Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Generació d'horaris d'institut amb operacions lògiques

Document: Memòria

Alumne:Ismael El Habri

Tutor: Josep Suy Franch i Jordi Coll Caballero
Departament: Informàtica, matemàtica Aplicada i Estadística
Àrea:

Convocatòria (mes/any) Setembre 2019

Índex

1. Introducció, motivacions, propòsit i objectius

La confecció d'horaris és un problema recorrent amb el que es troben els instituts que amaga una alta combinatòria, dificultant-ne molt la seva elaboració manual posat que s'han de prendre moltíssimes decisions a cegues, fent que sigui molt probable cometre errors en la confecció. En aquesta situació, amb la aparició dels computadors molts instituts varen decidir ajudar-se en aquesta tasca utilitzant eines informàtiques capaces d'explorar grans quantitats de combinacions per segon. No obstant a això, amb aquelles primeres eines es seguien tenint molts problemes per generar horaris viables, que normalment eren de qualitat baixa i requerien un refinament posterior que s'havia de fer de forma manual.

El HSTT (High School TimeTabling) consisteix en la solució de forma automàtica de aquest problema d'alta complexitat (NP). Així doncs, el que s'intenta és la configuració automàtica de horaris de institut partint de una sèrie de recursos (per exemple: aules, professors, assignatures, grups) i repartir-los de manera que sigui viable i tenint en compte de manera total o parcial les preferències del professorat en quant a horaris, continuïtat, grups, etc. Tot això fa que el problema sigui molt difícil de resoldre, degut al gran nombre de combinacions possibles entre els diferents recursos.

Afegint dificultat al problema, depenguen del país del qual estiguem parlant existeixen una gran diversitat de requisits propis, degut a les característiques pròpies del sistema d'estudis secundaris de cada lloc.

Degut a tot el mencionat encara no existeix cap eina capaç de trobar una solució òptima al problema de manera determinista en un temps raonable, encara que s'ha avançat molt en la resolució de problemes d'aquest tipus i costen de trobar instituts que optin per fer els horaris de forma manual.

Els objectius d'aquest treball són els següents:

- Aprofundir sobre el problema de la generació d'horaris en sí i sobre els problemes de satisfacció de restriccions en general i les tècniques que s'utilitzen per resoldre'ls, com ara, SAT i les seves diverses extensions.

- Desenvolupar un generador d'horaris automàtic utilitzant les tècniques estudiades anteriorment, aprofitant les eines relacionades que s'han desenvolupades recentment pel grup de recerca Lògica i Programació, que resolgui el problema en un temps raonable i intenti trobi la millor solució possible, cosa que es preveu molt complicada d'aconseguir en un temps raonable en les instàncies grosses.

1.1. Grup de recerca Lògica i Programació

Aquest treball s'emmarca dins del grup de recerca de Lògica i Programació de l'àmbit d'àrea tècnica de la Universitat de Girona.

El grup basa la seva recerca en l'estudi de satisfactibilitat de fórmules proposicionals booleanes (SAT) i Satisfiability Modulo Theories (SMT) i les seves aplicació per a la resolució de problemes combinatoris com ara problemes de *scheduling* i *planning* arribant a utilitzar amb èxit tècniques innovadores en altres àmbits com pot ser: els problemes de *scheduling* i *planning*.

Durant la elaboració del treball he rebut ajuda i assessorament dels membres del grup, incloent el meu tutor de projecte, el Dr. Josep Suy, qui també en forma part.

1.2. Estructura del Treball

A continuació s'exposa l'estructura d'aquest document:

1. Introducció, motivacions, propòsit i objectius: Situa el marc del projecte, explica les raons per les quals s'ha escollit el treball i en defineix els objectius.
2. Estudi de viabilitat: Justifica la capacitat de fer el projecte
3. Metodologia i Planificació: Tot i que a la guia van separats, s'ha decidit ajuntar-los per la gran relació que tenen entre ells. En aquest apartat s'exposa la metodologia que s'ha seguit a la hora de implementar el projecte i es defineix l'estratègia seguida per arribar als objectius.
4. Marc de treball i conceptes previs: S'exposa el treball previ realitzat sobre el problema, i els coneixements necessaris per a la implementació del programa.
5. Requisits del sistema: Requisits funcionals i no funcionals que ha de complir el sistema.
6. Anàlisi i disseny del sistema: Descripció del maquinari i programari utilitzat durant el desenvolupament del projecte.
7. Implementació i proves: Es detalla la implementació del programa, els problemes que han aparegut i les solucions que s'han donat a aquests.
8. Resultats: Detalla els resultats obtinguts. L'apartat d'implantació que segons la guía donada va aquí, s'ha descartat posat que en aquest treball no té l'objectiu d'implantar res.
9. Conclusions: Conclusió del projecte i els resultats
10. Treball futur: Possibles ampliacions, millores o treballs futurs que es poden realitzar.
11. Bibliografia: Referències utilitzades per desenvolupar el projecte.
12. Manual d'usuari i instal·lació: Especifica les passes a seguir per utilitzar el sistema creat en un ordinador.

2. Estudi de viabilitat

2.1. Viabilitat tècnica

Per tal de realitzar aquest projecte farà el software necessari per poder tractar un fitxer XML, codificar una instància del problema en C++ i resoldre-la amb un *solver* SAT o d'alguna extensió de SAT. Abans de iniciar el projecte, s'ha comprovat l'existència d'aquest software i que aquest estigués disponible de forma gratuïta, de fet, tot el software que s'utilitzi en aquest projecte serà de codi obert.

Per part dels requisits de maquinari, tot i que HSTT és un problema dur, després de veure el treball previ realitzat en aquest mateix problema i d'altres de característiques semblants, s'ha determinat que la meva màquina d'ús diari és més que suficient per a tal de realitzar el desenvolupament del generador d'horaris.

Pel que fa a la habilitat de desenvolupar aquest treball s'ha determinat que era possible degut a que el grup de recerca Lògica i Programació ha treballat i aconseguit solucionar problemes semblants de forma exitosa utilitzant tècniques innovadores. Utilitzant els avenços fets pel grup s'ha decidit que es podria desenvolupar el treball en el marc de temps desitjat.

