implementació de tecnologies que són del tot noves per al desenvolupador. De manera que una estratègia de prototipatge pot ser el més adient ja que està més enfocada a un desenvolupament iteratiu i incremental, a l'hora que permetrà obtenir més *feedback* a mesura que es van creant prototips del generador final.

Aquest últim punt és essencial ja creiem que la millor manera d'aplicar noves tècniques i tecnologies consisteix en començar resolent problemes senzills i anar incrementant la dificultat a mesura que el grau de coneixement i experiència de les eines augmenta. És a dir, començar amb un generador d'horaris simple i anar-li afegint complexitat i funcionalitat a mesura que augmenta el grau de coneixement tant del problema com de les eines.

2.1 Software Prototyping

El model de prototipatge de software [1] consisteix a crear prototips d'aplicacions de software. És a dir, versions incompletes del software que s'està desenvolupant. Generalment, un prototip només compleix uns quants requeriments del sistema final i pot no tenir res a veure amb el producte final. El prototipatge presenta diversos beneficis:

- Permet al desenvolupador obtenir *feedback* molt aviat en les etapes de desenvolupament. En contraposició a altres metodologies que el *feedback* no arriba fins ben entrada l'etapa d'implementació. En el nostre cas, com ja hem dit, ens és molt valuós ja que ens permet realitzar tests unitàris sobre funcionalitats específiques.
- També augmenta le precisió de les estimacions de temps de dedicació per a la implementació de les funcionalitats. Permetent fixar dates límit i *milestones*[2] més coherents amb la realitat del desenvolupament.
- Els desenvolupadors poden centrar-se en desenvolupar parts del sistema que comprenen en comptes d'haver d'implementar tot el sistema de cop.

A grans trets el procés de prototipatge segueix els següents pasos:

1. Identificar requeriments bàsics

   Consisteix en identificar només les parts essencials del projecte a desenvolupar. En aquest cas les del generador. En el nostre cas consisteix

[2] Una *milestone* és una data senyalada a mode de referència per un grup de desenvolupament de software.

en identificar el mínim indispensable per a poder aplicar una determinada tècnica. És a dir identificar les restriccions que recullen la majoria d'instàncies obviant tots els casos extrems.

2. Desenvolupar un prototip inicial

   En general el prototipus inicial d'una tècnica només es capaç de resoldre instàncies específiques.

3. Revisar

   A partir del feedback del prototip inicial, podem comprovar com de bé funciona la nostra implementació d'una determinada tècnica. Com d'apropiada és aquella tècnica i quina complexitat té treballar-hi.

4. Millorar el protoip amb la informació obtinguda

   Procés iteratiu. Una vegada arribat a aquest punt, si el prototip ha donat bons resultats és refina i se li afegeixen noves funcionalitats (suport a nou tipus de restriccions, etc). Si per contra el prototip no ha donat bons resultats, és el moment d'abandonar-lo i començar el procés de nou.

3 Etapes del Treball

Aquest projecte tenia dos objectius principals. Per una banda la confecció d'un generador d'horaris i per l'altre la realització d'un estudi sobre les principals tècniques per resoldre problemes de satisfactibilitat de restriccions (CSP) de manera determinista. Per sort, els dos objectius combinen bé i centrant-nos en qualsevol dels dos ajuda a entendre/resoldre millor l'altre.

A mode esquemàtic, els següents punts recullen la planificació del temps i la feina realitzada durant la realització d'aquest treball fi de grau.

1. Abans de començar es va concertar una reunió amb els responsables de la generació d'horaris de l'institut Jaume Vicens Vives de Girona[3]. La trobada va servir per conèixer quin tipus de problemàtica els suposa la generació d'horaris per al seu institut així com obtenir informació del tipus de *solvers* que utilitzen.

2. A partir de l'informació obtinguda de la reunió es va concretar una primera formulació del problema. Una formulació simplista que permetés la confecció d'un primer prototip basat en MiniZinc, un framework de CSP. La confecció del primer prototip, juntament amb un conjunt d'instàncies de joguina. Tot plegat va portar al voltant d'una setmana de feina.

3. Un cop acabat el primer prototip es va resvisar juntament amb membres del grup de recerca LAP. Amb el feedback tant dels temps de còmput de les instàncies de joguina com dels membres del grup de recerca Es va realitzar un refinament sobre el primer prototip en MiniZinc.

4. Una setmana després i amb vàris refinaments introduïts, els resultats semblaven ser poc prometedors. Els temps de còmput eren elevats, tenint en compte que només preguntàvem per la satisfactibilitat del model, és a dir que no l'estàvem optimitzant. Per a més inri la formulació era rígida i poc flexible. Realitzant una mica de recerca en busca de consens sobre una formulació més flexible ens vam topar amb el web de la competició internacional de generadors d'horaris d'instituts [2]. Aquesta competició té una formulació pròpia que aspira a convertir-se en estàndard. A més incorpora nombroses instàncies basades en instituts reals d'arreu del món, així que decidim adoptar-la.

5. S'intenta realitzar un prototip amb MiniZinc capaç de generar horaris en format XHSTT (el format pròpi de la competició internacional), però després de més d'una setmana treballant-hi sense resultats concrets a causa de la complexitat del format XHSTT i els pobres resultats obtinguts de l'últim prototip ens fan desistir de la idea. S'acaba la nostra experiència amb MiniZinc.

[3] `http://ins-jvicensvives.xtec.cat/`

6. El format XHSTT és complex i autocontingut, per tant decidim optar per una codificació ad-hoc de la instància XHSTT a al tipus de problema que ens interessi (SAT, SMT...). Per a fer-ho comencem el desenvolupament d'una plataforma capaç de llegir arxius XML en format XHSTT i obtenir-ne una representació més orientada a objectes, per facilitar-ne la posterior codificació a un problema concret. Creem un $parser$[4] de XHSTT. A causa de l'encert en la tecnologia usada, en tenim prou amb poc més d'un dia.

7. Comencem la implementació d'un mòdul capaç de codificar una instància XHSTT a una instància SMT. Per tal de programar ad-hoc per au un sistema SMT o bé cal aprendre's el format SMTLIB [3] o programar a través de l'API d'algun SMT-solver. Comença un període de recerca fins que trobem el **z3 Theorem Prover**, un SMT-Solver que ens proporciona una API pel llenguatge en el que haviem programat fins ara. Tot plegat, trobar l'eina adecuada i documentar-nos mínimament sobre la seva interfície ens va dur més de dues setmanes.

8. En aquest moment el laboratori LAP ens convida a la pre-presentació de la tesi doctoral del doctorand Jesús Giráldez sobre SAT. La tesi tracta d'explicar a partir de l'estudi de l'estructura de les instàncies industrials la raó perquè els SAT-solvers són tan eficients, segons resultats empírics, a l'hora de resoldre-les [4]. L'event va ser de valor acadèmic incalculable, ja que es van tocar molts temes d'interés per al treball i des d'una perspectiva molt il·lustrativa.

9. Desenvolupem un prototip per a SMT basat en pseudo-booleans. Els resultats són regulars. Com abans, només estem preguntat per satisfactibilitat, encara no optimitzem horaris. I, tot i que acaba, els temps de còmput són respectables[5]. Tot plegat però ens serveix per agafar experiència amb l'API del SMT-solver. La confecció d'aquest prototip va necessitar de dues setmanes.

10. Revisant l'anterior prototip, ens adonem que aplicar SMT ha suposat una millora sobre el MiniZinc, però que el problema continua essent massa dur. Revisant documentació sobre els generadors implementats per participants de la competició internacional ens adonem que la majoria de generadors es realitzen a mida per cada tipus d'instància. Per exemple les instàncies de Brazil, tenen tots els recursos assignats a algun event i al generador només se li demana que organitzi els events. Ens sembla raonable doncs, intentar generar un prototip ad-hoc per instàncies de Brazil i veure què passa.

11. Implementem un prototip ad-hoc per instàncies de Brazil i similars[6] utilitzant vectors de bits (una altre teoria de SMT) i SAT. En una setmana

[4] Un parser és un algorisme capaç d'interpretar un format específic.
[5] De l'ordre de segons, per a instàncies reals, però petites
[6] Instàncies que no requereixin l'assignació de recursos als events.

més, tenim el primer prototip de vectors de bits funcionant i comencem una recerca més en profunditat sobre l'estat de l'art de SAT. tres setmanes més tard tenim la primera versió del prototip SAT. En total s'estima prop d'un mes.

12. Revisant els anteriors prototips, veiem que suposen una millora substancial respecte l'anterior versió amb pseudo-booleans. Ambdós generadors són capaços de decidir les instàncies en mil·lisegons. Aquest fet ens permet ser optimistes i passar a la següent fase: **optimitzar**.

13. Comencem per SAT, que és el més senzill. Refinem el prototip SAT a MaxSAT i utilitzem un dels múltiples solvers disponibles a la xarxa capaços d'optimitzar problemes d'aquest estil. El refinament requereix d'una tècnica especial per a les clàusules considerades "soft" així que comporta més d'un dia de feina.

14. Per a l'optimització del model de vectors de bits cal implementar l'algorisme d'optimització. Creem un prototip basat en un esquema de backtracking que utilitza el model de vectors de bits a mode d'oracle. Com en el cas de SAT, la codificació de les restriccions "soft" requereix d'una tècnica especial, molt semblant a la de SAT. Aquest primer refinament comporta poc més d'un dia de feina.

15. Comencem a efectuar els primers tests per comparar el prototip basat en MaxSAT i el basat en BitVectors (vectors de bits). El prototip basat en MaxSAT és molt més eficient a l'hora d'optimitzar, probablement a causa de la pobre implementació de l'optimitzador que hem implementat per als BitVectors. Realitzem un refinament sobre l'optimitzador de manera que el SMT-solver pugui mantenir la informació apresa entre crides i d'altres millores en el nombre de crides al SMT-solver. L'eficiència millora molt.

16. Finalment realitzem els tests comparatius entre el generador basat en MaxSat i el basat en BitVectors i en treiem conclusions.

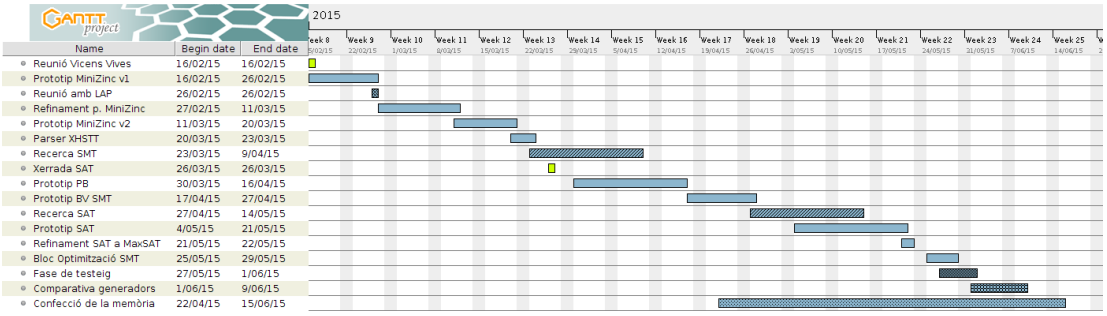

Figura 1: Diagrama de Gantt de la planificació del treball.

4 Definició del problema

El problema de la generació d'horaris per a un institut (HSTT) correspon al problema en què es troben de manera recorrent els instituts a l'hora d'assignar per a cada assignatura que el centre imparteix una sèrie de recursos i un espai de temps durant el qual s'imparteix l'assignatura. Típicament els recursos són professors i aules, però es parla de recursos de manera genèrica per contemplar la possibilitat que hi hagi instituts amb requeriments especials. La part dura doncs, és la d'assignar un espai de temps a cada assignatura sense violar cap restricció sobre els seus recursos. Com per exemple, una restricció natural podria ser imposar que a l'assignar una assignatura a un espai de temps determinat, tots els seus recursos han d'estar disponibles per aquest espai de temps. Com aquesta es poden imposar restriccions de naturaleses diverses, cada una d'elles reduint el nombre de combinacions vàlides. De manera que amb l'addició de noves restriccions es va fent més i més difícil la confecció de l'horari de manera manual.

4.1 Duresa del problema.

En teoria de la computació es classifiquen els problemes segons la seva duresa. Existeixen vàries classificacions:

- Problemes decidibles
- Problemes indecidibles

El problema HSTT és decidible i és fàcil veure que pertany a **NP**[7] ja que comprovar si una solució satisfà totes les restriccions es clarament polinòmic.

Si considerem que tots els recursos estan sempre disponibles, llavors el problema HSTT és polinòmic[8]. Però aquesta consideració no és real, ja que els centres disposen d'un nombre limitat de professors i un professor només pot atendre un nombre limitat d'assignatures simultàniament, els instituts també tenen un nombre limitat d'aules disponibles i en una aula només hi caben un nombre limitat de grups d'estudiants, etc.
És precisament aquest aspecte el que fa especialment dur el problema de la generació d'un horari, com assignar hores a les diferents assignatures mantinguent la coherència entre els recursos.

4.1.1 NP-Completesa de HSTT

En aquest apartat demostrarem que el problema HSTT és **NP − Complet**[9] realitzant una reducció d'un problema NP-Complet conegut a HSTT. En concret

[7] NP engloba tots els problemes decidibles pels quals existeix un algorisme indeterminista de complexitat polinòmica.

[8] Polinòmic o **P** fa referència als problemes pels quals existeix un algorisme determinista de complexitat polinomial.

[9] NP-Complet engloba tots aquells problemes NP als quals qualsevol problema decidible es pot reduïr en termes del primer.

utilitzarem el problema de la motxilla [5], d'acord amb la demostració proposada per B. Cooper i J.H Kingston a [6].

Com ja hem apuntat una de les fonts de la duresa del problema ve a l'hora de gestionar els recursos de manera coherent. De manera que al confeccionar un horari serà d'interès mantenir el nombre d'incoherències per sota un llindar. Les instàncies HSTT acostumen a tenir com a mínim un tipus de recurs que compleix aquestes premises. En cas que no en tinguin, és possible realitzar una transformació binària per arribar a aquesta formulació, ja que tots els recursos poden atendre a un nombre limitat d'assignatures de manera simultànea i totes les instàncies disposen d'un nombre limitat de recursos.
Per tant, i per simplificar, considerarem només un únic recurs, de disponibilitat limitada. També considerarem que un recurs només pot atendre una assignatura alhora.

Diem que dos assignatures són incoherents si comparteixen algun espai de temps. És a dir, si es solapen. Com que els recursos són limitats interessa limitar el nombre de solapaments. Utilitzarem una codificació del problema de la motxilla per representar aquesta situació.

El problema de la motxilla consisteix en determinar si un conjunt d'items $U = \{u_1 \dots u_n\}$, cadascun amb un pes específic associat w_i, es poden col·locar dins d'un conjunt de motxilles $B = \{b_1 \dots b_m\}$, cadascuna amb una capacitat màxima c_i de manera que cap motxilla sobreexcedeixi la seva capacitat.

Transformem el problema de la motxilla al següent problema HSTT.

$$Times = \{t_{1,1} \dots t_{n,m}\}$$
$$Events = X \cup Y, \quad X = \{x_1 \dots x_n\} \quad Y = \{y_1 \dots y_m\}$$

De manera que:

- A cada x_i se li han d'assignar tants espais de temps com w_i.
- A cada y_i se li han d'assignar tants espais de temps com c_i i els corresponents a la motxilla que representen.

  És a dir, y_i tindrà assignats els espais de temps $\{t_{i,1} \dots t_{i,c_i}\}$

El problema doncs, rau en determinar quins espais de temps s'assignen a cada x_i.
Suposem que aquest problema formulat com el de la motxilla té solució $f : U \to B$ on el valor de retorn de f és l'index de la motxilla on s'ha col·locat l'ítem d'entrada. Llavors, per cada assignatura x_i, escollim w_i espais de temps, que no hagin estat escollits prèviament, del conjunt $S_k = \{t_{k,1} \dots t_{k,c_k}\}$, on $k = f(u_i)$. És a dir, s'escullen tants espais de temps com el pes de l'ítem que representa, de

manera que no hi hagi solapaments entre els membres de X. Aquest procés és possible perquè f ens garanteix que com a molt s'escolliran c_k espais de temps de S_k. Al final tenim que tots els events tenen assignats exactament w_i espais de temps i cada X_i se solapa amb un, i només un, event de Y. per tant tenim que el nombre d'incoherències o solapaments és n.

Ara, suposem que la instància HSTT que hem descrit genera una solució amb un nombre de solapaments $\leq n$. Sabem que com a mínim el nombre de solapaments ha de ser $\geq n$, ja que cada x_i s'ha de solapar com a mínim una vegada amb algun membre de Y. Per tant, el nombre de solapaments de la solució generada per la instància HSTT ha de ser exactament n i cada event x_i es solapa només una vegada amb un sol membre de Y. Podriem reeimplementar doncs f de manera que a partir de la solució obtinguda per la instància HSTT es limiti a esbrinar per cada event u_i, amb quin event y_j se solapa, de manera que $f(u_i) = j$. $\square$

4.2 Format XHSTT

Com ja hem vist el problema de la generació d'horaris d'institut és complex, tant per l'enorme combinatòria que amaga com per la dificultat a l'hora d'establir un model prou genèric per abarcar totes les possibles restriccions. És per aquest motiu que en la confecció d'aquest treball s'ha decidit adoptar el format genèric per la resolució del *High School Timetabling problem* (HSTT), tal i com es descriu a [7]. El format rep el nom de XHSTT [8] que és l'acronim de *Xml-format High School Timetabling*.

Aquesta formulació defineix tres entitats principals: hores, recursos i events. Les hores fan referència als periodes de classe disponibles com ara Dilluns de 9:00 a 10:00. No hi ha cap mena de limitació en la durada d'aquests períodes. Ara bé es demana que tots tinguin la mateixa durada. Els recursos fan referència a les aules, professors, grups, etc. La formulació gira en torn els events, que representen les assignatures en sí mateixes i són les que requereixen que se'ls assigni certs recursos i hores. Per exemple, un event podría ser una assignatura de Matemàtiques, la qual necessita un professor, una aula, un grup d'alumnes determinat (per exemple, els alumnes de 1er A) i un nombre específic d'hores. Les restriccions ens serveixen per definir quines assignacions estàn permeses. Com per exemple, pot passar que calgui imposar que un professor determinat només pugui *assistir* a un nombre específic d'assignatures; o exigir que als alumnes dels cursos de primer se'ls programin les assignatures més "dures" (segons algún criteri arbitrari) a les primeres hores del dia.

A continuació farem una síntesi a alt nivell de les restriccions que apareixen al format XHSTT [8]. Es pot trobar una descripció més detallada i específica al al web que s'adjunta al peu de pàgina[10].

[10] Format XHSTT: `http://sydney.edu.au/engineering/it/~jeff/hseval.cgi?op=spec`

Fins al moment hem vist que el format consta de tres entitats principals:

- Hores

  Cada hora pot estar vinculada a diversos grups d'hores. Mitjançant la definició de grups el format permet dotar de major expressivitat al model·lar. Els grups poden indicar un dia o una frànja horaria determinada, com per exemple matí o tarda.

Figura 2: Exemple de declaració d'una Hora en format XHSTT

```
<Time Id="Mon4">
    <Name>Mon4</Name>
    <Day Reference="Monday" />
    <TimeGroups>
        <TimeGroup Reference="BeforeLunch" />
    </TimeGroups>
</Time>
```

L'element *Day* de l'xml de la figura 2 és el mateix que un *TimeGroup*, però el format permet definir-los a part per facilitar la feina de les eines gràfiques.

- Recursos

  Cada recurs té un tipus determinat. Exemples de tipus poden ser Professor, Aula, Classe (grup d'alumnes), etc... Els que s'han enumerat són els més comuns, però la definicio de tipus és totalment arbitrària i serveixen per dotar de major semàntica.

Figura 3: Exemple de declaració d'un Recurs en format XHSTT

```
<Resource Id="r03">
    <Name>r03</Name>
    <ResourceType Reference="Room" />
    <ResourceGroups>
        <ResourceGroup Reference="ScienceLab" />
    </ResourceGroups>
</Resource>
```

- Events

  A la definició d'un Event s'hi especifica quin tipus de recursos necessita (També és possible assignar un recurs en concret), el nombre d'hores que necessita l'event i si pertany a algún grup d'events. Com que el format

permet la possibilitat de que un event pugui requerir dos o més recursos del mateix tipus, a la definició de cada recurs s'hi ha d'adjuntar el rol del recurs. Altre cop això és més útil a eines gràfiques que no pas al nostre generador.