2.2. Viabilitat econòmica

Degut a que tot el software utilitzat és gratuït, lliure i de codi obert, i que el codi que el grup de recerca m'ha cedit també ha estat gratuït. Per part del maquinari, només s'ha utilitzat el que ja tenia amb anterioritat, així que per aquesta banda tampoc ha suposat cap cost. Així doncs, aquest projecte no s'ha requerit de cap inversió ni despesa econòmica. Així doncs l'únic cost que ha suposat aquest treball han estat el temps necessari per fer-lo i les ganes que se li han hagut de ficar.

2.2.1 Pressupost

Tot i que, com s'ha explicat anteriorment, el desenvolupament d'aquest treball no té cost, s'ha decidit fer una simulació del que costaria contractar un programador per fer el treball.

	€/h	Hores	Cost
Programador	14	180	2520

A part també s'han de tenir en compte el maquinari utilitzat:

	Cost Total	Hores	€/h
Ordinador Principal	828	170	4.9
Ordinador Portatil Secundari	200	10	20
Total	1028	190	5.71

També cal tenir en compte que els tutors del treball m'han ajudat molt, i han dedicat part del seu temps en les reunions, intercanvis de mails i conversacions que s'han tingut sobre el treball.

	Hores
Josep Suy	20
Jordi Coll	10

3. Metodologia i Planificació

3.1. Metodologia

La part més important i grossa d'aquest treball és la creació d'un generador automàtic d'horaris utilitzant les eines oferides per el grup de recerca. Primer caldrà estudiar el problema (HSTT) i la seva duresa, per poder ser capaç d'entendre el què estem treballant. Des de aquí es procedirà a la implementació del programa, utilitzant la API SMT creada per el Dr. Jordi Coll, el qual es membre del grup de recerca de Lògica i Programació del departament de Informàtica, Matemàtica Aplicada i Estadística. Aquesta API ens estalviarà la codificació de les restriccions de cardinalitat i les restriccions pseudo-booleanes, apart de oferir-nos una interfície senzilla per poder implementar el model en diferents *encodings* i múltiples opcions.

Després de fer un estudi preliminar del problema i el treball previ respecte aquest, la metodologia de treball que s'ha seguit ha estat la de entregues periòdiques. Consistint en fer una reunió amb els tutors per decidir el pròxim pas del treball i jo dedicar-me durant un temps a fer aquest pas. Després d'aquest temps es convoca una altre reunió on es valora el treball fet i es decideix com seguir. Des de aquí es va repetint fins a acabar el projecte.

El model de desenvolupament que s'ha triat en aquest treball ha estat el model de prototips (o prototipatge), que consisteix en crear, com el nom indica, prototips del programari que es pretén crear, és a dir, versions incompletes del software que s'està desenvolupant. Generalment, un prototip només compleix alguns dels requeriments del sistema i pot no tenir res a veure amb el producte final. Aquest model presenta diferents beneficis:

- Permet obtenir *feedback* molt aviat en les etapes de desenvolupament.
- Augmenta la precisió de les estimacions de temps de dedicació a la implementació de les funcionalitats.
- Permet que els desenvolupadors es centrin en en treballar en les parts del sistema que comprenen i no haver de implementar tot el sistema de cop.

3.2. Planificació

Com s'ha dit en l'apartat anterior primer caldrà estudiar el problema HSTT i la seva duresa. Després es procedirà amb el disseny i la implementació del generador. Primer caldrà dissenyar implementar i testejar un *parser* pels fitxers. En aquest pas també cal pensar i implementar en quina estructura es guardaran aquestes dades i com es transferiran en el model que es codificarà posteriorment.

Al tenir el *parser* i l'estructura de dades enllestits caldrà començar a estudiar com funciona la API per C++ del Yices i posteriorment la API SMT del Dr. Jordi Coll. Això ens permetrà començar a dissenyar, codificar i testejar el model, que és el següent pas del treball. Al tenir enllestit el model, es faran les proves de rendiment amb diferents límits de optimització i diferents *encodings* de les restriccions de cardinalitat. Finalment s'implementarà una forma maca i llegible de mostrar els horaris generats.

Amb això el generador es donarà per acabat i es passarà a la confecció de la memòria del treball. A continuació un diagrama de Gantt mostrant la planificació feta.

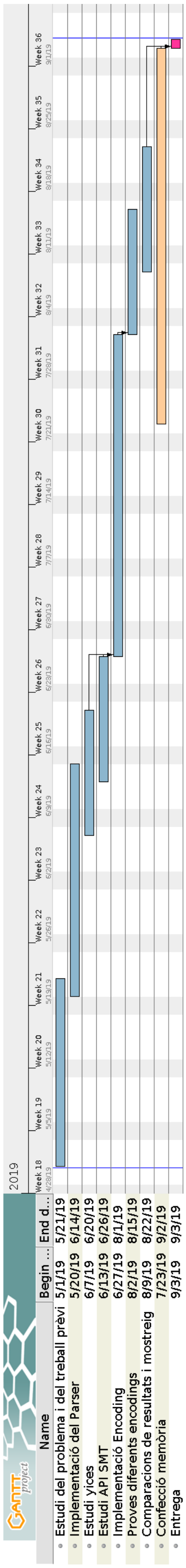

Figura 3.1: Diagrama de Gantt amb la planificació de treball del projecte

4. Marc de treball i conceptes prèvis

4.1. Marc de treball

Com s'ha mencionat en la introducció aquest treball s'emmarca dins del grup de recerca de Lògica i Programació (LAP) del departament de Informàtica, Matemàtica Aplicada i Estadística (IMAE) de la Universitat de Girona. En el treball s'utilitzaran les eines desenvolupades recentment en el grup de recerca, per ser exactes, la API SMT feta en C++ per el Dr. Jordi Coll. Aquesta API permet codificar problemes SAT, MaxSAT i SMT per a diferents *solvers* de forma transparent a aquest i te implementades les diferents restriccions de cardinalitat i pseudo-booleanes en les diferents possibles codificacions. També inclou diferents algoritmes d'optimització implementats.

La resta del treball previ seria el treball també sobre confecció d'horaris d'institut fet el 2015 per en Cristòfor Nogueira[1]

4.2. Definició del problema

El problema que es treballa és el de la confecció d'horaris per a institut (HSTT de *High School Time Tables*). Aquest consisteix en assignar a cada assignatura que es fa en un centre l'espai de temps en que s'impartirà i el conjunt de recursos que utilitzarà. Els recursos normalment seran professors i aules, però es contemplen altres possibles necessitats especials de cada centre, per això es generalitza.

La duresa de aquest problema es troba en assignar un espai de temps per a cada assignatura donant-li els recursos que necessita sense violar cap restricció que aquests tinguin, com podria ser no fer dues assignatures a la vegada en la mateixa aula, o sigui, que al realitzar-se la assignatura tots els seus recursos estiguin disponibles. De la mateixa manera es poden imposar diverses restriccions de naturaleses diferents i amb cada una s'aniran reduint les possibles combinacions vàlides i fent més i més difícil la generació del horari.

4.2.1 Estudi de duresa

4.2.1.1 Incís en la teoría de la computació

Abans de procedir a l'estudi de la duresa del problema HSTT, caldrà explicar els següents conceptes de la teoria de la computació:

Problemes decidibles i indecidibles Un problema decidible és aquell per el qual existeix una màquina de Turing que para en totes les entrades possibles amb una resposta: sí o no. Aquests problemes també són coneguts com a Turing Decidibles. Així doncs, un problema decidible és aquell pel qual sempre podrem construir un algorisme que sempre respon el problema.

Un problema pot ser semi-decidible, això passa quan una màquina de Turing quan l'entrada és acceptada, però es pot penjar o es pot parar quan l'entrada es rebutja. Aquests problemes també son referits com a Turing Reconeixibles.

Un problema indecidible és aquell pel qual no podem construir un algorisme que resolgui el problema en temps finit. Aquests problemes poden ser parcialment decidibles, però sempre hi haurà una condició que portara la màquina de Turing a bucle infinit.

P, NP i NP-Completesa

- **P:** És el conjunt de problemes que poden ser resolts en temps polinòmic amb una màquina de Turing determinista.
- **NP:** És el conjunt de problemes que poden ser resolts en temps polinòmic amb una màquina de Turing no determinista.
- **NP-Complet:** És el conjunt de problemes més durs en el conjunt NP. Un problema C és NP-Complet si C és NP i tot problema NP és reduïble a C.
- **NP-Hard:** És el conjunt de problemes als quals es pot reduir tot problema NP. O sigui, un problema C és NP-Hard si tot problema NP és reduïble a C.
- **Reducció:** Si tenim dos problemes L_1 i L_2 i tenim un algoritme A_2 que resol L_2. Reduir L_1 a L_2 és transformar el problema L_1 a L_2 així poder utilitzar el algoritme A_2 per resoldre el problema, creant així un algoritme A_1 amb l'estructura que es pot veure en la figura 4.1

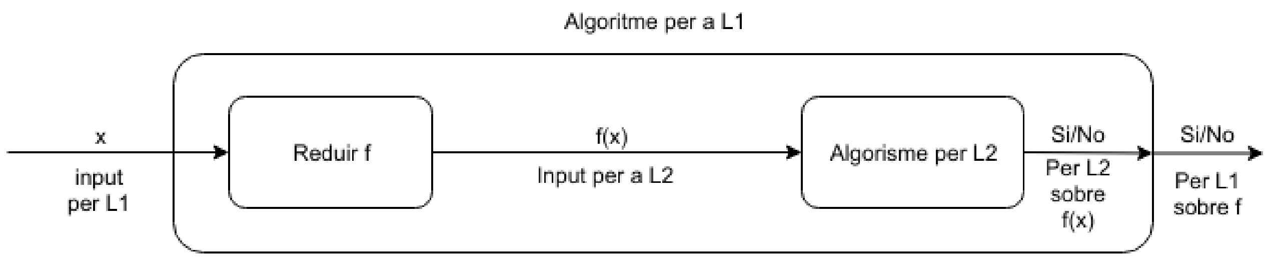

Figura 4.1: Esquema de Reducció

4.2.1.2 Duresa de HSTT

El problema HSTT és clarament decidible i, posat que comprovar si una solució satisfà totes les restriccions imposades és d'ordre polinòmic, aquest pertany al conjunt de problemes NP.

NP-Completesa de HSTT

En aquest apartat s'inclou la demostració que es veu en el treball d'en Cristòfor Nogueira[1].

Es pot demostrar la NP-Completesa del problema HSTT reduint un problema NP-Complet conegut a HSTT, en aquest cas, el problema de la motxilla[2], d'acord amb la demostració proposada per B. Cooper i J.H. Kingston [3].

Una de les fonts de la duresa del problema ve a l'hora de gestionar els recursos de manera coherent. Així 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 premisses. En cas q 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 simultània 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 a una assignatura alhora.

Diem que dues 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'ítems $U = \{u_1, u_2, ..., u_n\}$, cadascun amb un pes associat w_i, es poden col·locar en un conjunt de motxilles $B = \{b_1, b_2, ..., b_m\}$, 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:

$$Times = \{t_1, 1, ..., t_n, m\}$$

$$Events = X \cup Y, X = \{x_1, ..., x_n\} Y = \{y_1, ..., y_m\}$$

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 $\{t_i, 1...t_{i,c_i}\}$

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 de col·locat l'ítem d'entrada. Llavors, per cada assignatura x_i, escollim w_i espais de temps, q no hagin estat escollits prèviament, del conjunt $S_k = \{t_k, 1...t_{k,c_k}\}$, on $k = f(u_i)$. Es a dir, s'escullen tants espais de temps com el pes de l'ítem que representa de manera que no hi hagin 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 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. Podríem reimplementar, 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 quint event y_i es solapa, de manera que $f(u_i) = j$.

4.3. Format XHSTT

Un dels problemes que presenta HSTT és la complexitat que té representar una instància amb totes les possibles restriccions possibles. Per això s'ha optat utilitzar el format genèric per a representar les instàncies de HSTT anomenat XHSTT (de *Xml-format High School TimeTabling*) utilitzat per HSEval[4]. Aquest format és obert i preveu l'addició d'elements i restriccions, així que en aquest treball només es tindran en compte un subconjunt d'ells.