Figura 4: Exemple de declaració d'un Event en format XHSTT

```
<Event Id="T1-S1">
    <Name>T1-S1</Name>
    <Duration>3</Duration>
    <Course Reference="gr_T1-S1" />
    <Resources>
        <Resource Reference="S1">
            <Role>Class</Role>
            <ResourceType Reference="Class" />
        </Resource>
        <Resource>
            <Role>Teacher</Role>
            <ResourceType Reference="Teacher" />
        </Resource>
    </Resources>
    <EventGroups>
        <EventGroup Reference="gr_AllEvents" />
    </EventGroups>
</Event>
```

En l'xml de la figura 4 podem veure com es defineix un event que requereix dos recursos. Un de concret i l'altre sense especificar. El que està sense especificar indica que és responsabilitat del generador assignar-hi un recurs del tipus que s'especifica. Pot assignar-hi el que vulgui d'aquell tipus, sempre que cap *constraint* que es defineixi a posteriori indiqui el contrari.

L'objectiu del generador d'horaris és el d'assignar a cada Event els Recursos i Hores que requereixi. Per tal de posar límits a com el generador realitza aquestes assignacions, el format preveu un seguit de restriccions. En aquest treball hem cobert un subconjunt de les instàncies XHSTT, per tant hi han tipus de restriccions que el nostre generador no té en compte. A continuació s'enumeren totes les restriccions que el generador sí té en compte.

4.2.1 Restriccions XHSTT

En aquesta secció enumerem els tipus de restriccions definides pel format XHSTT que el nostre generador reconeix. La secció té com a objectiu donar idea del tipus de problemes que serà capaç de resoldre el nostre generador, no pas de

descriure cada restricció al detall[11].

En general una restricció és una condició que la solució ha d'intentar complir, si és possible.
Cada restricció té un conjunt de punts d'aplicació, que corresponen a entitats d'algun dels tipus vistos en l'apartat anterior (Hores, Recursos o Events). Tota solució té un cost associat, que s'extreu del conjunt de restriccions de l'instancia. El cost d'una solució correspon a la suma del cost associat a cada restricció que la solució violi. Per tant, cada restricció també du informació del cost que suposa no respectar-la.

També cal diferenciar entre dos tipus de restriccions: *hard-constraints* i *soft-constraints*, la primera fent referència a restriccions que el generador no té permès violar i la segona a les restriccions que el generador pot violar. No obstant, tant les *hard* com les *soft*, tenen un cost associat. Això provoca que a l'hora de fer el còmput del cost d'una solució es retornin dos valors: el cost d'inviabilitat i el cost de la solució. Òbviament, si el generador viola alguna hard-constraint, la solució es marca com a inviable, i per tant no es considera solució. Però calcular-ne el valor d'inviabilitat pot donar idea de la duresa de la instància, que pot ser útil a l'hora de descriure la instància. El cost de la solució dona idea de la qualitat d'aquesta. Com més baix és el cost, de més qualitat és la solució.

El format XHSTT és un format encara no tancat, i preveu l'addició de nous tipus de restriccions en un futur. Per tant, i com ja hem dit, en aquest treball només en considerem un subconjunt.

- **Assign Time Constraints**

  Determina el cost que té no assignar cap hora als events als quals s'aplica. Acostuma a ser *hard* ja que un horari en el qual queden assignatures sense assignar no té massa sentit.

[11] Per una informació més detallada podeu visitar la documentació a `http://sydney.edu.au/engineering/it/~jeff/hseval.cgi?op=spec&part=constraints`

```
<AssignTimeConstraint Id="AssignTimes">
    <Name>AssignTimes</Name>
    <Required>true</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <EventGroups>
            <EventGroup Reference="gr_AllEvents"/>
        </EventGroups>
    </AppliesTo>
</AssignTimeConstraint>
```

Figura 5: Exemple d'una restricció del tipus Assign Time Constraint

- **Split Events Constraints**

  S'utilitza per indicar el generador com volem que ens parteixi les diferents assignatures. Per exemple, si s'han de fer 5 hores de matemàtiques aquesta restricció ens pot servir per indicar-li al generador que volem que ens parteixi l'assignatura en 3 lliçons d'entre una i dues hores.

```
<SplitEventsConstraint Id="SplitEventsConstraint">
    <Name>Split events to duration 1 and 2</Name>
    <Required>true</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <EventGroups>
            <EventGroup Reference="gr_AllEvents"/>
        </EventGroups>
    </AppliesTo>
    <MinimumDuration>1</MinimumDuration>
    <MaximumDuration>2</MaximumDuration>
    <MinimumAmount>1</MinimumAmount>
    <MaximumAmount>999</MaximumAmount>
</SplitEventsConstraint>
```

Figura 6: Exemple d'una restricció del tipus Split Events Constraint

- **Distribute Split Events Constraints**

  Ens serveix per posar límits en el nombre de lliçons d'una durada determinada que volem d'un grup d'assignatures. Per exemple serà útil si volem que totes les lliçons d'educació física de l'institut tinguin com a molt una lliçó que només duri una hora.

```
<DistributeSplitEventsConstraint Id="DistributeSplit_2">
   <Name>At least 2 double lesson(s)</Name>
   <Required>false</Required>
   <Weight>1</Weight>
   <CostFunction>Linear</CostFunction>
   <AppliesTo>
      <EventGroups>
         <EventGroup Reference="gr_T2-S1" />
         <EventGroup Reference="gr_T2-S2" />
         <EventGroup Reference="gr_T5-S2" />
         <EventGroup Reference="gr_T5-S3" />
         <EventGroup Reference="gr_T6-S1" />
         <EventGroup Reference="gr_T6-S2" />
         <EventGroup Reference="gr_T6-S3" />
         <EventGroup Reference="gr_T7-S1" />
         <EventGroup Reference="gr_T7-S3" />
      </EventGroups>
   </AppliesTo>
   <Duration>2</Duration>
   <Minimum>2</Minimum>
   <Maximum>2</Maximum>
<DistributeSplitEventsConstraint>
```

Figura 7: Exemple d'una restricció del tipus Distribute Split Events Constraint

- **Prefer Times Constraints**

  Ens serveix per indicar al generador que certs events s'han de programar en unes hores determinades. Per exemple, ens pot servir per evitar que el generador intenti assignar events que durin més d'una hora a la última hora del dia. Fet que podria provocar que hi hagin events que comencin a última hora d'un dia i acabin a primera hora del dia següent, que tampoc té massa sentit.

```
<PreferTimesConstraint Id="PreferredTimes">
    <Name>Times for duration 2</Name>
    <Required>true</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <EventGroups>
            <EventGroup Reference="gr_AllEvents"/>
        </EventGroups>
    </AppliesTo>
    <TimeGroups>
        <TimeGroup Reference="gr_TimesDurationTwo"/>
    </TimeGroups>
    <Duration>2</Duration>
</PreferTimesConstraint>
```

Figura 8: Exemple d'una restricció del tipus Prefer Times Constraint

- **Spread Events Constraints**

  Una restricció d'aquest tipus ens serveix per indicar que els events d'un determinat grup s'han de separar en el temps. Per exemple, pot ser útil per imposar que per cada grup com a molt hi hagi una lliçó de matemàtiques al dia.

```
<SpreadEventsConstraint Id="SpreadEvents_2">
    <Name>Spread events max 1 per day</Name>
    <Required>true</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <EventGroups>
            <EventGroup Reference="gr_T1-S1" />
            <EventGroup Reference="gr_T1-S2" />
        </EventGroups>
    </AppliesTo>
    <TimeGroups>
        <TimeGroup Reference="gr_Mo">
            <Minimum>0</Minimum>
            <Maximum>1</Maximum>
        </TimeGroup>
        <TimeGroup Reference="gr_Tu">
            <Minimum>0</Minimum>
            <Maximum>1</Maximum>
        </TimeGroup>
    </TimeGroups>
</SpreadEventsConstraint>
```

Figura 9: Exemple d'una restricció del tipus Spread Events Constraint

- **Avoid Clashes Constraint**

  Aquesta restricció especifica que cap dels recursos als que s'aplica pot assistir a més d'un event a l'hora. Restricció natural per modelar el comportament dels professors, que no poden estar a dos llocs al mateix temps i per les aules, que no poden acollir dues assignatures de manera simultània. Notis però, que el format permet modelar situacions en que un recurs pugui assistir a més d'un event de manera simultània.

```
<AvoidClashesConstraint Id="NoResourceClashes">
    <Name>NoResourceClashes</Name>
    <Required>true</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <ResourceGroups>
            <ResourceGroup Reference="gr_Teachers"/>
            <ResourceGroup Reference="gr_Classes"/>
        </ResourceGroups>
    </AppliesTo>
</AvoidClashesConstraint>
```

Figura 10: Exemple d'una restricció del tipus Avoid Clashes Constraint

- **Avoid Unavailable Times Constraints**

  Indica que hi han certes hores durant les quals certs recursos no estan disponibles. Per exemple, el cas que un professor no treballi dilluns, o només estigui disponible al matí.

```
<AvoidUnavailableTimesConstraint Id="AvoidUnavailableTimes_T1">
    <Name>ForbiddenTimesOfT1</Name>
    <Required>true</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Resources>
            <Resource Reference="T1"/>
        </Resources>
    </AppliesTo>
    <Times>
        <Time Reference="We_1"/>
        <Time Reference="We_2"/>
        <Time Reference="We_3"/>
        <Time Reference="We_4"/>
        <Time Reference="We_5"/>
    </Times>
</AvoidUnavailableTimesConstraint>
```

Figura 11: Exemple d'una restricció del tipus Avoid Unavailable Times Constraint

- **Limit Idle Times Constraint**

  Es considera que un recurs està ocupat en un determinat espai de temps si en aquell espai de temps assisteix a algun event. Llavors, es considera que un recurs està ociós (respecte un grup d'hores, per exemple un dia) en un determinat espai de temps quan en aquell espai de temps no està ocupat, però en canvi si que té programats events abans i després d'aquell espai dins del mateix grup d'hores. És a dir, la situació en que un professor té classe de 9:00 a 10:00 i de 11:00 a 12:00 però no té res programat entre les 10:00 i les 11:00 del mateix dia.

  Aquesta restricció ens permet posar límits en el nombre d'hores en que un recurs pot estar "ociós".

```
<LimitIdleTimesConstraint Id="noIDLETimesT">
    <Name>No IDLE times for teachers</Name>
    <Required>false</Required>
    <Weight>3</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <ResourceGroups>
            <ResourceGroup Reference="gr_Teachers"/>
        </ResourceGroups>
    </AppliesTo>
    <TimeGroups>
        <TimeGroup Reference="gr_Mo"/>
        <TimeGroup Reference="gr_Tu"/>
        <TimeGroup Reference="gr_We"/>
        <TimeGroup Reference="gr_Th"/>
        <TimeGroup Reference="gr_Fr"/>
    </TimeGroups>
    <Minimum>0</Minimum>
    <Maximum>0</Maximum>
</LimitIdleTimesConstraint>
```

Figura 12: Exemple d'una restricció del tipus Limit Idle Times Constraints

- **Cluster Busy Times Constraint**

  Es considera que un recurs està ocupat en un determinat espai de temps si en aquell espai de temps assisteix a algun event. Aquesta restricció permet posar límits en el nombre de dies o grups d'hores en que un recurs pot estar ocupat. Per exemple, ens permet indicar que un professor ha de donar totes les seves classes en dos dies.

```
<ClusterBusyTimesConstraint Id="MaxNofDaysConstraint_T_days_3">
    <Name>Not more than 3 days with lessons</Name>
    <Required>false</Required>
    <Weight>9</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Resources>
            <Resource Reference="T6" />
        </Resources>
    </AppliesTo>
    <TimeGroups>
        <TimeGroup Reference="gr_Mo" />
        <TimeGroup Reference="gr_Tu" />
        <TimeGroup Reference="gr_We" />
        <TimeGroup Reference="gr_Th" />
        <TimeGroup Reference="gr_Fr" />
    </TimeGroups>
    <Minimum>0</Minimum>
    <Maximum>3</Maximum>
</ClusterBusyTimesConstraint>
```

Figura 13: Exemple d'una restricció del tipus Cluster Busy Times Constraint

5 Estat de l'art

En aquesta secció realitzarem un repàs sobre diverses tecnologies aplicables en el camp de l'optimització en general així com una síntesi d'aquelles tecnologies que són especialment indicades per a tractar problemes de l'estil HSTT. Com ja hem vist en la secció anterior, HSTT és un problema de *scheduling* NP-complet, i per tant sense un mètode de resolució determinista en temps polinòmic conegut. El fet que el problema en sí es presenti de forma recurrent, el fa d'especial interès. Per tal d'estimular la participació d'investigadors s'han arribat a organitzar vàries edicions d'una competició mundial[12] de generadors d'horaris.

En tot problema d'aquestes característiques, i la generació d'horaris no n'és cap excepció, existeixen tres aproximacions:

1. Resolució per força bruta (*naive approach*)

   Consisteix en generar totes les combinacions possibles (de manera més o menys intel·ligent) i quedar-nos amb l'òptima. Només raonable per a instàncies trivials o quasi-trivials[13]. En general les instàncies dels instituts, no cauen dins aquesta categoria. Per tant calen aproximacions alternatives.

2. Resolució determinista (amb certificat d'optimalitat)

   La idea de fons no dista molt de la resolució per força bruta, ja que es busca el màxim absolut de totes les solucions possibles. És a dir, la millor solució d'entre totes les combinacions que respecten els requeriments de l'horari (si n'hi ha). Essent puristes la resolució per força bruta podria considerar-se com a un subconjunt de la resolució determinista, però en aquesta secció només considerarem totes aquelles tècniques que suposen una millora clara respecte la primera. Entren dins la categoria totes les tècniques basades en *constraint programming* (CP) així com les basades en reduccions a altres problemes NP-complets com poden ser SAT[14], SMT[15], MIP[16], etc.

   Tot i que no aconsegueixen sobrepassar la barrera de la complexitat, en els últims anys el camp ha patit grans avenços i presenta un futur molt prometedor. Avui dia algunes d'aquestes eines son capaces de certificar optimalitat (o insatisfactibilitat) d'instàncies reals en temps molt competitius. Malgrat l'esforç i la dedicació de un bon gruix de la comunitat científica, queda molt espai per a la millora fet que n'augura un futur molt prometedor.

[12] `http://www.utwente.nl/ctit/hstt/itc2011/welcome/`

[13] Definim *quasi trivial* com aquella instància que tot i suposar un esforç massa gran per a ser resolta a mà, es manté computable per a un procés informatitzat de força bruta. Sería el cas de la generació d'horaris per a acadèmies de reforç o horaris d'escoles molt i molt petites

[14] Problema de satisfactibilitat d'una fòrmula proposicional Booleana.

[15] *Satisfiability Modulo Theories*, un SAT enriquit on cada literal booleà amaga una fòrmula lògica d'un altre tipus de teoria.

[16] *Mixed Integer Programming*, proposa modelar el problema com un conjunt d'equacions definides amb enters i reals.

3. Resolució indeterminista (mètodes quasi-òptims)

   La idea és simple: una solució bona, encara que no òptima és millor que cap solució. Aquestes tècniques deixen de banda la certificació d'optimalitat i es centren en trobar la millor de les solucions en el mínim espai de temps possible. Aconseguint un molt bon ràtio entre el temps de còmput i la qualitat de la solució obtinguda. Aquesta característica i l'extrema duresa del problema perfila a aquest tipus de tècniques com les vencedores absolutes al camp pràctic, ja que els instituts els preocupa poc l'optimalitat de l'horari obtingut. Aquest mètodes són especialment indicats a causa de les limitacions del hardware sobre el que cada institut fa córrer el seu generador d'horaris i les limitacions en el temps de còmput de l'execució, que generalment és de l'ordre de segons (al voltant de 5 minuts[17]). Per tant doncs, per als instituts són d'especial interès els mètodes que permetin obtenir solucions de qualitat raonable en poc temps.
   Com a contrapartida, aquests mètodes sovint els manca la noció d'acabament ja que és difícil saber quan s'ha assolit el màxim (o mínim) absolut. I el que es més greu, no són complets i no és estrany que es marquin instàncies satisfactibles com a insatisfactibles.

Les instàncies reals d'HSTT acostumen a generar models d'extrema duresa, això suposa una dificultat afegida per als generadors deterministes, que en l'intent de certificar l'optimalitat d'una solució han d'explorar tot l'espai de cerca. No obstant, i gràcies a l'esforç i dedicació de la comunitat científica, el sector està patint una autèntica revolució. Com ja hem dit en aquest treball ens centrem en les estratègies de resolució deterministes. A continuació realitzem una petita síntesi dels punts clau de les tècniques deterministes que ens han semblat de més interès.

5.1 Constraint Satisfaction Problems (CSP)

CSP presenta un paradigma de programació diferent del clàssic paradigma imperatiu, on lluny d'implementar algorismes de resolució, l'atenció es centra en la declaració de les diferents restriccions (*constraints*) que defineixen el problema. Aquest paradigma és altament declaratiu i permet resoldre problemes complexos amb poques línies de codi. Reduint notablement l'aparició d'errors en el codi del problema. I permetent al programador centrar-se més en el problema i menys en l'algorísmica.
però el que el fa únic no és la seva alta declarabilitat, sino les tècniques que amaga. Per entendre bé què fa que

Com ja hem dit, CSP es centra en les restriccions. La idea es trobar una assignació de totes les variables que no violi cap de les restriccions definides. És a dir, que sigui consistent amb totes. Lluny de provar totes les combinacions de valors per a totes les variables (pròpi d'una estratègia per força bruta), la

[17] Segons l'encarregada de la generació d'horaris de l'institut *Jaume Vicens Vives* de Girona.

resolució d'un model CSP passa per minimitzar el nombre d'assignacions que violen alguna restricció. De manera que es construeix una solució a partir d'assignacions parcials que sempre són consistents amb les restriccions. Tant bon punt es troba una variable tal que, sigui quin sigui el valor que se li assigni acaba violant alguna restricció, es fa marxa enrere cap a alguna assignació anterior. En aquest sentit, a l'hora d'assignar nou valor a una variable s'escull només d'entre el rang de valors del seu domini que són consistents amb les variables ja assignades i les restriccions del model.

Arc-Consistència (AC)

Un concepte molt important en els CSP és el de l'arc-consistència. La noció de consistència són un conjunt de propietats que s'extreuen de les restriccions i s'usa per eliminar valors inconsistents dels dominis de les variables. L'arc consistència és un tipus de consistència, en particular és la consistència que s'aplica a nivell de restricció.
Diem que una restricció **c** és Arc-Consistent (AC) *sii* per cada variable $\mathbf{x} \in vars(\mathbf{c})$ i per cada valor de x, existeix un suport a **c**. És a dir, per cada variable i valor d'una restricció existeix alguna combinació de la resta de variables que satisfà la restricció. Un CSP és AC *sii* cap variable té domini buit i totes les seves restriccions són arc-consistents.
Com últim apunt, cal veure que l'Arc-Consistència no és completa, és a dir, no n'hi ha prou amb mantenir l'arc-consistència per evitar arribar a contradiccions. Per contrapartida, és molt més ràpida de computar que la k-consistència (que en el seu límit té cost exponencial); fet que permet executar-la molt més sovint sense comprometre la velocitat de resolució final.

Algorisme de resolució d'un problema CSP:

```
def CSP(V, C):
    AC(V, C)
    for v in V if v.unassigned():
        for value in v.domain():
            v.assign(value)
            solution = CSP(V, C)
            if solution is not 'unsatisfiable':
                return solution
        return 'unsatisfiable'
    return V  # satisfiable
```

La funció **AC** s'encarrega de mantenir l'arc-consistència entre totes les seves restriccions, actualitzant els dominis de cada variable per assignar. Com que en tot moment tenim assignacions parcials consistents amb totes les restriccions, tan bon punt som capaços d'assignar valor a totes les variables, ja podem dir que hem trobat una solució. Mentre que si fallem a assignar valor a una variable, hem de fer marxa enrera i refer l'assignació de la variable anterior. Notís que en el cas que alguna variable es quedi amb un domini buit també toca fer

marxa enrera.

Òbviament els algorismes de resolució reals són més complexos que els presentats en l'anterior figura, però ens serveix per a fins il·lustratius. Un altre aspecte que aquí no revisarem en detall però que val la pena mencionar són les anomenades restriccions globals (*global constraints*). Aquest tipus de constraint s'utilitzen a alt nivell i permeten al *solver* atacar amb estratègies específiques petites parts del problema general. La restricció "*alldifferent*" especifica que un conjunt de variables han de prendre valors diferents les unes amb les altres. És a dir, totes han de prendre un valor diferent. Doncs bé, aquesta restricció apareix a molts problemes de naturaleses diferents, i poder-ne fer abstracció permetent al *back-end* del solucionador aplicar tècniques específiques per a inferir el màxim de coneixement possible, permetent propagar més, reduïnt el temps de còmput. En particular per a aquesta restricció s'apliquen tècniques basades en grafs bipartits[9].

Val la pena fer especial menció al MiniZinc[18], un llenguatge de nivell intermig basat en flatzinc que pretén convertir-se en llenguatge estàndard per a la programació de problemes CSP. Un dels principals beneficis és que permet desacoplar la modelització del problema (la seva declaració) de la seva resolució, permetent que les constraints declarades per el programador siguin interpretades per un solucionador (*solver*) a mode de *caixa negra*[19], que s'encarrega de buscar una assignació de variables que satisfaci totes les constraints del model.

5.2 Mixed Integer Linear Programming (MILP)

Un problema MIP és un problema d'optimització matemàtic. Cauen en aquesta categoria tots els problemes CSP on les variables són enteres o reals i totes les restriccions són lineals. Generalment Tenen una funció objectiu a maximitzar (o minimitzar) i un seguit d'inequacions lineals. El prefix *Mixed* indica que permet que algunes de les seves variables siguin reals.

La duresa del paradigma radica en la restricció de mantenir enteres algunes de les seves variables, ja que sense tal restricció el problema seria fàcilment optimitzable aplicant eines de resolució de sistemes d'equacions.

Més formalment, un programa lineal es defineix com una matriu $A : \mathbb{R} \to \mathbb{R}^n$ i dos vectors $b, c \in \mathbb{R}^n$. La seva resolució es basa en trobar el vector x que maximitzi el valor d'una funció objectiu alhora que satisfà totes les restriccions. És a dir:

$$c^t x = \max_{y \ t.q. \ Ay \leq b} c^t y$$

En altres termes, el màxim $c^t x$ tal que $Ax \leq b$. On $c^t x$ sería la funció objectiu i $Ax \leq b$ les restriccions del problema.

[18] `www.minizinc.org`

[19] On el *solver* és totalment transparent a l'usuari.

Un exemple de MILP:

$$\begin{aligned} Max &: 2x + y + 3z \\ tal\ que &: x + 2y \leq 4 \\ &\quad y - 5z \leq 5 \\ &\quad x, y, z \geq 0 \end{aligned}$$

Que tindria la següent matriu i vectors:

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & -5 \end{pmatrix} \qquad c = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} \qquad b = \begin{pmatrix} 4 \\ 5 \\ 0 \end{pmatrix}$$

Una característica dels problemes MILP és que les variables han de ser positives. Però no suposa un problema major, ja que per representar valors negatius n'hi ha prou amb desplaçar els resultats. Això és possible sempre que els dominis siguin finits, és clar.

5.3 SAT

El problema de satisfactibilitat d'una formula en lògica proposicional (SAT) fou el primer en demostrar-se NP-Complet per Steven Cook l'any 1971 [10]. Consisteix en decidir la satisfactibilitat d'una fòrmula en lògica proposicional booleana, i en cas que sigui satisfactible proporcionar un model que la satisfaci. Per exemple, la fòrmula $\phi = (A \vee B) \wedge \overline{B}$ és satisfactible i té l'assignació I com a model tal que $I(A) = Cert$, $I(B) = Fals$.
Es requereix que les instàncies de SAT estiguin en CNF[20] però no suposa cap problema ja que la conversió d'una formula booleana qualsevol a una equivalent en CNF es pot efectuar en temps polinòmic.

La seva simplicitat de formulació y el fet que qualsevol problema decidible es pot reduïr[21] polinomicament a SAT l'ha fet d'especial interès per a la comunitat científica. Malgrat tot, avui dia no es coneix cap algorisme capaç de resoldre aquest problema en temps polinòmic. No obstant, en els últims anys hi han hagut importants avenços i els *solvers* actuals són capaços de resoldre instàncies de milers de variables i fins a milions de clàusules. Una fita totalment inabastable per a un mecanisme de resolució per força bruta.

Però com s'ho fa un *sat solver* per a resoldre instàncies d'aquestes magnituds? Mètodes indeterministes a banda, ens centrarem en el mètode determinista més rellevant dels últims anys. Però abans introduirem dos conceptes bàsics però essencials.

[20] CNF (Conjunctive Normal Form): Tota formula booleana està en CNF *sii* (1) només es compon de literals i els operadors $\vee$,$\wedge$ i $\neg$. (2) és una conjunció de clàusules on cada clàusula és al seu torn una disjunció de literals i/o les seves negacions. Notis que en una formula CNF les negacions només s'apliquen als literals, mai a cap clàusula ni la pròpia formula.

[21] Per reduïr volem dir que qualsevol problema decidible es pot reescriure en termes de SAT.

1. **Propagació unitària**

   Imaginem que en el procés de resolució d'una instància ens trobem amb la següent clàusula:

   $$(X_a \vee X_b \vee X_c \vee X_z)$$

   Suposem també que totes les variables menys X_a tenen valor assignat i que cap satisfà la clàusula.

   $$I(X_b) = I(X_c) = I(X_z) = Fals$$

   Llavors, si no volem fer insatisfactible la clàusula hem de fer certa la variable X_a ($I(X_a) = Cert$). A aquest raonament l'anomenarem propagació unitària, ja que un cop hem inferit el valor de la variable X_a, podem propagar-lo a la resta de clàusules on apareixia X_a. Clàusules on pot tornar-se a repetir la mateixa situació però ara amb una altra variable.

2. **Regla de Resolució**

   En lògica proposicional la regla de resolució es una regla d'inferència que produeix una nova clàusula com a resultat de la implicació de dos clàusules que contenien literals complementaris[22].

   Per exemple, considerem les dues clàusules següents:

   $$(A \vee B \vee C) \;\; i \;\; (\overline{A} \vee B)$$

   aplicant la regla de resolució obtenim, cancel·lant A amb $\overline{A}$:

   $$(B \vee C)$$

   Si analitzem la semàntica de la clàusula obtinguda, ens fixem que ens indica que B i C no poden ser falses alhora. I efectivament, no es difícil veure que si falsifiquem les dues variables, la formula es torna insatisfactible.

Tot seguit presentem l'algorisme base de la majoria de SAT-solvers deterministes actuals. L'algorisme DPLL.

5.3.1 Davis-Putnam-Logemann-Loveland (DPLL) algorithm

Data de l'any 1962 i és un refinament de l'algorisme basat en la regla de resolució presentat per Martin Davis i Hilary Putnam l'any 1960. Ambdós tenen com a finalitat la resolució d'instàncies del problema SAT. Si bé es l'ultim el més interessant ja que es basa en la propagació unitària. Més de 50 anys després, aquest algorisme encara és la vela major dels *SAT-solvers* actuals.

L'algorisme està basat en l'esquema d'un backtracking.

[22] Dos literals complementaris són el literal sense negar i el mateix literal negat.

```
def DPLL(F):
    if F.nomes_te_clausules_unaries():
        return F  # 'satisfiable'
    if F.conte_alguna_clausula_buida():
        return 'insatisfactible'
    for clausula in F if len(clausula) == 1:
        F = propagacio_unitaria(clausula, F)
    for literal in F.literals_purs():
        F = assigna_literal_pur(literal, F)
    literal = escull_literal_lliure(F)
    return DPLL(F & literal) or DPLL(F & neg(literal))
```

on **propagacio_unitaria**$(\mathbf{c}, \mathbf{f})$ és una funció que retorna el resultat d'aplicar propagació unitària de **c** sobre **f** i **assigna_literal_pur**$(\mathbf{l}, \mathbf{f})$ és una funció que assigna al literal pur[23] **l** el valor de veritat amb què apareix a la formula i propaga el resultat.

L'algorisme es molt depenent de l'ordre en que s'escullen els literals a l'hora de decidir-ne el valor (funció *escull_literal_lliure*(F)). En aquest aspecte no se sap de cap estratègia clarament superior a la resta i en general s'apliquen heurístiques a l'hora d'optimitzar la tria de literals lliures, en l'intent d'afavorir el màxim la propagació unitària. Com a norma general (*rule of thumb*) s'intenta afavorir aquells literals que apareixen amb major freqüència a la formula sobre d'altres que potser apareixen de manera més aïllada. Però, altre cop, res garanteix que aquesta estratègia resulti sempre la més efectiva.

5.3.2 Conflict-Driven Clause Learning (CDCL)

El Conflict-Driven Clause Learning és un refinament que s'aplica conjuntament amb l'algorisme DPLL. De fet té com a objectiu principal evitar el *trashing*[24] de l'esquema de backtracking del DPLL. La idea de base és enriquir la teoria amb noves clàusules que impedeixin la repetició d'assignacions inconsistents en passos de resolució futurs.

Imaginem un cas ideal on durant la resolució d'una formula proposicional booleana **F** arribem a una contradicció. Suposem també, que sabem que tal contradicció s'ha produït per l'assignació de tres variables concretes (que són un subconjunt de l'assignació parcial) $x_a, x_b, x_c \subset variables(F)$ i $I(x_a), I(x_b), I(x_c) \subset I$, on I és l'assignació parcial que havíem fet des de l'inici de la resolució fins arribar a la contradicció.
Suposem:

$$I(x_a) = Cert,\ I(x_b) = Fals,\ I(x_c) = Fals$$

[23] Un literal pur es aquell literal que sempre apareix de la mateixa forma a la formula. O bé apareix sempre negat o sempre sense negar.

[24] En un algorisme de backtracking, es denomina *trashing* al fenòmen que es produeix quan l'esquema de resolució repeteix de manera continuada assignacions de variables que se sap que són inconsistents amb la teoria.

És fàcil veure que per impedir que l'assignació parcial de les variables x_a, x_b i x_c es torni a repetir, n'hi hauria prou afegint la següent clàusula c_1 a F.

$$c_1 = (\overline{x_a} \vee x_b \vee x_c)$$

Ara bé, com ja hem dit aquest era un cas ideal, ja que sabíem que de totes les variables de l'assignació parcial, el conflicte només el causaven les variables x_a, x_b i x_c. A la pràctica no és aconsellable aprendre clàusules que impedeixin la assignació concreta de les variables implicades per dos motius, per una banda el nombre de variables de decisió implicades pot ser gran, fet que provocaria la introducció de clàusules grans que rarament dispararien la resolució unitària, fet que li restaria eficàcia, per altra banda també s'ha de controlar el nombre de clàusules que s'aprenen ja que se'n poden arribar generar un nombre exponencial. Per tant s'apliquen heurístiques per tal de seleccionar les clàusules que s'aprenen a cada conflicte.

A continuació veurem el mètode Chaff [11] d'aprenentatge de clàusules que usa el concepte de **Unit Implication Point (UIP)**. Per explicar com funciona vegem-ho millor amb un exemple[25].

Sigui ϕ una subformula d'una instància SAT:

$$\phi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

$$\phi = (x_1 \vee x_{31} \vee \overline{x_2}) \wedge (x_1 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\overline{x_4} \vee \overline{x_5}) \wedge (x_{21} \vee \overline{x_4} \vee \overline{x_6}) \wedge (x_5 \vee x_6)$$

Suposem que durant la resolució es decideix $x_{21} = 0@2$ i $x_{31} = 0@3$. És a dir que la segona decisió que pren el *solver* és que la variable x_{21} s'avaluï a $Fals$ i la tercera decisió que pren és que la variable x_{31} s'avaluï també a $Fals$. També es decideix $x_1 = 0@5$. Podem suposar que la primera i quarta decisions es realitzen sobre variables que no apareixen a ϕ. Gràcies al graf d'implicacions de la figura 14 podem observar que les tres decisions ens porten a una contradicció ja que la clàusula ω_6: $(x_5 \vee x_6)$ es torna insatisfactible.

[25] L'exemple s'extreu directament del paper [12]

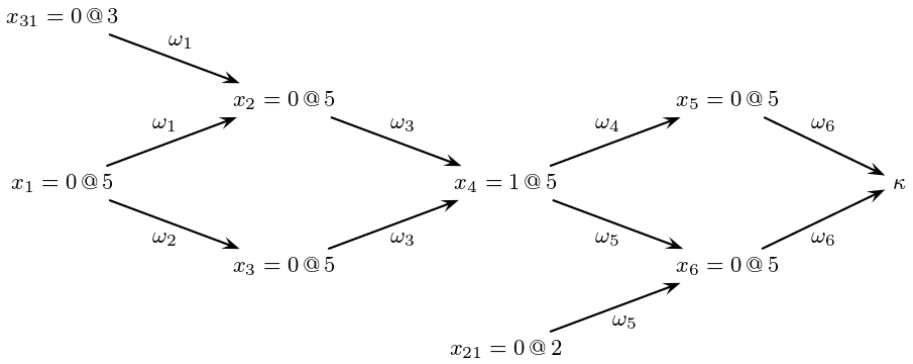

Figura 14: Graf d'implicacions per la formula ϕ

Del graf d'implicacions i, aplicant resolució podem arribar a generar la clàusula que impedeixi realitzar l'assignació de variables que ha provocat el conflicte. Només cal aplicar la regla de resolució partint de la clàusula que ha generat el conflicte amb les clàusules que intervenen al graf d'implicacions fins a obtenir una clàusula que només contingui literals que s'han assignat de manera arbitrària. A continuació il·lustrem el procediment continuant amb l'exemple de la formula ϕ.

$(x_5 \vee x_6)$	Clàusula on s'ha generat el conflicte (ω_6)
$(\overline{x_4} \vee x_6)$	Resolució amb ω_4
$(\overline{x_4} \vee x_{21})$	Resolució amb ω_5
$(x_2 \vee x_3 \vee x_{21})$	Resolució amb ω_3
$(x_1 \vee x_3 \vee x_{21} \vee x_{31})$	Resolució amb ω_1
$(x_1 \vee x_{21} \vee x_{31})$	Resolució amb ω_2
$(x_1 \vee x_{21} \vee x_{31})$	Stop. Tots els literals són de decisió

Taula 1: Passos de resolució durant l'aprenentatge de clàusules

Totes les clàusules que s'han generat a través de la regla de resolució compleixen que tot model de ϕ també ho és de $\{\phi \cup c_i\}$ on c_i és qualsevol clàusula de la taula 1. De fet aquesta propietat es manté per qualsevol formula φ i qualsevol clàusula C que s'obtingui mitjançant la regla de resolució sobre les clàusules de φ.
Com ja hem dit, però, introduir la última de les clàusules pot no ser el més eficient per afavorir la propagació unitària i introduir totes les clàusules no és raonable atesa la complexitat espacial que suposa un nombre exponencial de conflictes (en el pitjor cas). Aquí és on pren rellevància el UIP ja que suposa un refinament sobre la clàusula que s'aprèn de manera que s'afavoreixi al màxim la propagació unitària.

- **Unit implication Point (UIP)**

Un UIP és un node dominant[26] del graf d'implicacions. Bàsicament representa un punt de decisió alternatiu al nivell de decisió actual. I és útil sobretot perquè al formar part del camí de l'últim node de decisió, es possible que també domini altres nodes de decisió anteriors, permetent així reduïr el tamany de les clàusules apreses, que és l'objectiu últim dels UIP. En l'exemple anterior, és fàcil veure que la vàriable x_4 és un UIP ja que domina l'últim node de decisió x_1. I afortunadament també domina un node de decisió d'un pas anterior, el de la variable x_{31}. per tant l'algorisme de resolució s'aturaria amb la clàusula $(\overline{x_4} \vee x_{21})$, que és una clàusula molt més eficient en termes de propagació unitària ja que qualsevol combinació que faci certa x_4 dispararà la propagació unitària. Això no només inclou la combinació puntual de $I(x_1) = Fals$ i $I(x_{31}) = Fals$ sinó que també recull d'altres combinacions que de l'altre manera no haguéssim pogut evitar.
Existeixen algorismes de complexitat lineal per identificar els UIP's d'un graf d'implicacions, per tant, aplicar la tècnica no suposa cap contrapartida per al temps de resolució del *solver*.

Les instàncies SAT acostumen a ser molt grans, amb un nombre elevat de clàusules i variables. No obstant, resultats empírics evidencien que els *SAT-solvers* basats en CDCL són especialment eficients en la resolució d'instàncies industrials[27]. CDCL és la punta de llança de la tecnologia en resolució d'instàncies SAT, ja que com hem dit aquesta tècnica és especialment efectiva per a problemes que amaguen algun tipus d'estructura[28], que són gran part dels problemes reals. La idea rere el CDCL és evitar que el *solver* repeteixi errors, un dels principals problemes dels algorismes basats en *backtracking*, aprenent de cadascun dels errors que es cometen durant la resolució. Però com ja hem comentat, el nombre de conflictes té complexitat exponencial en el pitjor cas, per tant no és raonable aprendre clàusules a cada conflicte. Tot i que ja hem vist que és molt interessant fer-ho. Per tal de permetre l'aprenentatge de clàusules gestionant l'espai utilitzat, s'aplica una tècnica que consisteix a, de tant en tant, oblidar/-netejar algunes de les clàusules que s'han aprés anteriorment. Això pot semblar un pèl contraproduent, però com ja hem dit, no és raonable mantenir totes les clàusules que es van aprenent.

- **Clause Cleaning**

  De totes les millores i refinaments aplicats sobre el DPLL original, aquest camp és potser el que més marge de millora té. Resultats empírics evidencien que estratègies agressives de neteja de clàusules poden arribar a millorar notablement els temps de resolució dels *solvers*. La intuició dicta que si s'han d'oblidar clàusules millor oblidar les menys significatives, però

[26] En un graf dirigit, es diu que un node u domina un altre node x si qualsevol camí de x a un altre node k passa per u. En el graf d'implicacions un UIP u domina el punt de decisió x respecte del punt de conflicte k.

[27] Instàncies generades per un problema real

[28] Els *solvers* basats en CDCL no són tan eficients a l'hora de resoldre instancies artificials generades a l'atzar, per exemple.

per a tal efecte cal un mètode per identificar la qualitat de les clàusules apreses. Eliminar les de major tamany podría ser una bona idea, però un tamany menor no sempre indica una millor qualitat (podriem imaginar el cas extrem d'una clàusula amb dos literals aïllats sobre els quals el *solver* no decidís fins al cap de moltes decisions). Una altre bona estratègia seria la de fer un seguiment de les variables més actives[29] i eliminar primer aquelles clàusules que continguessin literals poc actius. O fins i tot una mescla de les dues estratègies anteriors.

En qualsevol cas, la neteja de la base de dades de clàusules apreses és una necessitat i una bona idea, ja que a major nombre de clàusules major es el cost de les operacions de propagació sobre el conjunt de clàusules.

5.3.3 Restarts

La idea consisteix en permetre al *solver* tornar a començar des del principi si al cap d'un temps intueix que ha arribat a un *cul de sac*. És a dir si "creu" que ha comès algun error en alguna de les assignacions inicials i per tant cap de les següents decisions que prengui el duran a un model fins que no es canvii l'assignació inicial. No oblidem però que als *solvers* se'ls exigeix completesa, per tant tard o d'hora hauran d'explorar la branca que decideixen abandonar (si no troben cap solució abans, és clar).
Bé doncs, per sorprenent que sembli, resultats empírics evidencien que incorporar aquest tipus d'estratègia als *solvers* redueix dramàticament la resolució d'instàncies satisfactibles[30], mentre que no penalitza massa aquelles que no ho són.

L'aspecte clau dels *restarts* és que *solver* no s'oblida del que ha après, només refà les decisions de variables, que amb el nou coneixement que ha adquirit des de l'ultim *restart* serà amb tota probabilitat diferent a la successió de decisions que havia efectuat en estadis anteriors. En concret el *solver* manté les clàusules apreses i informació sobre l'activitat de les variables; començant a decidir primer les variables més actives ja que amb les clàusules apreses li permetran propagar més.

5.4 MaxSAT

MaxSAT és el problema que consisteix a determinar el màxim nombre de clàusules satisfactibles d'una fòrmula proposicional booleana en CNF. És l'equivalent a trobar el mínim núcli de clàusules insatisfactible o *minimal unsatisfiable core*. Que es defineix com el conjunt de clàusules insatisfactible d'una fòrmula tal que qualsevol subconjunt seu és satisfactible. El problema és NP-Complet ja que és

[29] Una variable activa és una variable sobre la qual el *solver* hi ha aplicat algun tipus de raonament recentment, ja sigui una decisió o una propagació unitària.
[30] Que tenen solució

una generalització de SAT. De fet, MaxSAT amaga un *SAT-Solver* a mode d'oracle, on un algorisme superior optimitza el nombre de clàusules satisfactibles a base de crides a l'oracle.