Aquest format utilitza quatre tipus de fills en les instancies:

- *Times* pels espais de temps.
- *Resources* pels recursos.
- *Events* pels events, com ara assignatures.
- *Constraitns* per les restriccions.

4.3.1 Temps, Recursos i Events

Temps
En aquest tipus de fill es defineixen els múltiples espais de temps, i opcionalment els grups de espais de temps. Els Grups de espais de temps (*TimeGroups*) poden ser de tres tipus: *Week*, *Day* i *TimeGroup*. Cada grup consisteix en un nom i un identificador.

Un espai de temps es defineix amb la clau *Time*. Aquesta es pot relacionar amb un grup d'espais de temps utilitzant l'identificador d'aquest. Apart d'això té un nom i un identificador.

A continuació un exemple amb el bloc d'espais de temps.

```
<Times>
    <TimeGroups>
        <Day Id="DayId">
            <Name>DayName</Name>
        </Day>
        <Week Id="WeekId">
            <Name>WeekName</Name>
        </Day>
        <TimeGroup Id="TimeGroupId">
            <Name>TimeGroupName</Name>
        </TimeGroup>
```

```
    </TimeGroups>
    <Time Id="TimeId">
        <Name>Time Name</Name>
        <Day Reference="DayId"/>
        <Week Id="WeekId"/>
        <TimeGroups>
            <TimeGroup Reference="TimeGroupId"/>
        </TimeGroups>
    </Time>
<\Times>
```

Recursos
Cada recurs necessita d'un tipus, com ara aules, professors, classes, etc. Aquests es poden definir amb *ResourceTypes*. A més a més, de la mateixa forma que amb els espais de temps, es poden definir grups de recursos, però a part dels paràmetres mencionats amb els grups de temps, s'ha de incloure el tipus de recursos que inclou.

A continuació un exemple:

```
<Resources>
    <ResourceTypes>
        <ResourceType Id="Room">
            <Name>Room</Name>
        </ResourceType>
    </ResourceTypes>
    <ResourceGroups>
        <ResourceGroup Id="Rooms">
            <Name>Rooms</Name>
            <ResourceType Reference="Room"/>
        </ResourceGroup>
    </ResourceGroups>
    <Resource Id="Room1">
        <Name>Room1</Name>
        <ResourceType Reference="Room"/>
        <ResourceGroups>
            <ResourceGroup Reference="Rooms"/>
        </ResourceGroups>
    </Resource>
</Resources>
```

Events
Al definir un Event cal especificar els tipus de recursos que necessita (es poden assignar recursos concrets), la seva duració i, opcionalment, a quin grup d'events pertany. A més, es pot afegir el rol que té cada recurs en aquest event.

A continuació un exemple:

```xml
<Events>
    <EventGroups>
        <Course Id="gr_T1-S1">
            <Name>T1-S1</Name>
        </Course>
        <EventGroup Id="gr_AllEvents">
            <Name>All Events</Name>
        </EventGroup>
    </EventGroups>
    <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 Reference="T1">
                <Role>Teacher</Role>
                <ResourceType Reference="Teacher"/>
            </Resource>
        </Resources>
        <EventGroups>
            <EventGroup Reference="gr_AllEvents"/>
        </EventGroups>
    </Event>
</Events>
```

4.3.2 Restriccions

Aquí s'enumeraràn els diferents tipus de restriccions definides en el format.[1]

Com a pautes generals, cada restricció tindrà els següents camps:

- *Name*: un nom.
- *Required*: ens diu si el generador té permès (*false*) o no (*true*) violar la restricció . O sigui, si és una *Soft Constraint* o una *Hard Constraint*.
- *Weight*: ens indica el pes de la restricció.
- *CostFunction*: Ens indica la funció que segueix el cost.
- *AppliesTo*: Grups o Elements als quals s'aplica la restricció.

[1]Més informació a `http://www.it.usyd.edu.au/~jeff/cgi-bin/hseval.cgi?op=spec&part=constraints`

Assign Time Constraints

Restricció que imposa que no hi hagi espais de temps sense assignar.

A continuació un exemple:

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

Split Events Constraints

Restricció que indica com s'han de partir les diferents assignatures, indicant la duració mínima i màxima de cada impartició d'un event o grup d'events.

A continuació un exemple:

```xml
<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>
```

Distribute Split Events Constraints

Restricció que limita el nombre de lliçons de una duració determinada d'un grup d'events. Per exemple, per fer que totes les lliçons de Matemàtiques siguin de duració 2.

A continuació un exemple:

```xml
<DistributeSplitEventsConstraint Id="DistributeSplit_1">
    <Name>At least 1 double lesson(s)</Name>
    <Required>false</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <EventGroups>
            <EventGroup Reference="gr_T1-S1"/>
            <EventGroup Reference="gr_T1-S2"/>
            <EventGroup Reference="gr_T1-S3"/>
            <EventGroup Reference="gr_T3-S1"/>
            <EventGroup Reference="gr_T3-S2"/>
            <EventGroup Reference="gr_T3-S3"/>
            <EventGroup Reference="gr_T4-S1"/>
            <EventGroup Reference="gr_T4-S2"/>
            <EventGroup Reference="gr_T4-S3"/>
            <EventGroup Reference="gr_T8-S1"/>
            <EventGroup Reference="gr_T8-S2"/>
            <EventGroup Reference="gr_T8-S3"/>
        </EventGroups>
    </AppliesTo>
    <Duration>2</Duration>
    <Minimum>1</Minimum>
    <Maximum>1</Maximum>
</DistributeSplitEventsConstraint>
```

Prefer Times Constraints

Restricció que indica temps determinats per a certs events. Per exemple per evitar que events de més de una hora a la última hora del dia i acabin a la primera hora del dia següent.

A continuació un exemple:

```xml
<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>
```

Spread Events Constraints

Restricció que indica que els events d'un grup concret san de separar en el temps.

A continuació un exemple:

```
<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"/>
            <EventGroup Reference="gr_T1-S3"/>
        </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>
```

Avoid Clashes Constraint

Restricció que especifica que cap dels recursos al que s'aplica pot assistir a més d'un event a la vegada. Cal notar que el format permét que un recurs pugui assistir a més dun event a la vegada.

A continuació un exemple:

```
<AvoidClashesConstraint Id="NoResourceClashes">
    <Name>NoResourceClashes</Name>
    <Required>true</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
```

```xml
        <ResourceGroups>
            <ResourceGroup Reference="gr_Teachers"/>
            <ResourceGroup Reference="gr_Classes"/>
        </ResourceGroups>
    </AppliesTo>
</AvoidClashesConstraint>
```

Avoid Unavailable Times Constraints

Restricció que indica que hi ha certes hores durant les quals certs recursos no estan disponibles. Útil per a professors que no treballen cert día o prefereixen no fer-ho en certes hores.

A continuació un exemple:

```xml
<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>
```

Limit Idle Times Constraint

Restricció que límita el número d'espais de temps en qué un recurs o grup de recursos no està ocupat.

A continuació un exemple:

```xml
<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>
```

Cluster Busy Times Constraint

Restricció que limita el nombre d'hores en que un recurs pot estar ocupat.

A continuació un exemple:

```
<ClusterBusyTimesConstraint Id="MaxNofDaysConstraint_T_days_2">
    <Name>Not more than 2 days with lessons</Name>
    <Required>false</Required>
    <Weight>9</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <Resources>
            <Resource Reference="T1"/>
            <Resource Reference="T2"/>
            <Resource Reference="T3"/>
            <Resource Reference="T4"/>
            <Resource Reference="T5"/>
            <Resource Reference="T7"/>
            <Resource Reference="T8"/>
        </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>2</Maximum>
</ClusterBusyTimesConstraint>
```

4.4. Estat de l'art

En aquest apartat del treball es repassaran diverses tecnologies i conceptes que ens podrien ajudar a resoldre el problema HSTT. Com s'ha vist anteriorment HSTT és un problema de *scheduling* que pertany a NP-Complet i, per tant, no es coneix un mètode determinista en temps polinòmic que pugui resoldre el problema.

Existeixen diferents aproximacions per problemes del tipus de HSTT, en aquest treball es centrarà en els mètodes deterministes que garanteixen la optimitat. Aquests mètodes busquen la millor solució al problema entre totes les combinacions que respecten totes les condicions imposades per la instància del problema que volem resoldre. HSTT entra en la categoria de tècniques basades en programació per restriccions i les tècniques basades en reduccions de altres problemes NP-complets com SAT, SMT, etc.

4.4.1 Problemes de Satisfacció de Restriccions

Els problemes de Satisfacció de Restriccions (CSP de *Constraint Programming Problem*) són una representació de problemes combinatoris. Un CSP està format per un conjunt finit de variables, cada una de les quals te un domini, i un conjunt de restriccions. Cada restricció esta definida sobre un subconjunt de les variables i en restringeix els valors que poden agafar. La idea és trobar una assignació de variables que compleixi totes les restriccions imposades. En alguns problemes, l'objectiu es trobar-les totes, o trobar la millor, si hi ha alguna forma de determinar quines solucions són millors que d'altres utilitzant una formula objectiu.

4.4.2 SAT

SAT prové de SATISFIABILITY, que és la abreviació de *Boolean Satisfiability Problem.* El problema SAT consisteix en, donada una fòrmula de lògica proposicional (una expressió booleana), determinar una assignació de variables (model) per el qual la fórmula sigui certa, o la determinació de que no existeix tal assignació. Per exemple, per a la fòrmula $A \vee \neg B$ és satisfactible amb $A = Cert$ i $B = Fals$ posat que farien la fórmula certa; en canvi per la fórmula $A \vee \neg A$ no hi ha cap assignació que la faci certa, per tant en diríem insatisfactible. SAT consisteix doncs, en determinar si una fòrmula booleana és o no satisfactible.

SAT va ser el primer problema en ser demostrat que era NP-Complet el 1971 per Steven Cool[5], per tant, tal i com s'ha dit anteriorment, tot problema NP es pot reduir a SAT. El fet de que es pot reduir qualsevol problema decidible a SAT en temps polinòmic i la seva simplicitat de formulació el fan un problema d'especial interès en la comunitat científica, generant grans quantitats d'avenços en aquest camp, però, òbviament, tot i així, encara no existeix cap forma de resoldre'l en temps polinòmic ni s'ha demostrat que es pugui (això seria demostrar si P=NP o no![2] per més informació sobre P=NP)

Representació formal, CNF

[2] https://en.wikipedia.org/wiki/P_versus_NP_problem

CNF prové de *Conjunctive Normal Form* que es tradueix com a Forma Normal Conjuntiva. En lògica booleana una fòrmula esta en CNF si és una conjunció de una o més clàusules i aquestes són una disjunció de literals. O sigui, una AND de ORs. Tota fórmula proposicional pot ser transformada a CNF. Aquesta transformació es basa en les regles d'equivalències lògiques: la doble negació, les lleis De Morgan i la propietat distributiva

4.4.3 Extensions de SAT

4.4.3.1 MaxSAT

MaxSAT de *Maximum SATisfiability problem*és una generalització de SAT que consisteix en trobar el màxim nombre de clàusules d'una fórmula booleana en CNF que es poden satisfer. Es pot definir una versió de MaxSAT amb pesos: donada una fórmula CNF assignem pesos no negatius a cada clàusula i busquem la assignació de variables que maximitzen el pes sumat de les clàusules satisfetes. Es pot considerar que MaxSAT seria una instància de aquesta versió on tots els pesos son 1.

4.4.3.2 SMT

De *Satisfiability Modulo Theories*, SMT és una generalització de SAT on algunes de les variables proposicionals tenen el paper de predicats amb interpretacions predefinides a d'altres teories. Existeixen diversos tipus de teories: d'igualtat, d'aritmètica lineal, entera, d'aritmètica lineal mixta, d'arrays, de BitVectors, etc.

Exemple de fórmula SMT: $p \vee q \vee (x < 5) \vee (y < x)$

Una teoria es defineix com a conjunt de fórmules lògiques de primer ordre tancades sota conseqüència booleana, o sigui, s'han de poder reduir a un resultat booleà.