En general diferenciem entre tres tipus de *solvers* per a MaxSAT:

1. Solvers basats en satisfactibilitat

   Sempre es mouen amb instàncies satisfactibles fins a trobar la primera insatisfactible, moment en el qual s'aturen. Comencen permetent que totes les clàusules es violin i a mesura que van trobant models, minimitzen el nombre de clàusules violades. La característica principal és doncs, que sempre tenen a punt la última solució que han trobat, tot i que pugui estar allunyada de l'òptima. Indicats per a instàncies dures, on el cost del certificat d'optimalitat sigui prohibitiu.

   Alguns solvers: SAT4j, QMaxSat.

2. Solvers basats en insatisfactibilitat

   Al contrari que les anteriors, aquestes sempre es mouen amb instàncies insatisfactibles fins que en troben una de satisfactible, moment en el qual s'aturen. Es basen en *cores* d'insatisfactibilitat. Comencen imposant que totes les clàusules es satisfacin, com si es tractés d'una instància SAT. Un cop certifiquen que és insatisfactible, n'extreuen un *core* d'insatisfactibilitat i repeteixen el procés però ara permetent que es violi una de les clàusules del *core*. Aquest procés es repeteix fins arribar a trobar un model que satisfaci totes les clàusules. Actualment se sap de *solvers* que apliquen tècniques més refinades i eficients. Indicats per problemes que requereixin certificat d'optimalitat.

   Alguns solvers: msuncore, WPM1, PM2.

3. Solvers basats en un esquema d'optimització branch and bound

   Són un hibrid entre els dos tipus de solvers anteriors. Si bé utilitzen técniques de relaxació a l'hora d'estimar la bondat de cada branca per tal de minimitzar el nombre de crides a l'oracle. Es podría dir que el seu principal objectiu és precisament aquest, el de minimitzar el nombre de crides.

   Alguns solvers: Clone, MaxSatz, IncMaxSatZ, IUT_MaxSatz, WBO, GIDH-Sat

   Hi han diverses extensions al problema MaxSat pur:

   - Partial MaxSAT
     S'introdueix el concepte de *soft-clauses* i *hard-clauses*, fent referència a clàusules que poden ser violades i clàusules que no poden violar-se respectivament. En aquest sentit, es maximitza el nombre de *soft-clauses* que se satisfant juntament amb totes les *hard-constraints*.

- Weighted MaxSAT
  S'introdueix la noció de pes a les clàusules. De manera que les clàusules amb major pes, són més "cares" de violar que les de menys pes.
- Partial Weighted MaxSAT
  Combinació de les dues anteriors. Conté *hard-clauses* i *soft-clauses* amb pes assignat. Acostuma a ser el més utilitzat per al mapeig de problemes reals, ja que és més descriptiu a l'hora de formular el problema.

5.5 Satisfiability Modulo Theories (SMT)

SMT és una generalització de SAT on algunes de les variables proposicionals tenen el paper de predicats amb interpretacions predefinides a d'altres teories. Per exemple, la següent és una formula SMT: $p \vee q \vee (x < y) \vee (y \geq 5)$ on p i q són variables booleanes i x i y són variables enteres. La gràcia és que els predicats que no contenen variables booleanes s'avaluen d'acord amb una teoria subjacent. Existeixen diversos tipus de teories com ara la teoria d'igualtat, aritmètica lineal entera, aritmètica lineal mixta, arrays, vectors de bits. Com que cada predicat és avaluat per la corresponent teoria, es permeten combinacions de teories al llarg d'una formula.

Una *teoria* es defineix com un conjunt de formules de primer ordre tancades sota conseqüència lògica. És a dir, les formules s'han de poder reduir a un resultat booleà.

Llavors una instància SMT per una teoria T consisteix en, donada una formula de primer ordre F, determinar si existeix un model de $T \cup \{F\}$. És a dir, consisteix a trobar un model que satisfaci tant els predicats de T com la pròpia formula booleana F. Normalment es requereix que T sigui decidible i F estigui lliure de quantificadors, de manera que s'aconsegueix enriquir el llenguatge SAT sense comprometre'n la decidibilitat. Per tant, les instàncies SMT tendeixen a ser més fàcils de descriure i per tant, més amigables des del punt de vista del programador.

La idea principal a l'hora d'atacar aquest tipus de problemes és el d'integrar un textitSAT solver conjuntament amb un *solver* específic a cada teoria. Simplificant, podriem dir que el procés de resolució d'una instància SMT consisteix en el SAT solver prenent decisions sobre F a l'hora que el solver de teoría va comprovant que les assignacions que fa el SAT solver son viables. Posant un exemple, imaginem la formula F següent:

$$F = (x \geq 0) \wedge (x < y) \wedge (y < 0 \vee y > 10)$$

el SAT solver ho veuría com:

$$F = (P1) \wedge (P2) \wedge (P3 \vee P4)$$

on $P1: \ x \geq 0$, $P2: \ x < y$, $P3: \ y < 0$ i $P4: \ y > 10$
Per satisfer la primera clàusula el sat solver només té una opció: $X \geq 0$, per tant li comunica al solver de teoria el que $x \geq 0$. En aquest punt el solver de teoria comprova que sigui viable assignar valor a x sense violar la restricció que li ha passat el SAT solver. Òbviament existeixen valors de x que satisfan la restricció, per tant continua la resolució.
A continuació el SAT solver es troba amb la clàusula $(x < y)$. Igual que abans, per que es satisfaci la clàusula només té una opció, que $x < y$. Per tant li comunica el que ha inferit al solver de teoria. Aquest repeteix el procés i s'assegura que existeixi alguna assignació per x i y que satisfaci les dues restriccions. En aquest cas també existeixen valors de x i y que satisfan les restriccions, per tant continua la resolució.
El SAT solver avalua la tercera clàusula. Recordem que el SAT solver no en té ni idea de les teories i que ell només veu $(P3 \vee P4)$. Suposem que pren la decisió de considerar que tant $P3$ com $P4$ s'avaluen a $Cert$, moment en el qual el solver de teoria es queixa i li comunica que amb les restriccions donades no és possible trobar cap assignació (és a dir model) de les variables x i y. En aquest moment el SAT solver faria backtracking i intentaria satisfer només $P3$. El solver de teoria tornaria a queixar-se dient-li que no pot trobar cap assignació. el SAT solver tornaria a fer backtracking i comprovaria $P4$, que és la última possibilitat que té de satisfer la tercera clàusula. En aquest punt el solver de teoria sería capaç de trobar una assignació.

Un cop satisfetes totes les clàusules el SAT solver ens donaira el seu model I per a F:

$$I(P1) = Cert, \ I(P2) = Cert, \ I(P3) = Fals, \ I(P4) = Cert$$

Després només queda preguntar al solver de teoria que realitzi una assignació per al model. Una possible assignació sería $x = 0$ i $y = 11$.

Òbviament aquest ha estat un exemple molt simplístic i un SMT solver real pot no prendre les decisions de manera tant arbitrària. Però l'exemple serveix per il·lustrar la idea rere SMT.

6 Cardinality Encodings

Els operadors lògics de disjunció, conjunció i negació ens proporcionen una alta expressivitat a l'hora de codificar les restriccions d'un problema a una instància SAT o SMT. En el cas de SMT encara és més cert tenint en compte que també incorpora operadors relacionals. Però en el cas concret de SAT i dels Bit-Vectors de SMT, la codificació de constraints que parlin de la cardinalitat d'algun conjunt de variables no és directe.

En aquesta secció enumerarem el diferents *cardinality encodings* que s'han implementat en la confecció d'aquest treball fi de grau. Diem *encoding* a aquella tècnica que ens permet transcriure restriccions d'un paradigma a un altre. I *cardinality encoding* aquelles tècniques que permeten transcriure restriccions de l'estil $|\mathbf{A}|\ \#\ \mathbf{n}$, on $\mathbf{A}$ és un conjunt de variables, $\#$ és un operador del conjunt $\{<, \leq, =, \geq, >\}$ i $\mathbf{n} \in \mathbb{N}^{\mathbf{0}}$.

En aquest treball hem analitzat i implementat encodigns per a dos grups:

1. Encodings de cardinalitat per a vectors de bits (SMT)

   Totes aquelles tècniques que ens permeten codificar restriccions sobre el nombre de bits "actius" d'un vector de bits. En aquesta secció no considerem codificacions per a l'implementació amb Pseudo-Booleans (teoria aritmètica lineal entera) ja que són directes.

2. Encodings de cardinalitat per a SAT

   Comprenen aquelles tècniques que ens peremeten traduïr restriccions formulades en llenguatge natural a un conjunt de clàusules CNF que capturin l'essència de les restriccions.

6.1 Bit Vector Cardinality Encodings

Codificant instàncies a SMT amb vectors de bits com a teoria, sovint ens trobarem amb la necessitat d'imposar restriccions sobre el nombre de bits "activats" d'un bit vector (bits amb valor 1 d'un vector). És important tenir present el tipus d'operadors que tenim disponibles. Un vector de bits és exactament el que el nom indica, un vector de n bits, on n és un nombre arbitràri de bits. Cal tenir present que tots els operadors de la teoria requereixen que el nombre de bits de cada operand sigui el mateix. Per una banda tenim operadors relacionals: $=$, $\neq$, $<$, $\leq$, $>$, $\geq$. Per altra banda tenim operadors booleans: $\sim$ (negació), & (*and*), | (*or*), tots ells funcionen bit a bit. També tenim a disposició operadors aritmètics bàsics: $+$, $-$. Cal destacar que els operadors aritmètics consideren els vectors de bits com la representació binària en complement a 2 d'un enter (sempre que no s'especifiqui el contrari). Per últim també tenim disponibles operacions de desplaçament: $\ll n$ (desplaçament a l'esquerra de n bits) o $\gg n$ (desplaçament a la dreta de n bits).

La codificació de les restriccions sobre la cardinalitat en el nombre de bits "actius" d'un vector, de l'estil $card(X) \leq k$ s'ha realitzat aplicant un mètode iteratiu sobre k. La idea es definir una funció que donat un vector de bits X d'entrada, generi un vector de bits S de sortida que tingui exactament un (1) bit "actiu" menys. A aquesta funció la anomenarem *reduce* ja que redueix el nombre de bits "actius" del vector d'entrada.

Funció reduce

$$reduce(X) = X\&(X-1)$$

Exemple:

	X =	0	1	1	1	0
+	-1 =	1	1	1	1	1
	(X - 1) =	0	1	1	0	1
	X & (X-1) =	0	1	1	0	0

Es pot comprovar doncs, que si $X > 0$ $reduce(X)$ sempre produeix un vector amb exactament un bit "actiu" menys que X. Una altra propietat important, és que si $X = 0$, $reduce(X) = 0$.

Llavors, per codificar $|X| \leq k$, n'hi ha prou amb:

$$reduce_k(reduce_{k-1}(reduce_{...}(reduce_1(X)))) = 0$$

En el cas que la restricció sigui *soft*, i per tal d'efectuar la codificació de pesos, ens quedarem amb els resultats intermitjos e imposarem que siguin diferents de 0, en cas de desitjar codificar un at_least_k. En cas de volguer codificar un at_most_k seguirem aplicant reduccions fins on creiem oportú i a cadascuna d'aquestes reduccions hi imposarem que sigui igual a 0.

6.2 SAT Cardinality Encodings

Com ja hem dit, els cardinality encodings són aquelles tècniques que permeten transcriure restriccions de l'estil $|\mathbf{A}| \ \# \ \mathbf{n}$, on $\mathbf{A}$ és un conjunt de variables, $\#$ és un operador del conjunt $\{<, \leq, =, \geq, >\}$ i $\mathbf{n} \in \mathbb{N}^{\mathbf{0}}$ a SAT.

Com pasava amb SMT, n'hi ha prou amb codificar encodings per l'operador $\leq$ ja que a partir d'aquest, és senzill obtenir tota la resta. Els encodings que veurem a continuació sempre fan refererència a l'encoding d'una restricció de l'estil $x_1 + \cdots + x_n \leq k$, amb $k < n$. Aquest tipus de restriccions també s'anomenen *at_most_k* constraints.

A l'hora d'analitzar els diferents tipus d'encodings s'han de tenir en compte dos aspectes de gran importància:

1. El nombre de clàusules i variables que generen.

   Com hem dit una tècnica d'encoding no és més que una manera de transcriure restriccions a clàusules CNF. Interessa però que aquesta transcripció sigui el més eficient possible. Eficient en termes del nombre de clàusules i/o variables auxiliars que introdueixi l'encoding a la formula. Com a regla general, els encodings que permeten capturar l'essència d'una restricció en menor nombre de clàusules i variables són més eficients.
   Ara bé, el nombre de variables auxiliars i clàusules que es generen acostumen a ser inversament proporcionals i per tant no sempre és senzill determinar de manera un-equívoca quin encoding és més eficient i sovint fa falta comparar-los empíricament.

2. Mantenen l'arc-consistència de la restricció original.

   El concepte d'arc-consistència aplicat a un encoding de cardinalitat es defineix com al fet que, l'encoding garanteix que tan bon punt k variables de l'entrada tenen assignat el valor de veritat $Cert$, la propagació unitària s'encarrega d'assignar a la resta de variables el valor de veritat $Fals$. De manera que el *solver* mai podrà prendre una decisió que violi la restricció de manera directa. No cal dir que aquesta propietat és importantíssima, i és per això que nosaltres només hem considerat encodings que garanteixen l'arc-consistència.

En el cas de SAT existeix una codificació directa d'aquest tipus d'encodings. El problema és que pot generar un nombre prohibitiu de clàusules i cal doncs, buscar encodings alternatius que siguin més eficients en termes de clàusules i variables auxiliars generades. A continuació enumerem diversos tipus d'encodings.

6.2.1 Direct Encoding

Una manera simple de codificar un *at_most_k* consisteix a imposar que per cada subconjunt de $k+1$ variables del total, com a mínim una ha de ser falsa. És a dir per tot $Y \subset \{x_1, ..., x_n\}$, tal que $|Y| = k+1$, com a mínim una variable de Y ha de ser falsa, de manera que mai hi podran haver k+1 (o més) variables a cert. A la literatura també rep el nom de *naive approach*, *combinatorial encoding*.

Seguint aquest procediment s'introdueixen $\binom{n}{k+1}$ clàusules del tipus $(\overline{x_{i_1}} \vee ... \vee \overline{x_{i_{k+1}}})$. Aquest encoding funciona bé per a valors de n i k petits ja que no requereix de cap variable auxiliar. Però obviament no és raonable per a valors grans de n i/o k.

6.2.2 Totalizer Encoding

El Totalizer encoding es basa en generar una representació unària de la cardinalitat del conjunt de variables d'entrada. Fou introduït per Bailleux i Bougkhad [13]. L'encoding es basa en l'estructura d'un arbre binari on a les fulles hi han

les variables d'entrada i cada nus interior, conté la representació unària de la cardinalitat dels seus descendents fulla. Per tant amb tantes variables auxiliars com descendents fulla tingui.

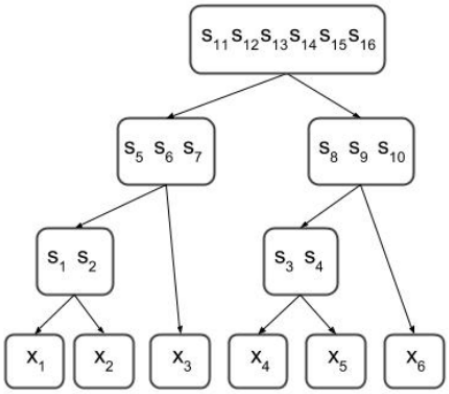

Figura 15: Representació gràfica de l'estructura d'un Totalizer

Seguint l'exemple de la figura 15, tindríem com a entrada el conjunt de variables $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ i com a sortida el conjunt de variables $\{s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}\}$. El detall important, és que cada node codifica la cardinalitat de les fulles que en pengen, en format unari. Que significa que per tot i de cada node, $s_i \geq s_{i+1}$.

En la figura es veuen totes les variables auxiliars que generaria l'encoding per codificar un *at_most_k* sobre 6 variables. Per veure quines clàusules genera, cal definir un sumador unari. Que donats dos nombres unaris, en retorni la suma (també en unari). Aplicant aquest sumador a cada node interior n'hi ha prou per garantir que $\{s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}\}$ sigui la representació unària de la cardinalitat de $\{x_1, x_2, x_3, x_4, x_5, x_6\}$. Un cop tenim la representació unària de la cardinalitat, n'hi ha prou amb imposar que $I(out_{k+1}) = Fals$, per fer efectiva la restricció inicial.

```
def Totalizer(a):
    n = len(a)
    if n == 1:
        return a[0]
    esquerra = Sorter(a[:n/2])
    dreta = Sorter(a[n/2:])
    return UA(esquerra, dreta)
```

Figura 16: Algorisme Totalizer Encoding

Sumador Unari (UA)

Sigui $\langle a_1, ..., a_n \rangle$ i $\langle b_1, .., b_m \rangle$ dos representacions unàries d'entrada i $\langle r_1, ..., r_{n+m} \rangle$ la representació unària de $\langle a \rangle + \langle b \rangle$.
Llavors, $UA(\langle a_1, ..., a_n \rangle, \langle b_1, .., b_m \rangle) = (\langle r_1, ..., r_{n+m} \rangle, \phi)$ on,

$$\phi = \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{m} (\overline{a_i} \vee \overline{b_j} \vee r_{i+j}) \qquad a_0 = b_0 = r_0 = Cert$$

Les variables $a_0, b_0 i r_0$ s'han afegit per simplificar l'expressió de la formula, però no es difícil efectuar la implementació del sumador sense l'ajut d'aquestes variables.

L'encoding Totalizer genera $O(n\ log\ n)$ variables auxiliars i $O(n^2)$ clàusules.

6.2.3 Simplified Totalizer

Un refinament del totalizer ve de l'observació que de les variables de sortida $\{s_1, ..., s_n\}$ només ens interessen les $k+1$ primeres ja que el que fem és imposar que la variable s_{k+1} s'evalui a fals afegint la clàusula $\phi \cup (\overline{s_{k+1}})$ a la formula. Per tant les variables $\{s_{k+2}, ..., s_n\}$ no ens interessen per res i podem eliminar totes les clàusules i variables auxiliars que en pengen.

A la pràctica per treure profit de l'anterior raonament n'hi ha prou amb fer una petita modificació al nostre UA. Així que l'algorisme del Simplified Totalizer es manté igual a excepció que aquest utilitza la versió simplificada del sumador unari.

Sumador Unari Simplificat (SUA)

$$SUA(\langle a_1, ..., a_n \rangle, \langle b_1, .., b_m \rangle, k) = (\langle r_1, ..., r_{min(n+m,k)} \rangle, \phi)$$

on,

$$\phi = \bigwedge_{i=0}^{min(n,k)} \bigwedge_{j=0}^{min(m,k)} (\overline{a_i} \vee \overline{b_j} \vee r_{i+j}) \qquad a_0 = b_0 = r_0 = Cert$$

Fixem-nos que el sumador només genera nombres de mida k, per tant serveix per codificar restriccions de l'estil $|A| < k$. Però no és difícil adaptar-ho per a poder codificar restriccions de l'estil *at_most_k*. De fet només cal augmentar en 1 la k de l'algorisme.

L'encoding del Simplified Totalizer genera $O(nk)$ clàusules i $O(n\ log\ k)$ variables.

6.2.4 Sorting Networks

Aquest encoding es basa en dues idees, per una banda la idea de que tota formula proposicional booleana es pot traduir a un circuit electrònic i vice-versa,en aquest sentit adopta l'estructura d'un comparador de dos bits com a unitat bàsica de l'encoding. La segona i més rellevant és que s'intenta ordenar les variables de l'entrada mitjançant un esquema de *merge-sort*, aprofitant el comparador de dos bits.

Comparador de dos bits:

$$comparador\_binari(a,b) = (\langle s1, s2\rangle, (\overline{a} \vee \overline{b} \vee s_2) \wedge (\overline{a} \vee c_1) \wedge (\overline{b} \vee s_1))$$

Notis que al codificar restriccions de l'estil *at_most_k* no ens cal incloure les clàusules $(a \vee \overline{s_2}) \wedge (b \vee \overline{s_2}) \wedge (a \vee b \vee \overline{s_1})$ ja que aquestes recullen el comportament de la sortida quan les variables de l'entrada s'avaluen a $Fals$. És a dir que a nosaltres només ens interessa restringir el nombre de variables de l'entrada que s'avaluen a $Cert$ de manera simultànea, no pas les que s'avaluen a $Fals$.