4.4.4 Cardinality Encodings

Les *cardinality constraints* són aquelles que expressen límits numèrics en quantitats discretes, sorgeixen freqüentment en la codificació de problemes del món real: per exemple un enginyer vol expressar que almenys n parts d'un cert tipus son necessàries per fer funcionar el producte.

Podem definir dos tipus de *cardinality constraints*:

- Les que comparen amb 1: *Exactly One (EO)*, *At Most One (AmO)* i *At Least One (ALO)*
- Les que comparen amb k: *Exactly k (EK)*, *At Most k (AMK)* i *At Least k (ALK)*

4.4.4.1 Cardinality Constraints comparadaes amb 1

ALO i EO Per codificar una *ALO* a CNF consisteix en una clàusula amb totes les variables sobre les quals s'aplica. Al ser una OR, per complir la clàusula, una de les variables haurà de ser certa.

Per altre banda per codificar una *EO*, només cal codificar una ALO i una *AMO* per el conjunt de variables.

AMO Per al AMO existeixen diferents encodings. A continuació en veurem alguns:

- Quadràtic: consisteix en en fer clàusules binaries (mutexes) del tipus:

$$\neg x_i \vee \neg x_j \qquad i \in 1..n-1, j \in i+1..n$$

  Aquest encoding introdueix $O(n^2)$ clàusules binaries, però no introdueix variables auxiliars.

- Logarítmic: aquest encoding consisteix en introduir $O(log_2 n)$ variables auxiliars i $O(nlog_2 n)$ clàusules i consisteix en generar clàusules binàries per $i \in 0..n-1, j \in 0..m-1$ on m és $log_2(n)$:

  [label=$\circ$] $x_i \rightarrow \neg y_i$, si el bit j del nombre i és 0.
  $x_i \rightarrow y_i$, si el bit j del nombre i és 1.

- Ladder encoding: Aquest encoding introdueix $O(n)$ variables auxiliars a_i. Cada variable d'aquestes expressa *ALO(*$\{x_0, ..., x_i\}$*)* és cert.
  Per cada $i \in 0...n-1$ hi haurà 3 clàusules:
  - $\neg x_i \vee a_i$
  - $\neg a_i \vee a_{i+1}$ (aquesta clàusula només per $i \in 0...n-2$)
  - $\neg a_i \vee \neg x_{i+1}$ (aquesta clàusula només per $i \in 0...n-2$)

- Heule encoding: Aquest encoding afegeix $O(n)$ variables auxiliars i introdueix $O(n)$ clàusules. Consisteix en fer el següent:

  - Si $n \leq 3$, l'encoding és el mateix que el quadràtic.
  - si $n \geq 4$, introdueix una variable i codifica $AMO(\{x_0, x_1, y\})$ i $AMO(\{x_2, x_3, ..., \neg y\})$ de forma recursiva.

4.4.4.2 Cardinality Constraints comparadaes amb k

Sorting networks

Una sorting network agafa una entrada de mida n i l'ordena. Es pot fer de manera recursiva com es pot veure a continuació, utilitzant una estratègia semblant a la del *mergesort*:

- Si $n = 1$, la sortida de la sorting network és:

$$\text{Sorting}(x_0) = x_0$$

- Si $n = 2$, la sorting network és un sól merge (un comparador entre 2):

$$\text{Sorting}(x_0, x_1) = \text{Merge}(x_0, x_1)$$

- Si $n > 2$, agafem una l amb $0 \leq l < n - 1$: Definim:

$$\begin{aligned}(z_0, z_2, ..., z_{l-1}) &= \text{Sorting}(x_0, x_1, ..., x_{l-1}),\\ (z_l, z_{l+1}, ..., z_{n-1}) &= \text{Sorting}(x_l, x_{l+1}, ..., x_{n-1}),\\ (y_0, y_1, ..., y_{n-1}) &= \text{Merge}(z_0, z_1, ..., z_{l-1}; z_l, ..., z_{n-1}).\end{aligned}$$

  Així ens queda que :$\text{Sorting}(x_0, x_1, ..., x_{n-1}) := (y_0, y_1, ..., y_{n-1})$.

El nombre de variables i clàusules introduides per una sorting network de mida n es pot calcular recursivament. Si aquesta està formada per dues *sorting networks* de mida l i $n - l$, introduirem $V_1 + V_2 + V_3$ variables i $C_1 + C_2 + C_3$ clàusules, on $(V_1, C_1)i(V_2 iC_2)$ són les variables i clàusules utilitzades en les dues *sorting networks* internes i (V_3, C_3) són el nombre de variables i clàusules necessàries en el merge de les entrades de mida $(l, n - l)$.

Totalizer

El Totalizer encoding es basa en generar una representació unària de la cardinalitat del conjunt de variables d'entrada. Va ser introduït per Bailleux i Bougkhad[6]. L'encoding consisteix en l'escriptura d'un arbre binari on a les fulles hi han les variables d'entrada i cada uns interior, conté la representació unària de la cardinalitat dels seus descendents fulla. Per tant amb tantes variables auxiliars com descendents fulla tingui.

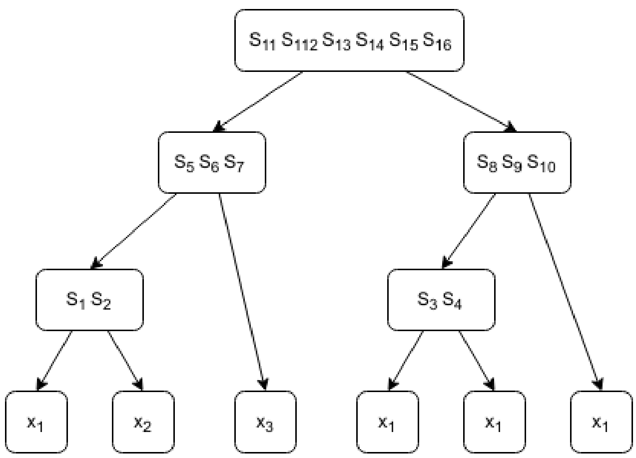

Figura 4.2: Representació Gràfica del arbre d'un totalizer

Seguint amb l'exemple de la figura 4.2 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 $\{s_1 1, s_1 2, s_1 3, s_1 4, s_1 5, s_1 6\}$. És important notar que cada node codifica la cardinalitat de les fulles q en pengen, en format unari. Això significa que per tot i de cada node, $s_i \geq S_{i+1}$.

En la figura també es poden veure totes les variables auxiliars que introdueix l'encoding per codificar un *at-most-k* sobre 6 variables. Per veure quines clàusules genera, caldria definir un sumador unari, que donats dos nombres unaris, en retorni la suma en unari. Aplicant-lo a cada node interior és suficient per garantir que $\{s_1 1, s_1 2, s_1 3, s_1 4, s_1 5, s_1 6\}$ és la representació unària de la cardinalitat de $\{x_1, x_2, x_3, x_4, x_5, x_6\}$. Amb la representació unària de la cardinalitat, n'hi ha prou amb imposar que l'interpretació de out_{k+1} sigui falsa, per fer efectiu un, *at-most-k*.

5. Requisits del sistema

En aquest treball es pretén desenvolupar un generador de horaris automàtic amb SAT i/o SMT que serà capaç de rebre una instància XHSTT en un fitxer XML passat per paràmetre, llegir-lo, codificar-ne un model utilitzant la API SMT desenvolupada pel Dr. Jordi Coll, resoldre'l i retornar un resultat, mostrant-lo per pantalla, o retornant un fitxer amb la instància XHSTT amb la nova solució inserida.

Per fer-ho, serà necessari l'ús de un conjunt de llibreries i programari:

- Llibreria que permeti el tractament de fitxers XML, S'ha optat per pugixml[1]
- Posat que la API SMT utilitza Yices 2[2] com a únic *solver* SMT, necessitarem tenir-ne la seva llibreria instal·lada.
- Posat que la API SMT està desenvolupada per GNU/Linux, serà necessari treballar una màquina que tingui instal·lat una distribució Linux.

Apart de tot el mencionat, també ha calgut decidir un país per el qual el generador d'horaris estarà desenvolupat. Això és degut a que en funció del país els requisits dels horaris generats varien àmpliament en funció de les característiques pròpies del sistema educatiu de cada país. En aquest treball s'ha decidit optar per les instàncies de Brazil, degut que tenen tots els recursos assignats a algun event, de manera que només es demana al generador organitzar els events, reduint l'espai de cerca.

[1] https://pugixml.org/

[2] https://yices.csl.sri.com/

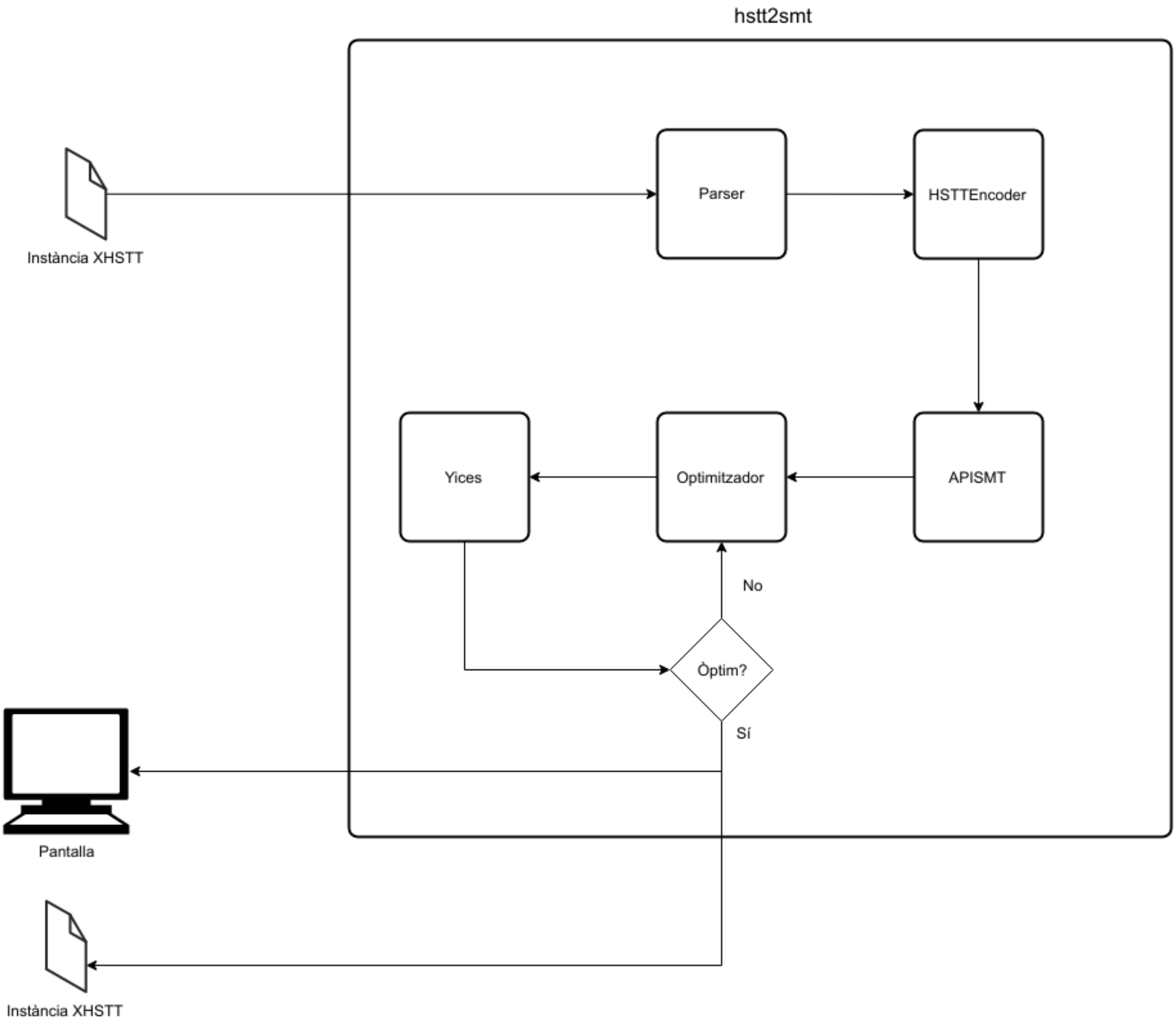

Figura 5.1: Arquitectura del programa

6. Estudis i decisions

6.1. Programari utilitzat

En aquesta secció del treball es veurà tot el programari que s'ha utilitzat per la confecció d'aquest.