Els sorting networks tenen dos blocs principals, un per a les funcions d'ordenació (*sort*) i l'altre per efectuar les fusions de vectors ordenats. Per a ordenar s'aplica el mètode del *merge-sort* que consisteix en anar partint l'entrada fins que aquesta ja no es pot partir més. Moment en què es fusionen els trossos fins a reconstruïr l'entrada ja ordenada.

```
def Sorter(a):
    n = len(a)
    if n == 1:
        return a[0]
    esquerra = Sorter(a[:n/2])
    dreta = Sorter(a[n/2:])
    return Merge(esquerra, dreta)
```

Figura 17: Algorisme d'un Sorter

La funció *Merge* de la figura 17 rep dos vectors ordenats i els fusiona. Funciona de manera que:

$$Merge(\langle a_1, ..., a_n\rangle, \langle b_1, ..., b_n\rangle) = (\langle d_1, c_2, ..., c_{2n-1}, e_n\rangle, \phi_1 \wedge \phi_2 \wedge \phi_3)$$

on

$$(\langle d_1, ..., d_n\rangle, \phi_1) = Merge(\langle a_1, a_3, ..., a_{n-1}\rangle, \langle b_1, b_3, ..., b_{n-1}\rangle),$$

$$(\langle e_1, ..., e_n\rangle, \phi_2) = Merge(\langle a_2, a_4, ..., a_n\rangle, \langle b_2, b_4, ..., b_n\rangle),$$

$$(\langle c_1, ..., c_{2n-1}\rangle, \phi_3) = \bigwedge_{i=1}^{n-1} (\overline{d_{i+1}} \vee \overline{e_i} \vee c_{2i+1}) \wedge (\overline{d_{i+1}} \vee c_{2i}) \wedge (\overline{e_i} \vee c_{2i})$$

si $|a| = |b| = 1$, llavors:

$$Merge(\langle a\rangle, \langle b\rangle) = (\langle c_1, c_2\rangle, (\overline{a} \vee \overline{b} \vee c_2) \wedge (\overline{a} \vee c_1) \wedge (\overline{b} \vee c_1))$$

Fixem-nos que tant ϕ_3 com les clàusules generades en el cas de $Merge(\langle a\rangle, \langle b\rangle)$ corresponen a un comparador de dos bits. Un aspecte a destacar es que el merge també funciona de manera recursiva separant els vectors d'entrada en parts senars i parells. També, en l'exemple es presenta la primera versió de les *sorting networks* que requereix que el nombre de variables d'entrada sigui una potència de dos. Actualment ja existeixen versions d'aquest mateix esquema que permeten un nombre arbitrari de variables d'entrada, però per simplicitat hem il·lustrat el primer ja que la nova versió és essencialment el mateix però amb major casuística.

L'encoding basat amb Sorting Networks genera $O(n\ log^2\ n)$ variables auxiliars i $O(n\ log^2\ n)$ clàusules.

6.2.5 K-Cardinality Network

De manera anàloga al Simplified Totalizer també existeix un refinament per a les sorting networks basat en el mateix principi. En aquest cas cal revisar la implementació de la funció de fusió.

Simplified Merge

$$SMerge(\langle a_1, ..., a_n\rangle, \langle b_1, ..., b_n\rangle, k) = (\langle d_1, c_2, ..., c_k\rangle, \phi_1 \wedge \phi_2 \wedge \phi_3)$$

on

$$\begin{aligned}
(\langle d_1, ..., d_{\frac{n}{2}+1}\rangle, \phi_1) &= SMerge(\langle a_1, a_3, ..., a_{n-1}\rangle, \langle b_1, b_3, ..., b_{n-1}\rangle, k),\\
(\langle e_1, ..., e_{\frac{n}{2}+1}\rangle, \phi_2) &= SMerge(\langle a_2, a_4, ..., a_n\rangle, \langle b_2, b_4, ..., b_n\rangle, k),\\
(\langle c_1, ..., c_k\rangle, \phi_3) &= \bigwedge_{i=1}^{k/2} (\overline{d_{i+1}} \vee \overline{e_i} \vee c_{2i+1}) \wedge (\overline{d_{i+1}} \vee c_{2i}) \wedge (\overline{e_i} \vee c_{2i})
\end{aligned}$$

si $|a| = |b| = 1$, llavors:

$$Merge(\langle a\rangle, \langle b\rangle) = (\langle c_1, c_2\rangle, (\overline{a} \vee \overline{b} \vee c_2) \wedge (\overline{a} \vee c_1) \wedge (\overline{b} \vee c_1))$$

Llavors, per el cas on $k \leq n$, és a dir on l'entrada sigui més petita o igual que el màxim de la restricció, s'aplica la funció Merge de l'apartat anteriror. En canvi quan l'entrada és més gran que el màxim de la restricció ($k > m$) s'aplica el *Simplified Merge*. De manera que restringim la sortida d'aquesta codificació a ser menor o igual al màxim de la restricció (és a dir a **k**).

L'encoding del K-Cardinality Networks genera $O(n\ log^2\ k)$ variables auxiliars i $O(n\ log^2\ k)$ clàusules

6.2.6 Modulo Totalizer (MTO)

És un refinament sobre l'encoding basat en el Totalizer. Si comparem els encodings del Totalizer i el basat amb Sorting Networks, el totalizer té un camí crític[31] més curt, però el sorter genera un ordre de complexitat inferior de clàusules. Per tant aquest encoding té com a objectiu principal pal·liar l'alt nombre de clàusules que genera l'encoding original, i que el fa competitiu només per a cardinalitats petites.
Com ja hem vist, l'encoding original es basa en una representació unària de la cardinalitat. Bé doncs el Totalizer Modular, es basa en una representació unària i modular de la cardinalitat. Es divideix la representació en dues parts **q** i **r**, de tal manera que $n = q * p + r$, on n és la cardinalitat de les variables d'entrada, p és una constant modular i $r < p$. L'esquema d'un MTO és el mateix que l'encoding Totalizer, però utilitzant un sumador capaç de sumar representacions modulars. Per tant cal definir el comportament d'un sumador modular.

Sumador Modular Unari (MUA)

$$MUA(\langle f_1..f_\alpha|_p a_1..a_n\rangle, \langle g_1..g_\beta|_p b_1..b_m\rangle) = (\langle q_1..q_{\alpha+\beta+1}|_p r_1..r_{p-1}\rangle, \phi_1 \wedge \phi_2)$$

on

[31] El Camí crític d'un cardinality encoding és el nombre de pasos de propagació que cal dur a terme per mantenir l'arc-consistència de la restricció.

$$\phi_1 = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4$$

$$\omega_1 = \left(\bigwedge_{i=1}^{n} \overline{a_i} \vee r_i \vee c\right) \wedge \left(\bigwedge_{i=1}^{m} \overline{b_i} \vee r_i \vee c\right)$$

$$\omega_2 = \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m} \overline{a_i} \vee \overline{b_j} \vee r_{i+j} \vee c \qquad (i+j<p)$$

$$\omega_3 = \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m} \overline{a_i} \vee \overline{b_j} \vee r_{mod(i+j,p)} \qquad (i+j>p)$$

$$\omega_4 = \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m} \overline{a_i} \vee \overline{b_j} \vee c \qquad (i+j\geq p)$$

$$\phi_2 = \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge (\overline{c} \vee q_1)$$

$$\psi_1 = \bigwedge_{i=1}^{\alpha} (\overline{f_i} \vee q_i) \wedge (\overline{f_i} \vee \overline{c} \vee q_{i+1})$$

$$\psi_2 = \bigwedge_{i=1}^{\beta} (\overline{g_i} \vee q_i) \wedge (\overline{g_i} \vee \overline{c} \vee q_{i+1})$$

$$\psi_3 = \bigwedge_{i=1}^{\alpha}\bigwedge_{j=1}^{\beta} (\overline{f_i} \vee \overline{g_j} \vee q_{i+j}) \wedge (\overline{f_i} \vee \overline{g_j} \vee \overline{c} \vee q_{i+j+1})$$

Si ens fixem, veiem que ϕ_1 es correspon amb la part baixa (el residu) de la representació modular. En concret és un sumador unari que preveu els escenaris on la suma de certs bits pugui causar sobreeiximent (overflow) o no. D'aquí l'addició de la variable auxiliar **c**, que representa el bit de *carry* (bit de sobreeiximent). Per altra banda ϕ_2 es correspon amb la part alta de la representació modular (el quocient). És un sumador unari enriquit amb la possibilitat que hi hagi *carry* de la part baixa.

L'encoding del Modulo Totalizer genera $O(n^{3/2})$ clàusules i $O(n\ log\ n)$ variables. Per tant suposa una millora sobre l'encoding original, que té com a problema principal l'alt nombre de clàusules que genera.

6.2.7 Mixed Cardinality Network

Els encodings basats en esquemes recursius com els basats en *totalizers* i *sorting networks*, acostumen a generar un nombre elevat de variables auxiliars, sobretot si els comparem amb l'encoding directe, que no en genera cap. És fàcil veure que per a conjunts de variables petits, és molt millor utilitzar una codificació directe. Si ens fixem, tots els algorismes recursius arriben a un punt on han de codificar un nombre relativament petit de variables. Robert Nieuwenhuis et

al.[14] proposen una codificació directa per els algorismes de Merge, Sort, Simplified Merge que juntes permeten una codificació directa per les K-Cardinality Network.

Direct Merge

$$DMerge(\langle a_1..a_n\rangle, \langle b_1..b_m\rangle) = (\langle s_1..s_{n+m}\rangle, \phi)$$

Amb les següents clàusules[32]:

$$\phi = \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m}(\overline{a_i} \vee s_i) \wedge (\overline{b_j} \vee s_j) \wedge (\overline{a_i} \vee \overline{b_j} \vee s_{i+j})$$

Per tant, genera $n+m$ variables auxiliars i $n*m+n+m$ clàusules.

Direct Sorting Networks

$$DSort(\langle a_1..a_n\rangle) = (\langle s_1..s_n\rangle, \phi)$$

Amb les següents clàusules:

$$\phi = \bigwedge_{k=1}^{n}\bigwedge_{i_1=1}^{i_2-1}\cdots\bigwedge_{i_k=i_{k-1}+1}^{n}(\overline{a_{i_1}} \vee \overline{a_{i_2}} \vee \cdots \vee \overline{a_{i_k}} \vee s_k)$$

Per tant, genera n variables auxiliars i $2^n - 1$ clàusules.

Direct Simplified Merge

$$DSMerge(\langle a_1 \cdots a_n\rangle, \langle b_1 \cdots, b_m\rangle, k) = (\langle s_1 \cdots s_k\rangle, \phi)$$

Amb les següents clàusules[32]:

$$\phi = \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m}(\overline{a_i} \vee s_i) \wedge (\overline{b_j} \vee s_j) \wedge (\overline{a_i} \vee \overline{b_j} \vee s_{i+j}) \quad (i+j \leq k)$$

Per tant, genera k variables auxiliars i $(n+m)*k - \binom{k}{2} - \binom{n}{2} - \binom{m}{2}$ clàusules.

Direct K-Cardinality Network

De manera anàloga a la codificació recursiva, diferenciem entre si l'entrada menor o igual que k o major. En el primer cas, apliquem el Direct Merge. En cas que l'entrada sigui més gran apliquem el següent:

$$DKCard(\langle a_1 \cdots a_n\rangle, k) = (\langle s_1 \cdots s_k\rangle, \phi)$$

[32]L'expressió genera clàusules repetides, però per simplicitat suposem que només considerem un cop cada clàusula repetida.

Amb les següents clàusules[33]:

$$\phi = \bigwedge_{h=1}^{k} \bigwedge_{i_1=1}^{i_2-1} \cdots \bigwedge_{i_h=i_{h-1}+1}^{n} (\overline{a_{i_h}} \vee \overline{a_{i_2}} \vee \cdots \vee \overline{a_{i_h}} \vee s_h)$$

Per tant, genera k variables auxiliars i $\binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{k}$ clàusules.

La idea es barrejar les dues codificacions segons cada cas, quedant-nos només amb la millor part de cadascuna. Així, a cada pas de recursió s'avalua quina de les dues tècniques (si la directa o la recursiva) és més *eficient*, i s'aplica una tècnica o una altre en conseqüència. Òbviament en el moment en què s'aplica la codificació directa es talla la recursió. El concepte d'eficiència de la codificació rau en el nombre de clàusules i variables que es generen ja que ambdues codificacions preserven l'arc-consistència. Precisament a l'hora d'avaluar l'eficiència de les codificacions, interesarà minimitzar l'expressió $\lambda \cdot V + C$, on $\mathbf{V}$ és el nombre de variables auxiliars que introdueix la codificació, $\mathbf{C}$ el nombre de clàusules que genera i λ representa un coeficient ajustable que permet afegir un grau de llibertat en el comportament de la codificació. Segons si decidim donar prioritat a minimitzar variables o a clàusules.

6.2.8 Mixed Modulo Totalizer (versió pròpia)

Durant la realització d'aquest treball s'han implementat tant el Modulo Totalizer com el Mixed Cardinality Network. Un cop implementats els dos, vam comprovar que el Mixed Cardinality Network suposava una millora dràstica sobre la versió purament recursiva (la K-Cardinality Network). Revisant l'estructura del Modulo Totalizer es va veure que es podia aplicar el mateix principi que a les Mixed Cardinality Networks.

En concret, si recordem el que fem a el Modulo Totalizer és anar partint l'entrada fins que l'entrada és menor que el mòdul, moment en el qual s'invoca el Totalizer convencional. Recordem també que el Totalizer convencional introdueix un nombre elevat de clàusules i un nombre no tan elevat de variables. I que, generalment, el valor del mòdul acostuma a ser un nombre petit. Així que com a codificació alternativa, en aquest treball s'ha codificat el Modulo Totalizer de la manera convencional a excepció del cas on l'entrada és menor que el valor del mòdul. Moment a partir del qual s'invoca el *Direct Sorting* descrit en la codificació del Mixed Cardinality Network. Que introdueix un nombre elevat de clàusules però cap variable. Tenint en compte que el mòdul és un nombre petit, la codificació directa és més competitiva que la del Totalizer convencional.

A aquesta versió l'anomenem **Mixed Modulo Totalizer**.

6.2.9 At_Least_K Mixed Cardinality Network (versió pròpia)

Fins ara hem vist codificacions per a restriccions del tipus $\leq k$. Però què passa quan tenim una restricció de l'estil $\geq k$?. La solució directa passa per negar tots els literals de l'entrada i imposar que dels n literals d'entrada, només $n-k$ poden estar negats al mateix temps; imposant així que com a mínim k literals no estiguin negats, és a dir s'avaluin a $Cert$.

Per altra banda, un dels punts forts de les $CardinalityNetworks$ és que només tenen $k+1$ variables de sortida. Però és fàcil veure que si volem codificar un at_least_k llavors tindrem exactament $n-k+1$ variables de sortida. Per tant, per a valors de k petits estem utilitzant moltes variables més de les estrictament necessàries.

La solució que proposem passa per considerar millor quines clàusules afegim al comparador de dos bits segons la restricció que estiguem codificant. A continuació enumerem les clàusules de que hauría de constar un comparador de dos bits complert:

$$\begin{aligned} 2\_COMP(a,b) =& (\overline{a} \vee c_1) \wedge (\overline{b} \vee c_1) \wedge (\overline{a} \vee \overline{b} \vee c_2) \wedge \\ & (a \vee \overline{c_2}) \wedge (b \vee \overline{c_2}) \wedge (a \vee b \vee \overline{c_1}) \end{aligned}$$

Les tres clàusules de la primera fila, contemplen els casos on les variables d'entrada $a,\ b$ s'avaluen a $Cert$, mentre que les tres de sota contemplen els casos on les variables d'entrada s'avaluen a $Fals$. La idea és introduïr les clàusules que convigui segons la restricció que estiguem codificant. És a dir:

$$\begin{aligned} at\_most\_k:\ 2\_COMP(a,b) =& \quad (\overline{a} \vee c_1) \wedge (\overline{b} \vee c_1) \wedge (\overline{a} \vee \overline{b} \vee c_2) \\ at\_least\_k:\ 2\_COMP(a,b) =& \quad (a \vee \overline{c_2}) \wedge (b \vee \overline{c_2}) \wedge (a \vee b \vee \overline{c_1}) \\ exactly\_k:\ 2\_COMP(a,b) =& \quad ((\overline{a} \vee c_1) \wedge (\overline{b} \vee c_1) \wedge (\overline{a} \vee \overline{b} \vee c_2) \wedge \\ & \quad (a \vee \overline{c_2}) \wedge (b \vee \overline{c_2}) \wedge (a \vee b \vee \overline{c_1})) \end{aligned}$$

D'aquesta manera podem explotar el mateix principi en els at_least_k i fins i tot, implementant el comparador amb totes les seves clàusules reduïm a la meitat el nombre de variables generades per una restricció de l'estil $exactly_k$, que de manera natural es correspondria a la combinació d'un at_least_k i un at_most_k.

6.2.10 Conclusions i Comentaris

En aquesta secció s'han presentat varis models de codificacions que tenien per objectiu reduir el el tamany de la traducció d'una restricció de cardinalitat a SAT. Totes les codificacions, llevat de la directa, afegeixen variables auxiliars la qual cosa no és desitjable. No obstant l'enorme quantitat de clàusules que genera la codificació directa fa interessants la resta de codificacions. Seguint aquest

raonament és fàcil extreure que per a conjunts de variables petits la codificació directa sigui interessant mentre que a mesura que el conjunt de variables creix, s'han de començar a considerar codificacions més sofisticades.

En el problema HSTT que tractem en aquest treball, la majoria de restriccions de cardinalitat acostumen a involucrar un nombre reduït de variables. Però com que necessitem poder calcular el cost de la violació de certes restriccions, sempre ens és d'interès mantenir la sortida ordenada. És per això que per a la codificació del generador d'horaris s'ha optat per aplicar el Mixed Cardinality Network, ja que és el que millor eficiència ens proporciona.

6.3 SAT At-Most-One Encodings (AMO)

Un cas especial de les codificacions de cardinalitat apareix quan d'un conjunt de variables, es vol imposar que només una es pugui evaluar a $Cert$. Per descomptat podríem aplicar qualsevol de les codificacions vistes en la secció anterior. Però val la pena fer menció de dos codificacions alternatives que suposen una millora substancial en vers les vistes en l'anterior secció. Això si, només serveixen per a restriccions de l'estil $|A| \leq 1$.

6.3.1 Direct Encoding

També conegut com a *Pairwise Encoding*, és un cas particular del Direct encoding vist en la secció anterior. consisteix en imposar que per tota variable del conjunt d'entrada, si és certa llavors la resta han de ser falses. És a dir:

$$Direct\_AMO(\langle a_1 \cdots a_n \rangle) = \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\overline{a_i} \vee \overline{a_j})$$

Aquesta codificació genera 0 variables i $\binom{n}{2}$ clàusules.

6.3.2 Ladder Encoding

També conegut com a *Sequential Encoding*, La idea és representar les variables d'entrada en una estructura d'escala. De manera que les variables de l'escala respectin una senzilla regla entre elles: $(s_{i-1} \rightarrow s_i)$ i $(\overline{s_i} \rightarrow \overline{s_{i-1}})$. La següent expressió regular descriu l'estructura de les variables de l'escala 0^*1^*. És important veure doncs que només hi pot haver un sol punt on $s_{i-1} = 0,\ i\ s_i = 1$. Lligant les variables d'entrada amb les variables escala, podem imposar que si una variable d'entrada és certa, llavors la seva posició en l'escala ha de correspondre en aquell punt. D'aquesta manera com que a l'escala només hi pot existir un punt d'aquestes característiques, estem imposant que a les variables d'entrada només hi pugui haver una variable a cert; que és el que volíem des de l'inici.

Més formalment:

$$Ladder(\langle a_1 \cdots a_n \rangle) = \psi \wedge \left(\bigwedge_{i=2}^{n-1} (\overline{s_{i-1} \vee s_i}) \wedge (\overline{a_i} \vee s_i) \wedge (\overline{a_i} \vee \overline{s_{i-1}}) \right)$$
$$\psi = (\overline{a_1} \vee s_1) \wedge (\overline{a_n} \vee \overline{s_{n-1}})$$

El Ladder encoding genera $n-1$ variables i $3n-4$ clàusules.

6.3.3 Commander Encoding

Codificació introduïda per Klieber i Kwon a [15]. Consisteix en dividir el conjunt de variables d'entrada $\{x_1 \dots x_n\}$ en m conjunts disjunts $G_1 \dots G_m$ i, on per cada subconjunt G_i s'introdueix una variable auxiliar de control c_i. Després s'imposa que de la unió de les variables de cada subconjunt amb la negació de la seva variable de control, exactament una s'avaluï a $Cert$.

$$\bigwedge_{i=1}^{n} EO(\overline{c_i} \cup G_i) = \left(\bigwedge_{i=1}^{n} AMO(\overline{c_i} \cup G_i) \right) \wedge \left(\bigwedge_{i=1}^{n} ALO(\overline{c_i} \cup G_i) \right)$$

on EO és la codificació d'un *exactly one*, AMO la d'un *At Most One* i ALO la d'un *At Least One*. La codificació d'un ALO és trivial (Correspon a una clàusula amb la disjunció de totes les variables), mentre que la codificació de l'AMO pot ser qualsevol de les conegudes.
Daquesta manera cada lliguem les variables de control amb les variables de cada grup, de manera que quan una variable de control s'avaluï a $Cert$, significarà que exactament una de les variables del seu grup s'avaluen a cert. I quan una variable de control s'avaluï a $Fals$, que totes les variables del seu grup s'avaluïn a $Fals$. Un cop lligades les variables de control, només queda imposar que *com a molt una* de les variables de control es pugui avaluar a $Cert$.

$$AMO(\langle c_1 \dots c_m \rangle)$$

A continuació adjuntem un exemple per fer-ho més entenedor. Suposem el conjunt d'entrada $A = \{a_1 \dots a_9\}$. Decidim dividir el conjunt d'entrada en $m = 3$ conjunts disjunts: $G_1 = \{x_1, x_2, x_3\}$, $G_2 = \{x_4, x_5, x_6\}$ i $G_3 = \{x_7, x_8, x_9\}$. Per tant, introduïm 3 variables auxiliars c_1, c_2 i c_3.

1. Vinculem les variables auxiliars amb cada grup.

$$EO(\overline{c_1} \cup G_1) = AMO(\overline{c_1}, x_1, x_2, x_3) \wedge (\overline{c_1} \vee x_1 \vee x_2 \vee x_3)$$
$$EO(\overline{c_2} \cup G_2) = AMO(\overline{c_2}, x_4, x_5, x_6) \wedge (\overline{c_2} \vee x_4 \vee x_5 \vee x_6)$$
$$EO(\overline{c_3} \cup G_3) = AMO(\overline{c_3}, x_7, x_8, x_9) \wedge (\overline{c_3} \vee x_7 \vee x_8 \vee x_9)$$

   En aquest punt la idea és escollir una bona m, de manera que ens permeti generar subconjunts de tamany acceptable per a l'aplicació del *pairwise encoding* per tal de poder codificar els AMOs.

2. Imposem que com a molt una de les variables de control avaluï a $Cert$.

$$AMO(c_1, c_2, c_3)$$

En aquest pas depenent del nombre de variables de control, utilitzarem la codificació directa (*pairwise encoding*) o bé la codificació *commander* de manera recursiva.

La codificació del Commander per a AMO, genera $\sim \frac{n}{2}$ variables[34] i $\sim 3n$ clàusules.

6.3.4 Conclusions

Com ja hem vist, existeixen encodings molt eficients per a les restriccions de l'estil $|A| \leq 1$. Tot i ser només un cas especial dins les restriccions de cardinalitat, la diferència amb la resta es prou significativa com ser considerades a part. I en general intentarem treure'n profit sempre que sigui possible.

No obstant com ja passava amb les codificacions generals, per a cardinalitats petites la codificació directa continua essent més competitiva i tenint en compte el nombre de clàusules que es generen amb un i altre codificació, s'ha optat per la codificació directa per conjunts de variables menors de 9, ja que considerem que per a valors inferiors no compensa aplicar altres tipus de codificacions i fins i tot pot arribar a ser contraproduent.

Per exemple, calculem l'eficiència de les diverses codificacions per al cas concret de $n = 8$ variables d'entrada. A criteri nostre hem considerat que el cost d'afegir una nova variable equival al d'afegir fins a 5 clàusules. És a dir, s'ha utilitzat la següent expressió per avaluar l'eficiència de les codificacions:

$$\eta = 5V + C$$

Per tant,

$$\begin{aligned} \eta_{pairwise} &= 5 \cdot 0 \ + 28 \ = 28 \\ \eta_{ladder} &= 5 \cdot 7 + 20 = 55 \\ \eta_{commander} &= 5 \cdot 2 + 21 = 31 \qquad\qquad (m = 2) \end{aligned}$$

No és difícil veure que a partir de $n \geq 9$, la codificació *Commander* guanya en eficiència a la resta. Però per a $n < 9$ la codificació més eficient continua essent la codificació directa.

Òbviament, aquestes conclusions s'extreuen de la decisió arbitrària de que les variables equivalguin a 5 clàusules. Però com és pot extreure de les conclusions de [14], un valor de 5 sembla una assumpció força raonable. De fet a l'introduir variables auxiliars augmenta el camí crític, que és el camí que ha de seguir

[34] Depenent de l'elecció de les m a cada pas

el mecanisme de propagació unitària per tal de preservar l'arc-consistència de la restricció. Per tant sembla raonable pensar que les variables afegeixen més *overhead* al mecanisme de resolució que no pas les clàusules.

7 Requeriments

Un dels objectius principals d'aquest treball és el d'implementar un generador d'horaris capaç de confeccionar els horaris de tot un institut en un temps acotat. Però a més fer-ho emprant técniques i eines específiques. Per tot això a continuació s'enumeren els diferents requeriments de software que s'han necesitat per al generador.

7.1 Requeriments de programai

L'objectiu no només es el d'implementar un generador d'horaris sinó varis emprant diverses tècniques. Comparant-los en igualitat de condicions de manera que ens permeti avaluar l'eficiència de cada tècnica a l'hora de resoldre problemes d'aquest estil.

Una de les eines utilitzades ha estat el MiniZinc [16], que és una eina que permet descriure CSP usant un llenguatge de nivell mitjà. Permetent abstracció sobre el funcionament intern del *solver* que acabi resolent el problema definit.

No obstant, l'implementació amb MiniZinc es va descartar en els estadis inicials de la confecció del treball i l'implementació del generador consisteix en vàries implementacions de diverses tècniques realitzades ad-hoc. Totes elles han estat realitzades en el llenguatge de programació Python [17].

A continuació enumerem tot el programari que s'ha utilitzat en la confecció d'aquest treball:

- Per resoldre les instàncies MiniZinc s'han utilitzat els solver mzn-g12fd en la seva versió lazy. Que s'adjunta al paquet d'instal·lació del MiniZinc 2.0[35] en la versió per a sistemes basats en Linux.
- Per resoldre les instàncies basades en SMT s'ha utilitzat el SMT-solver Z3 Theorem Prover 4.4.0 [18] a través de Z3py, la seva API per a Python.
- Per resoldre les instàncies basades en MaxSAT s'han utilitzat els següents solvers: sat4j 2.3.4 [19] en la seva versió MaxSAT i el solver pwbo 2.2 [20] que és un portfolio que realitza diverses crides a un SAT-solver. Cadascuna en un thread diferent i amb condicions específiques.
- Per executar el sat4j, cal tenir instal·lat java en una versió igual o superior a Java 1.5.
- La codificació del generador s'ha realitzat en llenguatge Python versió 2.7.10
- Per la confecció de la memòria del treball s'ha utilitzat LaTeX, amb l'IDE TeXstudio 2.6.6 (hg 4099).

[35] `http://www.minizinc.org/2.0/install-linux.html`

- Per generar el diagrama de Gantt de la figura 1 s'ha utilitzat el GanttProject 2.7 Ostrava (build 1891).

7.2 Requeriments de maquinari

Per efectuar les proves de rendiment de les implementacions basades en SMT i SAT del generador d'horaris, hem utilitzat un ordinador amb un processador Intel(R) Core(TM) i7 CPU 860 @ 2.80GHz, 8MB de memòria cau i arquitectura de 64 bits. 8GB de memòria RAM. Sistema operatiu Linux Mint 17.1 64-bits basat en Ubuntu 14.04, Linux 3.13.0-37-generic.

8 Implementació

L'implementació del generador s'ha efectuat en Python, No obstant el generador consta de vàris blocs o parts ben diferenciats:

1. Parser del fitxers XML en format XHSTT.

   S'encarrega d'extreure la informació de la instància del fitxer XML d'entrada. Per tant cal que el fitxer d'entrada respecti l'estructura establerta pel format XHSTT.

2. Bloc de codificació de la instància a format SMT.

   Aquest bloc conté tota la lògica necessària per, donat un fixter en format XHSTT i amb l'ajut del *parser* codificar la instància a SMT usant els vectors de bits com a teoria de rerefons.

3. Bloc de codificació de la instància a format DIMACS (SAT).

   De manera similar al bloc SMT, aquest bloc també és capaç d'extreure el model d'un fitxer en format XHSTT i codificar-lo a una formula proposicional booleana en forma clausal normal (CNF). Després volca la fòrmula en un fitxer en format DIMACS [21], fitxer que posteriorment s'envia a un MaxSAT-solver per a que optimitzi la instància.

4. Bloc d'optimització d'instàncies codificades en SMT.

   A diferència de SAT, per a SMT s'ha implementat *ad-hoc* l'estratègia d'optimització. Tal i com es descriu a l'apartat Optimització SMT. Aquest bloc s'encarrega de gestionar les *hard-constraints* i les *soft-constraints* de manera que es minimitzi el cost del total de *soft-constraints* violades.

La idea principal és que l'usuari interactui amb el programa principal situat al fitxer *xhstt.py* i que aquest al seu torn interactui només amb les dues implementacions del generador, l'implementació MaxSAT i l'implementació SMT. És responsabilitat de cadascuna de les implementacions doncs, gestionar el *parseig* del fitxer d'entrada, codificar la instància i resoldre-la; deixant el resultat al subdirectori *output/*.
Per fer-ho ambdós models requereixen d'accés al *parser* així com d'un mecanisme per a obtenir una representació orientada a objectes de la lògica continguda al fitxer d'entrada. Com que aquesta part ès comuna en ambdós models, s'ha decidit englobar aquestes funcionalitats en una classe superior continguda en el fitxer *model.py* que té la responsabilitat d'obtenir la representació orientada a objectes del fitxer d'entrada.

Finalment per a resoldre la instància codificada, el model basat en MaxSAT invoca un MaxSAT-solver en un procés paral·lel i espera a que aquest retorni. Ja sigui perquè ha trobat l'òptim o bé perquè ha exhaurit el temps de còmput. El model basat en SMT en canvi, s'ajuda d'un bloc optimitzador que també s'ha implementat en motiu de la confecció del generador i que es troba en el fitxer

optimizer.py. Aquest bloc s'encarrega d'interactuar amb l'API del SMT-solver per a efectuar l'optimització.

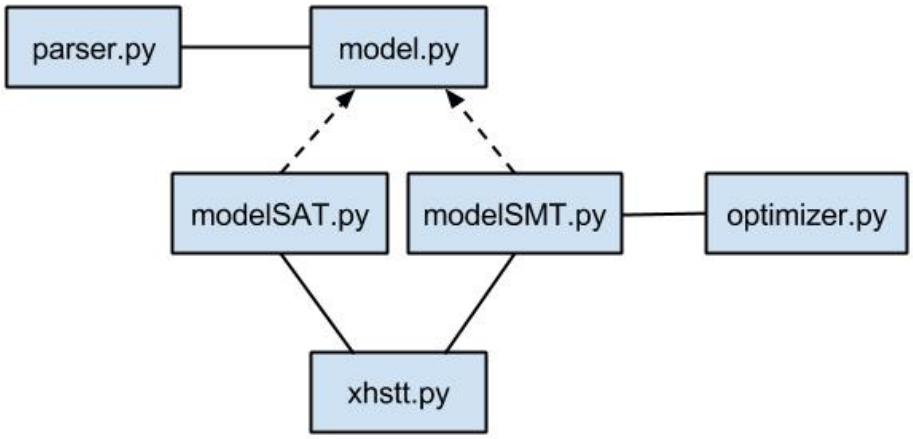

Figura 18: Esquema dels blocs que composen el generador d'horaris

8.1 Codificació SMT: BitVectors

En aquest apartat es descriu la implementació de la versió del generador d'horaris basat en vectors de bits, una teoria de les que ofereix SMT. Aquest paradigma ens força a codificar el problema en funció d'uns vectors de bits que ens faran de variables. Veurem que és un model molt semblant al respectiu model MaxSAT ja que la conversió entre bits i variables booleanes es natural. Tant és així que se sap que una de les tècniques que apliquen els soluciondors de teories de vectors de bits és l'anomenada *bitblasting*, que consisteix en codificar tots els vectors de bits (i les seves restriccions) a SAT, per finalment deixar que sigui un SAT solver qui solucioni el problema. No obstant, com ja hem vist SMT permet escriure formules en un llenguatge més enriquit que el de la lògica proposicional booleana. Fent que aquesta codificació pugui resultar més natural.

Cal tenir present que tots els operadors de la teoria requereixen que el nombre de bits de cada operand sigui el mateix. Per una banda tenim operadors relacionals: $=$, $\neq$, $<$, $\leq$, $>$, $\geq$. Per altra banda tenim operadors booleans: $\sim$ (negació), & (*and*), | (*or*), tots ells funcionen bit a bit. També tenim a disposició operadors aritmètics bàsics: $+$, $-$. Cal destacar que els operadors aritmètics consideren els vectors de bits com la representació binària en complement a 2 d'un enter (sempre que no s'especifiqui el contrari). Per últim també tenim disponibles operacions de desplaçament: $\ll n$ (desplaçament a l'esquerra de n bits) o $\gg n$ (desplaçament a la dreta de n bits).

8.1.1 Variables del Model

El model es centra en els *Events*, cada event es percep com una reunió on hi assisteixen diversos recursos. La feina del generador serà decidir els espais de temps en que les reunions tenen lloc. Per això el model consta dels següents conjunts de variables **per cada event**:

Sigui $BitVector(n)$ el constructor de vectors de bits de n bits.

- $Xt = BitVector(|Times|)$

  Vector de bits amb un bit per cada espai de temps de que consta la instància. Cadascun d'aquests bits està actiu[36] sii l'event que representa el vector de bits té lloc a l'espai de temps que representen.

- $Xs = BitVector(|Times|)$

  Vector de bits amb un bit per cada espai de temps de que consta la instància. Cadascun d'aquests bits està actiu sii una sessió de l'event en questió **comença** a l'espai de temps que representen. Diem que un event (o sessió) *comença* en un determinat espai de temps si l'event té lloc a aquella hora i, o bé és la primera hora del dia o no tenia lloc a l'hora anterior.

- $Xd_i = BitVector(|Times|)$, $1 \leq i \leq event.duration$

  Hi ha un vector de bits per cada possible duració de les lliçons de l'event en qüestió. És a dir, si l'event té una duració de 5 hores, es defineixen fins a 5 vectors de bits de $|Times|$ bits cadascun. Cada vector de bits representa els temps d'inici de les sessions d'una duració determinada d'aquell event.

8.1.2 Restriccions de Channeling de les variables del model

- Un event té lloc en tots els temps en que comença una sessió seva.

$$\bigwedge_{e \in Events} (Xt_e = Xt_e \mid Xs_e)$$

- Un event comença si té lloc a l'hora t però no a $t-1$.

$$\bigwedge_{e \in Events} (Xs_e = Xs_e \mid ((\sim Xt_e \gg_1) \;\&\; Xt_e))$$

- Xs correspon a l'*or* bit a bit de tots els vectors Xd.

$$\bigwedge_{e \in Events} (Xs_e = |_{i=1}^{duration} Xd_{e,i})$$

[36] Diem que l'i-éssim bit d'un vector de bits està **actiu** sii el vector de bits té un 1 a la posició i-éssima.

- Si comença una sessió de durada d, l'event té lloc en d hores consecutives, i no comença cap més sessió de l'event en les $d-1$ hores següents. També un cop acaba la sessió ja no es dóna l'event o bé comença una altre sessió (és el cas que l'actual sigui la última sessió del dia i que l'endemà a primera hora torni a haver-hi classe)

$$\bigwedge_{e \in Events} \bigwedge_{d=1}^{dur} (Xd_{e,d} = (\&_{i=0}^{d-1} Xt_e \ll_i) \,\&\, (\&_{i=1}^{d-1} \sim Xs_e \ll_i) \,\&\, (Xs_e \ll_{dur} \mid \sim Xt \ll_{dur}))$$

- Dues sessions no poden començar simultàniament

$$\bigwedge_{e \in Events} \bigwedge_{d_1=1}^{dur-1} \bigwedge_{d_2=d_1+1}^{dur} (Xd_{e,d_1} \,\&\, Xd_{e,d_2} = 0)$$

8.1.3 Restriccions XHSTT

- Assign Times Constraint

  Hem d'imposar que a tots els events se'ls hi assigni temps. Per tant imposem que cada vector de bits Xt ha de tenir tants bits activats com la duració de l'event.

$$\bigwedge_{e \in Events} exactly\_k\_bits(Xt_e,\ e.duration)$$

- Split Events Constraint

  Aquesta restricció posa límits en la manera com es fragmenten els events. És a dir:

  1. Límit sobre el nombre de sessions d'un event.
     N'hi ha prou amb posar limits sobre el vector Xs.

$$\bigwedge_{e \in Events} at\_least\_k\_bits(Xs_e,\ min) \wedge at\_most\_k\_bits(Xs_e,\ max)$$

  2. Límit sobre la durada de les sessions d'un event.
     Igualem a zero els vectors que representen duracions fora dels límits desitjats

$$\bigwedge_{e \in Events} \left(\bigwedge_{d=1}^{min} Xd_{e,d} = 0 \right) \wedge \left(\bigwedge_{d=max+1}^{e.duration} Xd_{e,d} = 0 \right)$$

- Distribute Split Events Constraint

  Aquesta restricció posa límits sobre el nombre de bits dels vectors Xd d'una determinada durada d.

$$\bigwedge_{e \in Events} at\_least\_k\_bits(Xd_{e,d},\ min) \wedge at\_most\_k\_bits(Xd_{e,d},\ max)$$

- Prefer Times Constraint

  Aquesta restricció ens indica que les sessions de durada d de certs events s'han de programar en un grup concret d'espais de temps. Definim $mask_{preferred}$ com el vector de bits de $|Times|$ bits, que té els bits activats sii el bit correspon a un espai de temps en els quals es poden programar les sessions d'un event.

$$\bigwedge_{e \in Events} (Xd_{e,d} = Xd_{e,d}\ \&\ mask_{preferred})$$

- Spread Events Constraint

  Aquesta restricció defineix grups d'hores que representen dies, i posa límits sobre el nombre de sessions de cada event que es poden programar cada dia o grup d'hores.

  Sigui Tg el conjunt de grups d'hores definits per la restricció i g_i un vector de bits amb els bits corresponents a l'i-èssim grup d'hores del conjunt activats i la resta a zero.

$$\bigwedge_{e \in Events} \bigwedge_{g \in Tg} at\_least\_k\_bits(Xs_e\ \&\ g,\ min) \wedge at\_most\_k\_bits(Xs_e\ \&\ g,\ max)$$

  També aprofitem aquesta restricció per afegir que totes les primeres hores de cada grup de temps (normalment dies) també han de ser inici de sessió. Sigui *heads* el vector de bits que té activats els espais de temps que corresponen a l'inici de cada grup d'hores definit a la restricció.

$$\bigwedge_{e \in Events} (Xs_e = Xs_e \mid (Xt_e\ \&\ heads))$$

- Avoid Clashes Constraint

  Aquesta restricció imposa que certs recursos no poden assistir a dos events de forma simultània. Per codificar-lo ens aprofitem del ft que els recursos ja estàn assignats a events des d'un inici, i definim E com el conjunt d'events als que ha d'assistir un recurs.

  Imposem que per cada recurs, per cada parella d'events dins E_r, aquestes no se solapin en el temps.

$$\bigwedge_{r \in Resources} \bigwedge_{e_1 \in E_r} \bigwedge_{e_2 \in E_r,\ e_1 < e_2} (Xt_{e_1} \ \& \ Xt_{e_2} = 0)$$

- Avoid Unavailable Times Constraint

  Aquesta restricció ens serveix per modelar el fet que certs recursos poden no estar disponibles per a certes hores.

  Sigui $mask_{unavailable}$ el vector de bits que només té activats els bits corresponents a les hores on el recurs no està disponible; i sigui E_r el conjunt d'events als que assisteix el recurs r:

$$\bigwedge_{r \in Resources} \bigwedge_{e \in E_r} (Xt_e = Xt_e \ \& \ \sim mask_{unavailable})$$

- Limit Idle Times Constraint

  Aquesta restricció té com a objectiu posar límits en el nombre de forats a l'horari de certs recursos. Definim forat com aquell espai de temps lliure entre dues hores ocupades d'un mateix dia. Per tal de posar límits sobre el nombre de forats en l'horari de cada recurs, ens cal afegir noves variables auxiliars al model. En concret afegirem un vector auxiliar que tingui els bits activats sii el temps al qual correspon el bit és un forat.

  Recuperem el conjunt Tg que apareixia en anteriors restriccions i que representa el conjunt de tots els grups de temps de què consta la setmana (generalment dies). Llavors, per cada Event s'introdueix un vector de $|Times|$ bits que anomenem $Idle_i$ i que tindrà actius només aquells bits corresponents a espais de temps en què l'event està ociós ($idle$). A continuació formalitzem la definició d'aquest nou vector de bits:

$$\bigwedge_{r \in Resources} \bigwedge_{g \in Tg} Idle_{r,g} = \sim occupied \ \& \ occupied\_B \ \& \ occupied\_A \ \& \ mask_g$$

  on

$$\begin{aligned} occupied &= \Big|_{e \in E_r} Xt_e \\ occupied\_Before &= \Big|_{e \in E_r} \Big|_{i=1}^{|g|} Xt_e \ll_i \\ occupied\_After &= \Big|_{e \in E_r} \Big|_{i=1}^{|g|} Xt_e \gg_i \end{aligned}$$

  Per cada recurs es generen $|Tg|$ vectors de bits que tindran els bits activats sii contenen forats. Per tant només cal imposar restriccions sobre el

nombre de bits activats de l'*or* bit a bit d'aquests vectors generats.

$$\bigwedge_{r \in Resources} at\_least\_k\_bits(\left(|_{g \in Tg} Idle_{r,g}\right), min)$$
$$\bigwedge_{r \in Resources} at\_most\_k\_bits(\left(|_{g \in Tg} Idle_{r,g}\right), max)$$

- Cluster Busy Times Constraint

  Aquesta restricció ens permet imposar límits sobre el nombre de dies (o grups d'hores) en què un recurs pot estar ocupat. Per codificar aquesta restricció amb vectors de bits ens ajudem d'un vector de bits auxiliar $Busy\_tg$ que tindrà el el bit corresponent a la primera hora de cada grup sii el recurs en qüestió està ocupat en aquell grup d'hores (dia). Per codificar el vector s'agafen grup a grup el resultat d'aplicar l'operació *or* bit a bit sobre els vectors Xt de cada event als quals assisteix el recurs. Es realitzen $|g|$ desplaçaments cap a l'esquerra, acumulant els resultats de cada operació amb un *or*. Finalment s'hi aplica la màscara del grup de temps amb només el primer bit de grup activat (per fer zeros la resta de bits) i ja tenim codificat el $Busy$ per un grup d'hores. Repetim la operació per la resta de grups i acumulem els resultats per acabar obtenint el $Busy\_tg$. Un cop codifificat el vector $Busy\_tg$ en tenim prou amb imposar restriccions sobre el nombre de bits actius d'aquest vector.

$$\bigwedge_{r \in Resources} at\_least\_k\_bits(Busy\_tg_r,\ min)$$
$$\bigwedge_{r \in Resources} at\_most\_k\_bits(Busy\_tg_r,\ max)$$

8.1.4 Codificació de soft-constraints

Per a codificar les restriccions "violables" s'utilitza una tècnica consistent a afegir variables de protecció a cada restricció. Així si tenim que una restricció a alt nivell es codifica en el conjunt de restriccions $C = \{c_1 \dots c_n\}$. Afegirem una variable de protecció p de tal manera que p impliqui C. És a dir transformarem C al conjunt de *soft-constraints* $SC = \{(\overline{p} \vee c_1) \dots (\overline{p} \vee c_n)\}$.D'aquesta manera, posant $I(p) = Fals$, eliminem totes les restriccions de SC ja que les satisfem totes.

Aquesta codificació ens serà molt útil ja que per a tasques d'optimització podrem saber fàcilment quins *soft-constraints* s'han violat mirant el conjunt de variables de protecció de la codificació. Totes les que estiguin negades indiquen que el conjunt de restriccions que protegien ha estat "violat".

8.1.5 Implementació d'optimitzador SMT

Per efectuar l'optimització dels horaris en la codificació SMT hem dissenyat un algorisme basat en cerca binària que minimitza la següent funció:

$$\sum_{i=1}^{|Soft-Constraints|} w_i \cdot \overline{p_i}$$

on w_i és el cost de violar la i-èssima *soft-constraint* i p_i és la representació pseudo-booleana de la variable de protecció associada a la restricció.

L'algorisme és el següent:

```
def optimize(HC, SC):
    s = solver()
    s.assert_constraints(HC)
    # first check if the hard constraints are satisfiable
    result = s.check()
    if result == sat:
        model = s.model()
        ub = sum([sc.weight for sc in SC])
        lb = 0
        while lb < ub:
            k = (ub+lb)/2
            s.assert(Sum([If(sc.p,0,1)*sc.weight for sc in SC]) <= k)
            result = s.check()
            if result = sat:
                model = s.model()
                ub = k
            else:
                s.retract_last()
                lb = k+1
        return model
    else:
        # Unsatisfiable
        return None
```

La funció $If(Bool, Int, Int)$, forma part de l'API del Z3 (SMT-solver) i si el primer paràmetre és cert s'avalua al segon paràmetre, altrament s'avalua al tercer.
La funció $Sum(list\_expr)$, també forma part de l'API del solver i s'avalua a la suma aritmètica de les expressions de la llista d'entrada.

8.2 Codificació MaxSAT

En aquest apartat es decriu la implementació de la versió MaxSAT del generador d'horaris. La codificació correspon a la categoria *Partial Weighted MaxSAT* ja que s'han de codificar *hard-constraints* i *soft-constraints* on cada *soft-constraint* té el seu cost associat. Per tant obtindrem un model amb clàusules "hard", i clàusules "soft" amb un cost associat. L'objectiu del generador serà satisfer totes les clàusules "hard" i el màxim nombre de clàusules "soft" de manera que es minimitzi el cost del total de clàusules violades.

Aquesta codificació està pensada per a resoldre un subconjunt de les instàncies presents en el repositori públic de XHSTT. En particular hem escollit especialitzar-nos en resoldre les instàncies on tots els recursos estan assignats d'entrada. Per una banda permet simplificar l'espai de cerca, al reduir la combinatòria del problema i per l'altra permet una codificació més simple i directe de les restriccions de que es compon. Es la situació més comuna en les instàncies reals, ja que les pròpies assignatures ja van lligades a un grup concret (1^{er} A, 2^{on} B, etc), cada grup acostuma a tenir assignada una aula específica, per evitar els canvis d'aula entre espais de temps. I els professors s'acostumen a repartir les assignatures que els tocarà impartir abans de la confecció de l'horari. Per tot això i donat l'alt nombre d'instàncies d'aquestes característiques hem optat per realitzar aquesta simplificació.

8.2.1 Model

El model es centra en els *Events*, cada event es percep com una reunió on hi assisteixen diversos recursos. La feina del generador serà decidir els espais de temps en que les reunions tenen lloc. Per això el model consta dels següents conjunts de variables **per cada event**:

- $Xt_0 \dots Xt_{|time\_slots|-1}$

  Hi ha una variable per cada espai de temps de que consta la instància. Cadascuna d'aquestes variables indica si l'event està programat per a l'espai de temps que representen. Per exemple si $I(Xt_7) = Cert$ i $I(Xt_8) = Fals$, indica que l'event en questió té lloc al vuitè espai de temps i no al novè.

- $Xs_0 \dots Xs_{|time\_slots|-1}$

  Hi ha una variable per cada espai de temps de que consta la instància. Cadascuna d'aquestes variables indica si l'event **comença** en l'espai de temps que representen. Diem que un event *comença* en un determinat espai de temps si l'event té lloc a aquella hora i, o bé és la primera hora del dia o no tenia lloc a l'hora anterior.

- $Xd_{1,0} \dots Xd_{duration,\ |time\_slots|-1}$

  Hi ha una variable per cada espai de temps i per cada possible duració d'una lliço d'un determinat event. Això és, si l'event té duració de 5 hores,

es defineixen fins a 5 conjunts de $|time\_slots|$ variables cadascun. Cada variable indica si **comença** una lliçó de la durada i en l'espai temps que representa la variable.

8.2.2 Clàusules de Channeling de les variables del model

A continuació s'enumeren les clàusules necessàries per dotar a les variables del model de la semàntica que hem descrit en l'apartat anterior.

- Si un event comença a una hora determinada, llavors té una duració.

$$\bigwedge_{i=0}^{|time\_slots|-1} exactly\_one\left(\overline{Xs_i} \vee \left(\bigvee_{j=1}^{duration} Xd_{j,i}\right)\right)$$

- Un event comença si té lloc a l'hora t però no a $t-1$.

$$\bigwedge_{i=1}^{|time\_slots|-1} (\overline{Xt_i} \vee Xt_{i-1} \vee Xs_i)$$

- Si un event comença amb duracio d, llavors té lloc en d hores consecutives.

$$\bigwedge_{d=1}^{duracio} \bigwedge_{i=0}^{|time\_slots|-d} \bigwedge_{j=i}^{i+d-1} (\overline{Xd_i} \vee Xt_j)$$

Queden pendents afegir clàusules que gestionin el canvi de dies. És a dir clàusules que imposin que la primera hora de cada dia ha de ser una hora d'inici. Però aquestes clàusules s'afegeixen quan el model introdueix la noció de dia mitjançant restriccions de l'estil Spread Events Constraint.

8.2.3 Restriccions de cardinalitat

Per a codificar les restriccions de cardinalitat[37] hem optat per una Mixed Cardinality Network amb $\lambda = 5$. En cas que la restricció de cardinalitat sigui "soft", utilitzen la codificació per ordenar l'entrada, i després afegim una clàusula amb pes forçant la cardinalitat desitjada. Per exemple, donat el conjunt d'entrada $X = \{x_1, x_2, x_3, x_4, x_5\}$ i s'ha de codificar la restricció $1 \leq |X| \leq 2$. Primer obtenim la representació del conjunt X ordenat

$$S = \{s_1, s_2, s_3, s_4, s_5\} = MixedCardinalityNetwork(X)$$

Després afegim les següents clàusules "soft":

$$soft\_clauses = (\langle s_1, weight\rangle) \wedge (\langle \overline{s_3}, weight\rangle) \wedge (\langle \overline{s_4}, weight\rangle)(\langle \overline{s_5}, weight\rangle)$$

Per tant si una solució té $|X| = 0$, violarà la primera clàusula i per tant tindrà cost $weight$. Si per contra té $|X| = 4$ estarà violant la segona i tercera clàusula i per tant tindrà cost $2 * weight$.

[37] *at_most_k, at_least_k* i *exactly_k.*

8.2.4 Restriccions XHSTT

- Assign Times Constraint

  Hem d'imposar que a tots els events se'ls hi assigni temps. Per tant, per cada event imposem que del seu conjunt de variables Xt, s'hauran d'avaluar a $Cert$ tantes variables com la durada de l'event en qüestió. És a dir:

$$\bigwedge_{e \in Events} exactly\_k(\{Xt_{e,0} \dots Xt_{e,|Times|-1}\},\ e.duration)$$

- Split Events Constraint

  Aquesta restricció posa límits en la manera com es fragmenten els events. És a dir sobre el nombre de sessions en què es fragmenta (1) i la durada d'aquestes sessions (2).

  1. $$\bigwedge_{e \in Events} at\_most\_k(\{Xs_{e,0} \dots Xs_{e,|Times|-1}\}, max) \quad (max < |Times|)$$
     $$\bigwedge_{e \in Events} at\_least\_k(\{Xs_{e,0} \dots Xs_{e,|Times|-1}\}, min) \quad (min > 0)$$

     On max correspon al tag $MaximumAmount$, que indica el màxim nombre de sessions en què es pot fragmentar un determinat event i min correspon al tag $MinimumAmount$ i indica el mínim nombre de sessions en què es pot fragmentar un determinat event.
  2. Per aquesta restricció, neguem totes les variables Xd de cada event que corresponguin a duracions que no estiguin compreses entre el rang definit pels tags $MinimumDuration$ i $MaximumDuration$

- Distribute Split Events Constraint

  Aquesta restricció posa límits a la cardinalitat de variables Xd d'una determinada durada. Per tant la seva codificació es redueix a la combinació d'un at_most_k i un at_least_k, on les k's venen definides per els tags $Maximum$ i $Minimum$ respectivament. El paràmetre d ve donat pel tag $Duration$ i indica la durada de les sessions a les quals fa referencia aquesta restricció.

$$\bigwedge_{e \in Events} at\_most\_k(\{Xd_{e,d,0} \dots Xd_{e,d,|Times|-1}\}, max) \quad (max < \frac{e.duration}{d})$$
$$\bigwedge_{e \in Events} at\_least\_k(\{Xd_{e,d,0} \dots Xd_{e,d,|Times|-1}\}, min) \quad (min > 0)$$

- Prefer Times Constraint

  Ens indica que certes sessions de certs events s'han de programar en un grup concret d'espais de temps. Més específicament, determina en quins espais de temps no es poden programar certes sessions. Per exemple, ens serveix per indicar que no es pot programar una sessió de dues hores a última hora del dia.

  Definim el conjunt Ta com el conjunt de slots de temps els quals la restricció ens diu que són els preferits. És a dir, els que es trobarien sota el tag $TimeGroups$. Neguem totes les variables Xd de duració d que representin espais de temps no continguts dins Ta.

$$\bigwedge_{e \in Events} \bigwedge_{t \in Times} (\overline{Xd_{e,d,t}}) \qquad (t \notin Ta)$$

- Spread Events Constraint

  Aquesta restricció defineix grups de hores que representen dies, i posa límits sobre el nombre de sessions de cada event que es poden programar cada dia o grup d'hores.

  Sigui Tg el conjunt de grups d'hores definits per la restricció i g_i l'i-èssim grup d'hores del conjunt Tg, llavors la restricció es codifica de la següent manera:

$$\bigwedge_{e \in Events} \bigwedge_{g \in Tg} at\_most\_k(\{Xs_{e,t} \mid t \leftarrow g\}, max)$$
$$\bigwedge_{e \in Events} \bigwedge_{g \in Tg} at\_least\_k(\{Xs_{e,t} \mid t \leftarrow g\}, min)$$

- Avoid Clashes Constraint

  Aquesta restricció demana que certs recursos no poden tenir assignats més d'un event al mateix temps. Per a codificar-lo aprofitem el fet que els recursos estàn assignats d'un inici, i definim E com el conjunt d'events als que ha d'assistir un recurs. Després imposem que per cada espai de temps de la setmana, com a molt un d'els events de E pot estar programat. Això ho fem mitjançant l'ús d'un *at_most_one* sobre el conjunt de variables Xt dels events als quals assisteix el recurs en qüestió.

$$\bigwedge_{r \in Resources} \bigwedge_{t \in |Times|} at\_most\_one(\{Xt_{e,t} \mid e \leftarrow E_r\})$$

- Avoid Unavailable Times Constraint

  Indica que hi ha certes hores durant les quals certs recursos no estan disponibles. Sigui T el conjunt d'hores durant les quals els recursos en qüestió no estan disponibles i E_i el conjunt d'events als que ha d'assistir l'i-èssim recurs. Llavors:

$$\bigwedge_{r \in Resources} \bigwedge_{t \in T} \bigwedge_{e \in Er} (\overline{Xt_{e,t}})$$

- Limit Idle Times Constraint

  Aquesta restricció té com a objectiu posar límits en el nombre de forats a l'horari de certs recursos. Definim forat com aquell espai de temps lliure entre dues hores ocupades d'un mateix dia. Per tal de posar límits sobre el nombre de forats en l'horari de cada recurs, ens cal afegir noves variables auxiliars al model. Variables que ens permetin capturar la semàntica que necessitem, que en aquest cas es tracta d'introduir per cada espai de temps candidat a ser forat una variable que sigui certa si l'espai de temps en concret és forat o no.

  Recuperem el conjunt Tg que apareixia en anteriors restriccions i que representa el conjunt de tots els grups de temps de què consta la setmana (generalment dies). Llavors, per cada Event i hora s'introdueix una variable $Idle_i$ que indica que l'event té un forat a la i-éssima hora. Perquè un espai de temps sigui considerat com a forat cal que algun dels espais de temps anteriors del seu grup estigui ocupat i algun altre dels posteriors també estigui ocupat. Per això definim dos conjunts més A i B on B significa *Before* i conté els slots de temps del mateix grup d'hores (o dia) que van abans cronològicament de l'i-éssim espai de temps. I de manera anàloga per a A però amb les hores que ocorren després.

  Llavors **per cada recurs** r:

$$\bigwedge_{g \in Tg} \bigwedge_{t \in g} \bigwedge_{e \in E_r} (\overline{Idle_t} \vee \overline{Xt_{e,t}}) \qquad (B_t \neq \emptyset, A_t \neq \emptyset)$$

$$\bigwedge_{g \in Tg} \bigwedge_{t \in g} \left(\overline{Idle_t} \vee \bigvee_{b \in B_t} \bigvee_{e \in E_r} Xt_{e,b} \right) \qquad (B_t \neq \emptyset)$$

$$\bigwedge_{g \in Tg} \bigwedge_{t \in g} \left(\overline{Idle_t} \vee \bigvee_{a \in A_t} \bigvee_{e \in E_r} Xt_{e,a} \right) \qquad (A_t \neq \emptyset)$$

$$\bigwedge_{g \in Tg} \bigwedge_{t \in g} \bigwedge_{b \in B_t} \bigwedge_{a \in A_t} \bigwedge_{e_1 \in E_r} \bigwedge_{e_2 \in E_r} \bigwedge_{e_3 \in E_r} (Xt_{e1,t} \vee \overline{Xt_{e_2,b}} \vee \overline{Xt_{e3,a}} \vee Idle) \quad (B_t \neq \emptyset, A_t \neq \emptyset)$$

Un cop definides i lligades les variables que representen els forats només queda afegir restriccions sobre la cardinalitat d'aquestes restriccions. Sigui $Idles$ el conjunt de totes les variables auxiliars que indiquen si es produeixen forats o no (fixem-nos que n'hi hauran menys que $|Times|$ ja que l'inici i final de cada dia no poden ser forats)

$$at\_most\_k(\{Idle_t \mid t \leftarrow Times\}, max)$$
$$at\_least\_k(\{Idle_t \mid t \leftarrow Times\}, min)$$

- Cluster Busy Times Constraint

  Aquesta restricció ens permet imposar límits sobre el nombre de dies (o grups d'hores) en què un recurs pot estar ocupat. Com en la restricció anterior ens ajudarem de variables auxiliars $Busy$ que ens indicaran si un determinat recurs està ocupat un determinat dia (o grup d'hores). Sigui Tg el conjunt de grups d'hores als quals fa referència la restricció i sigui E_r el conjunt d'events als que assisteix el recurs r, llavors **per cada recurs** r:

$$\bigwedge_{g \in Tg} \left(\overline{Busy_g} \vee \left(\bigvee_{e \in E_r} \bigvee_{t \in g} Xt_{e,t} \right) \right)$$
$$\bigwedge_{g \in Tg} \bigwedge_{t \in g} \bigwedge_{e \in E_r} (\overline{Xt_{e,t}} \vee Busy_g)$$

$$at\_most\_k(\{Busy_g \mid g \leftarrow Tg\}, max)$$
$$at\_least\_k(\{Busy_g \mid g \leftarrow Tg\}, min)$$

8.2.5 Codificació de soft-constraints

La codificació de *soft-constraints* funciona exactament igual que amb la codificació SMT.

Per a codificar les restriccions "violables" s'utilitza una tècnica consistent a afegir variables de protecció a cada restricció. Així si tenim que una restricció a alt nivell es codifica en el conjunt de clàusules $C = \{c_1 \dots c_n\}$. Afegirem una variable de protecció p de tal manera que p impliqui C. És a dir transformarem C al conjunt de *soft-constraints* $SC = \{(\overline{p} \vee c_1) \dots (\overline{p} \vee c_n)\}$.D'aquesta manera, posant $I(p) = Fals$, eliminem totes les clàusules de SC ja que les satisfem totes.

9 Resultats

En aquesta secció realitzarem una comparativa dels resultats dels diversos generadors donat un conjunt d'instàncies d'instituts reals.

El format XHSTT disposa d'un avaluador d'horaris online anomenat HSeval [22]. Aquest avaluador online, no només serveix per avaluar la bondat d'un horari (la seva optimalitat) sino que a més permet mostrar l'horari de forma gràfica. A continuació il·lustrarem els resultats de la resolució per a la instància *BrazilInstance1.xml*. La solució ha estat generada per el generador basat en vectors de bits de SMT. Però com que el format de l'horari és totalment agnòstic al generador, el procés és idèntic per a les solucions obtingudes a través del generador basat en MaxSAT.

A la pàgina principal de l'HSeval, carreguem el fitxer XML que ha creat el generador, seleccionem la opció **HTML Report** i fem click a **Submit**. Al cap d'uns segons, serem redirigits a una pàgina web amb l'anàlisi de l'optimalitat de l'horari que hem carregat. Tal i com es mostra a la figura19.

Si en comptes de l'anàlisi d'optimalitat es desitja obtenir una representació gràfica de l'horari generat, ho podem fer mitjançant l'HSeval també. Anem a l'inici, carreguem el fitxer XML amb la solució, seleccionem la opció **HTML Timetables** i clickem **Submit**. En aquesta ocasió la plataforma HSeval generarà una representació visual dels horaris de cada recurs. Per exemple, podrem observar els horaris que tenen els diferents professors o l'horari de tot una classe. Que és exactament el que voliem en primera instància. La instància *BrazilInstance1* consta de 8 professors i 3 classes. Per no carregar el document amb captures redundants, no hem adjuntat captures dels horaris de tots els professors ni de totes les classes. A la figura 20 si mostren alguns horaris de professors, mentre que a la figura 21 s'hi mostren els horaris de dues classes.

També, seleccionant la versió llarga de l'opció **HTML Timetables**, podem generar els horaris des del punt de vista de les assignatures. Tal i com es mostra a la figura 22

Solution obtained with a BitVector modelling powered by z3 theorem-prover

Contributed by Cristofor Nogueira, 17052015.

Solution of instance BrazilInstance1_XHSTT-v2014

Distribute Split Events Constraint	Constraint name	Calculation	Inf.	Obj.
T8-S1	At least 1 double lesson(s)	1 * Linear(1 too few in T8-S1)		1
T8-S2	At least 1 double lesson(s)	1 * Linear(1 too few in T8-S2)		1
T8-S3	At least 1 double lesson(s)	1 * Linear(1 too few in T8-S3)		1
T6-S3	At least 2 double lesson(s)	1 * Linear(1 too few in T6-S3)		1
Total (4 points)				4

Limit Idle Times Constraint	Constraint name	Calculation	Inf.	Obj.
T1	No IDLE times for teachers	3 * Linear(1 in Th)		3
T2	No IDLE times for teachers	3 * Linear(3 in Tu)		9
T3	No IDLE times for teachers	3 * Linear(1 in Mo)		3
T5	No IDLE times for teachers	3 * Linear(2 in Th)		6
T7	No IDLE times for teachers	3 * Linear(3 in Mo, We)		9
T8	No IDLE times for teachers	3 * Linear(2 in We)		6
Total (6 points)				36

Cluster Busy Times Constraint	Constraint name	Calculation	Inf.	Obj.
T2	Not more than 2 days with lessons	9 * Linear(1 too many of Mo, Tu, We)		9
T5	Not more than 2 days with lessons	9 * Linear(1 too many of Tu, We, Th)		9
T7	Not more than 2 days with lessons	9 * Linear(1 too many of Mo, We, Th)		9
Total (3 points)				27

Summary	Inf.	Obj.
Distribute Split Events Constraint (4 points)		4
Limit Idle Times Constraint (6 points)		36
Cluster Busy Times Constraint (3 points)		27
Grand total (13 points)		67

Figura 19: Anàlisi d'optimalitat d'una solució efectuat per l'HSeval

T8

Mo	Tu	We	Th	Fr
T8-S1		T8-S1		
T8-S3				
T8-S2		T8-S2		
		T8-S3		

Constraint violations	Constraint name	Calculation	Inf.	Obj.
Distribute Split Events Constraint	At least 1 double lesson(s)	1 * Linear(1 too few in T8-S1)		1
Distribute Split Events Constraint	At least 1 double lesson(s)	1 * Linear(1 too few in T8-S3)		1
Distribute Split Events Constraint	At least 1 double lesson(s)	1 * Linear(1 too few in T8-S2)		1
Limit Idle Times Constraint	No IDLE times for teachers	3 * Linear(2 in We)		6

Figura 20: Representació gràfica de l'horari d'un professor de BrazilInstance1

S3

Mo	Tu	We	Th	Fr
T7-S3	T3-S3	T7-S3	T1-S3	T4-S3
T8-S3	T3-S3	T7-S3	T4-S3	T4-S3
T3-S3	T5-S3	T5-S3	T7-S3	T1-S3
T6-S3	T5-S3	T5-S3	T7-S3	T1-S3
T6-S3	T6-S3	T8-S3	T5-S3	T6-S3

Constraint violations	Constraint name	Calculation	Inf.	Obj.
Distribute Split Events Constraint	At least 1 double lesson(s)	1 * Linear(1 too few in T8-S3)		1
Distribute Split Events Constraint	At least 2 double lesson(s)	1 * Linear(1 too few in T6-S3)		1

Figura 21: Representació gràfica de l'horari d'un grup de BrazilInstance1

T8-S3

Mo	Tu	We	Th	Fr
T8-S3 Class: S3 Teacher: T8				
		T8-S3 Class: S3 Teacher: T8		

Constraint violations	Constraint name	Calculation	Inf.	Obj.
Distribute Split Events Constraint	At least 1 double lesson(s)	1 * Linear(1 too few in T8-S3)		1

Figura 22: Representació gràfica de l'horari d'una assignatura de BrazilInstance1

9.1 Comparativa de Rendiment

Sempre és interessant comparar el rendiment dels generadors implementats en un intent d'esbrinar quina de les tecnologies emprades és més adient a l'hora de resoldre problemes d'aquest tipus. A tal fi s'ha dissenyat el següent experiment. Hem intentat resoldre fins a sis instàncies del repositori públic de l'HSTT, amb un temps de còmput màxim de 1800 segons per instància. És a dir a l'equivalent a mitja hora. L'objectiu és comprovar com d'eficients són els generadors a l'hora de resoldre i optimitzar instàncies en un període de temps molt acotat.

Generador	BrazilInstance1	BrazilInstance2	BrazilInstance3
SMT (BitVectors)	(0, 53)	(0, 88)	(0, 273)
MaxSAT (MCN)	(0, 86)	(0, 109)	(0, 245)
MaxSAT (MTO)	(0, 87)	(0, 103)	(0, 259)

Taula 2: Rendiment dels generadors basats en MaxSat, en les versions usant encodings de cardinalitat basats en *Mixed Cardinality Networks* i en la versió basada en *Modulo Totalizers* i SMT, basat en vectors de bits. Per cada solver i instància s'indiquen dos valors. El primer és el cost d'inviabilitat. És a dir, quin nombre de restriccions "inviolables" s'han violat. El segon representa el valor d'optimalitat de la solució obtinguda i es calcula a partir del nombre de restriccions "violables" que s'han violat.

Generador	BrazilInstance4	BrazilInstance5	BrazilInstance6
SMT (BitVectors)	(0, 141)	(0, 388)	(0, 578)
MaxSAT (MCN)	(0, 147)	(0, 276)	(0, 422)
MaxSAT (MTO)	(0, 158)	(0, 309)	(0, 453)

Taula 3: Rendiment dels generadors basats en MaxSat, en les versions usant encodings de cardinalitat basats en *Mixed Cardinality Networks* i en la versió basada en *Modulo Totalizers* i SMT, basat en vectors de bits. Per cada solver i instància s'indiquen dos valors. El primer és el cost d'inviabilitat. És a dir, quin nombre de restriccions "inviolables" s'han violat. El segon representa el valor d'optimalitat de la solució obtinguda i es calcula a partir del nombre de restriccions "violables" que s'han violat.

En aquesta comparativa s'han tingut en compte els generadors basats en SMT i les versions de MaxSAT que, han estat els generadors que han superat totes les etapes d'implementació al llarg de la confecció del treball. També és interessant realitzar una comparativa entre el tamany de les codificacions que generen els dos generadors en les dues versions de MaxSat.

Tots els tests s'han realitzat en una màquina amb un processador Intel(R) Core(TM) i7 CPU 860 @ 2.80GHz de 8 nuclis, 8MB de memòria cau i arqui-

	Mixed Cardinality Network	Modulo Totalizer
BrazilInstance1	(6332, 39158)	(8674, 48823)
BrazilInstance2	(14327, 103754)	(21411, 132255)
BrazilInstance3	(17288, 139867)	(23742, 166495)
BrazilInstance4	(28284, 274881)	(42510, 332474)
BrazilInstance5	(29906, 267886)	(43690, 322933)
BrazilInstance6	(32576, 266667)	(48372, 330091)

Taula 4: Nombre de variables i clàusules generats pels diferents generadors basats en MaxSAT. En parèntesis, el nombre de variables seguit del nombre de clàusules.

tectura de 64 bits. 8GB de memòria RAM. Sistema operatiu Linux Mint 17.1 64-bits basat en Ubuntu 14.04, nucli linux 3.13.0-37-generic. Els models generats pel generador SMT les ha resolt el SMT-solver *Z3 Theorem Prover* mentre que les instàncies per resoldre els problemes MaxSAT s'han resolt utilitzant el solver *sat4j* en la seva versió per a MaxSAT.

De les taules 9.1 i 9.1 se'n poden extreure vàries conclusions:

1. El generador MaxSAT (en qualsevol de les seves dues versions) sembla que en general té més bon rendiment que el generador SMT.

   Cal posar en context que el generador SMT optimitza mitjançant una estratègia de cerca binària. Apropant-se molt ràpidament als punts més "durs" del problema. Aquest fet en general és desitjable, sobretot si interessa provar l'optimalitat de la solució. Però en el context d'aquestes proves, i tenint en compte la limitació de temps evident, aquesta estratègia ha resultat ser un pèl contraproduent ja que mentre el solver MaxSAT es dedica a intentar millorar la seva solució pas a pas, la versió SMT ho fa salt a salt. Dit d'una altra manera, en la majoria d'execucions el SMT-solver trobava la millor solució en menys del primer quart de l'execució. Mentre que la progressió del solver MaxSAT era més lineal; en alguns casos arribant a millorar la solució en els últims segons d'execució.

   En favor de la codificació basada en vector de bits, afegir que l'algorisme d'optimització del generador ha estat implementat per nosaltres, mentre que l'optimització del generador MaxSAT està implementada pel propi solver incloent les tècniques i heurístiques més avançades fins la data.

2. El generador MaxSAT sembla millor a mesura que augmenta el tamany de la instància.

   Un aspecte que no es veu a les taules de resultats, és el temps que tarden

els diferents solvers a trobar la seva primera solució. En general el generador basat en MaxSAT es capaç de trobar una solució més ràpidament que el basat en SMT. En el sentit que l'oracle MaxSAT en general té un temps de resposta menor que l'oracle basat en SMT. I aquesta diferència es més notable a mesura que augmenta la duresa del problema.

Tot i que aquestes afirmacions s'ha d'agafar una mica en pinces ja que els dos generadors tenen estratègies d'optimització diferents.

3. En MaxSAT la codificació de restriccions de cardinalitat basades en *Mixed Cardinality Networks* és més eficient que la codificació basada en *Modulo Totalizer*.

   Estudis recents [23] donaven a entendre que la codificació basada en Modulo Totalizer era més eficient que la basada en Cardinality Networks. Més eficient en terme de clàusules i variables i temps de còmput, ja que la codificació basada en Totalizers té un camí crític menor a l'hora de preservar l'arc-consistència de la restricció de cardinalitat. No obstant [14] proposen un refinament de la codificació dels Cardinality Networks. De tal manera que la relació d'eficiència entre les dues codificacions no era evident. Gràcies a aquest treball podem dir que, a la vista dels resultats obtinguts, la codificació basada en Mixed Cardinality Networks no només millora en el tamany de la codificació sinó que a més millora en temps de còmput, ja que aconsegueixen superar en quasi tots els jocs de proves a la codificació basada en MTO.

9.2 Reproducció de Resultats

A continuació es descriuen les comandes usades per invocar el generador d'horaris.

- Executar el generador basat en vectors de bits (SMT)

  ```
  $ python xhstt.py -m bv <instance.xml_file>
  ```

- Executar el generador basat en MaxSAT (Mixed Cardinality Networks encoding)

  ```
  $ python xhstt.py -m sat -e mcm <instance.xml_file>
  ```

- Executar el generador basat en MaxSAT (Modulo Totalizer encoding)

  ```
  $ python xhstt.py -m sat -e mto <instance.xml_file>
  ```

Per indicar el timeout ho farem amb l'opcio *-t*. Exemple:

```
$ python xhstt.py -m bv -t 258745201 <instance.xml_file>
```

Si no s'indica el model s'usa el *SAT* per defecte. L'encoding per defecte és el Mixed Cardinality Network i el timeout per defecte és 1800000, és a dir 1800 segons.

10 Conclusions

Al llarg d'aquest treball s'han vist i aplicat diverses tècniques per resoldre el problema de generar horaris per a un institut. De les quals les més estudiades han estat les tècniques basades en codificacions SAT i SMT, que son les que hem acabat utilitzant per a la versió final del generador d'horaris.

Respecte al generador d'horaris, el problema HSTT ha permès posar a prova tots els coneixements que s'anaven adquirint al llarg del treball i, si bé el producte final és millorable en molts sentits, compleix amb els requisits inicials ja que s'ha aconseguit implementar un generador capaç de confeccionar els horaris per a tot un institut en un temps molt acotat; que era l'objectiu que ens havíem marcat a l'inici. La instància BrazilInstance6 de la secció de Resultats correspon als horaris de l'any 2000 de l'institut públic *Dom Silvério* de la localitat de Mariana, Minas Gerais, Brasil. Traduïda a format XHSTT per Haroldo Santos.

Malgrat que el generador d'horaris confeccionat durant aquest treball és capaç de resoldre instàncies reals amb relativament poc temps (menys de mitja hora), no té tan bon rendiment a l'hora d'optimitzar l'horari. Que era d'esperar donat que optimitzar sempre és més dur que no pas decidir. Com ja havíem apuntat a la descripció del problema, era molt interessant obtenir un generador capaç de confeccionar la millor solució en poc temps. Aspecte que clarament no s'ha assolit amb el nostre generador, que genera solucions que disten molt de les òptimes.

No obstant, el punt fort d'aquest treball ha estat sense dubte l'anàlisi realitzat sobre les tecnologies i el seu estat de l'art. Les motivacions inicials i les ganes d'aprendre més sobre el paradigma de l'optimització ens ha permès aprofundir en l'essència de vàries de les tècniques que s'han tractat. Reflectint-ho d'aquesta manera en una síntesi simplista i amb paraules pròpies d'aquests coneixements adquirits gràcies als coneixements publicats per desenes d'investigadors durant els últims anys.

En el camp SAT també s'han realitzat proves de rendiment sobre nous encodings que s'han implementat durant la confecció d'aquest treball seguint les indicacions d'investigadors de renom. I el més important és que aquest treball ens ha dotat del coneixement necessaris per afegir millores "ad-hoc" al problema com per exemple la millora sobre les Cardinality Networks a l'hora de codificar restriccions de cardinalitat amb pes. O bé importar la idea de les Mixed Cardinality Networks sobre els Modulo Totalizers.

Aquest treball m'ha servit per consolidar els meus estudis en enginyeria informàtica. Entendre millor el món de la recerca, ajudant-me a comprendre millor la naturalesa dels problemes, especialment els NP-Complets.

11 Treball Futur

En primer lloc, i continuant amb el model de prototipatge que ha caracteritzat la metodologia emprada en aquest treball, sería bo continuar el refinament del generador d'horaris mitjançant l'implementació de nous prototips basats altres tecnologies i/o refinant els ja existents.

També és d'interès ampliar el nombre d'instàncies suportades per el generador afegint-hi codificacions per a nous tipus de restriccions. Que en la confecció d'aquest treball s'han obviat.

Des del punt de vista de l'aplicació del coneixement adquirit durant aquest projecte, es proposa el següent:

- Traduïr l'encoding de la restricció de cardinalitat de vectors de bits de SMT a SAT. D'aquesta manera aconseguiriem crear un encoding que generaria $O(n\ k)$ clàusules i $O(n\ k)$ variables. Que en alguns casos pot arribar a ser millor que el *Mixed Cardinality Network* que recordem, genera $O(n\ log_2^2\ k)$ clàusules i $O(n\ log_2^2\ k)$ variables.
- Explorar més el paradigma que presenta SMT, ja que en aquest treball només hem explorat la teoria de vectors de bits i la de pseudo-booleans i ens hem quedat amb la sensació que només n'eren la punta de l'iceberg.

12 Referències

1. https://en.wikipedia.org/wiki/Software_prototypin.
2. International High School Timetabling competition. http://www.utwente.nl/ctit/hstt/
3. SMT-LIB. The Satisfiability Modulo Theories Library. http://smtlib.cs.uiowa.edu/
4. J. Giráldez Crú. Structure and Real-world SAT solving. Xerrada a la UdG. 26 de Març de 2015.
5. Bin packing problem. https://en.wikipedia.org/wiki/Bin_packing_problem
6. Tim B Cooper and Jeffrey H Kingston. The Complexity of Timetable Construction Problems. Techical Report Number 495. February 1995.
7. Gerhard Post, Jeffrey H Kingston, Samad Ahmadi, Sophia Daskalaki, Christos Gogos, Jari Kyngas, Cimmo Nurmi, Nysret Musliu, Nelishia Pillay, Haroldo Santos, et al. Xhstt: an xml archive for high school timetabling problems in different countries.Annals of Operations Research, 2011.
8. An XML format for benchmarks in High School Timetabling. Annals of Operations Research 194, pp. 385–397, 2012, by Gerhard Post, Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngas, Cimmo Nurmi, and David Ranson.
9. Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In Proceedings 12th National Conference on Artificial Intelligence (AAAI 94), pages 362–367, 1994.
10. Cook, Stephen A. (1971). "The Complexity of Theorem-Proving Procedures"(PDF). Proceedings of the 3rd Annual ACM Symposium on Theory of Computing: 151-158.
11. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT solver. In Design Automation Conferencepages 530–535, June 2001.
12. J.P. Marques-Silva, I. Lynce, S. Malik. Conflict-Driven Clause Learning SAT Solvers. Handbook of Satisfiability 2009: 131-153.
13. Olivier Bailleux and Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardi- nality Constraints. In F. Rossi, editor, Principles and Practice of Constraint Pro- gramming, 9th International Conference, CP '03, volume 2833 of Lecture Notes in Computer Science, pages 108–122. Springer, 2003.

14. Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell. A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints.

15. Klieber, W., Kwon, G.: Efficient CNF encoding for selecting 1 from n objects. In: the Fourth Workshop on Constraint in Formal Verification(CFV). (2007).

16. N. Nethercote, P.J. Stuckey, R.Becket, S. Brand, G. J. Duck, and G. Tack. MiniZinc: Towards a Standard CP Modelling Language. In Proceedings of the 13th International Conference on Principles and Pratice of Constraint Programming (CP 2007), volume 4741 of LNCS, pages 529-543. Springer, 2007.

17. Python. Python Software Foundation. https://www.python.org/

18. Z3 Theorem Prover. https://github.com/Z3Prover/z3/wiki

19. Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation, Volume 7 (2010), system description, pages 59-64. http://www.sat4j.org

20. Parallel Weighted Boolean Optimization Solver. http://sat.inesc-id.pt/pwbo/

21. DIMACS, Satisfiability Suggested Format. May 8, 1993. http://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf

22. The HSEval High School Timetable Evaluator. http://sydney.edu.au/engineering/it/ jeff/hseval.cgi

23. Toru Ogawa, YangYang Liu. Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT solvers. Graduate School of Information Science and Electrical Engineering. Kyushu University, Fukuoka Japan 819-0395.