6.1.1 Yices 2

Figura 6.1: Logo Yices 2

El Yices 2 és un *SMT Solver Open Source*[1] sota llicència GPL que decideix la satisfactibilitat de fòrmules que contenen símbols de funcions no interpretades amb igualtat, aritmètica real i entera, BitVectors, tipus escalars i tuples. El Yices 2 suporta aritmètica linear i no linear.

El Yices 2 pot processar fitxers d'entrada en notació SMT-LIB[2], es pot usar alternativament el llenguatge propi del Yices 2 i també te una API per C i C++.

En la implementació d'aquest treball, s'usa dins de la API SMT del Dr. Jordi Coll com un dels *solvers* que es poden utilitzar i és l'únic que permet SMT d'entre ells. S'ha decidit usar aquest a que és dels millors que hi ha i el grup de recerca hi te experiència prèvia.

S'ha utilitzat la versió 2.6.1.

6.1.2 pugixml

El pugixml és una llibreria C++ lleugera per al processament XML. És extremadament portable amb distribucions. També és opensource[3] sota llicència MIT.

pugixml permet un processament de documents XML molt ràpid, còmode i eficient amb la memòria. Tot i això, al tenir un *parser* DOM, no pot processar fitxers XML que no càpiguen a la memòria.

[1] Repositori GitHub: `https://github.com/SRI-CSL/yices2`
[2] `http://smtlib.cs.uiowa.edu/`
[3] Repositori GitHub: `https://github.com/zeux/pugixml`

Degut a les característiques mencionades i a experiència prèvia amb la llibreria, s'ha decidit utilitzar-la per a la programació del *parser* XHSTT.

S'ha utilitzat en la versió 1.9-1.

6.1.3 C++

Figura 6.2: Logo C++

Llenguatge de programació de propòsit general creat per Bjarne Stroustrup com a extensió del llenguatge de programació C. Des de llavors, el llenguatge s'ha expandit molt i el C++ permet programació orientada a objectes, genèrica i funcional, a més de facilitats per manipulació de memòria a baix nivell. Pràcticament sempre és implementat com a llenguatge compilat.

El compilador per a C++ que s'ha utilitzat es el G++ inclòs en el GNU Compiler Collection, que inclou compiladors per a C, C++, Objective-C, Fortran, Ada, Go i D, a més de llibreries per aquests llenguatges.

Figura 6.3: Logo G++

En aquest treball s'ha decidit utilitzar el C++ perquè és amb el que es treballa al grup de recerca i és amb el que està feta la API SMT. El C++ s'ha utilitzat en la revisió C++17 del seu estàndard. El GCC utilitzat ha estat en la versió 9.1.0-2.

Figura 6.4: Logo LaTeX

6.1.4 QtCreator

6.1.5 LaTeX

Aquesta memòria s'ha confeccionat amb LaTeX, que és un sistema de composició d'alta qualitat que inclou funcions dissenyades per a la creació de documents tècnics i científics amb la intenció de ajudar al creador a centrar-se més en el contingut que en la forma. LaTeXés l'estàndard de facto per a la comunicació i publicació de documents científics.

Per la confecció i edició del document en sí, s'ha utilitzat l'editor Open Source[4] Visual Studio Code. S'ha utilitzat en la versió 1.37.1-2 amb extensions per a facilitar l'edició del document tex.

6.1.6 GNU/Linux

Figura 6.5: Logo Linux

Linux és una familia de sistemes operatius formats pel *kernel* Linux juntament amb les utilitats GNU. El generador que s'ha fet en aquest treball ha estat desenvolupat per a Linux, degut a la facilitat d'accés a les múltiples llibreries que s'han utilitzat, a que la API SMT ha estat desenvolupada també per a Linux i simplement per preferència personal, degut a que el sistema operatiu que utilitzo dia a dia és una distribució Linux.

En concret s'ha treballat amb la distribució Arch Linux en la versió del kernel Linux 5.29.arch1-1 al final del treball (s'han usat versions anteriors que han anat sortint mentre es desenvolupava el treball).

[4] Link GitHub: `https://github.com/Microsoft/vscode`

6.2. Maquinari utilitzat

Per efectuar les proves de rendiment s'ha utilitzat un ordinador amb les següents especificacions:

- Processador AMD Ryzen^{TM} 3 1200 a 3.5 GHz amb 4 nuclis físics, 10MB de memòria cau i arquitectura 64 bits.
- 16GB de memòria RAM DDR4 a 3333MT.
- Sistema operatiu Arch Linux 64 bits amb kernel Linux 5.2.9.arch1-1

7. Anàlisis i disseny del sistema

7.1. Anàlisis

7.1.1 Necessitats del sistema

Les necessitats principals del sistema són les següents:

- Necessitem rebre un fitxer de l'usuari.
- Necessitem llegir les dades de un fitxer XHSTT tal i com s'ha explicat anteriorment, per tant requerirem de un *parser* per fer-ho.
- Necessitarem guardar les dades i les restriccions de alguna manera.
- Necessitarem un model lògic i codificar-lo utilitzant la API SMT del Dr. Jordi Coll.
- Necessitarem processar i guardar les dades de manera que ens faciliti el mostrar-les de forma que es pugui entendre.
- Necessitarem comprovar en la mesura del possible que la instància XHSTT sigui correcta.

7.1.2 Anàlisis de processos

L'usuari cridarà el programa, el programa llegirà el fitxer XHSTT que li ha passat l'usuari per paràmetre i s'encarregarà de codificar el model pel yices i cridar-lo per resoldre'l, utilitzant la llibreria per C++ pròpia del Yices 2.

7.2. Disseny

7.2.1 Interfícies d'usuari

El programa funcionarà via consola. Les dades necessàries, com ara el fitxer amb la instància, es passaran per paràmetres, utilitzant el sistema existent en la API SMT del Dr. Jordi Coll.

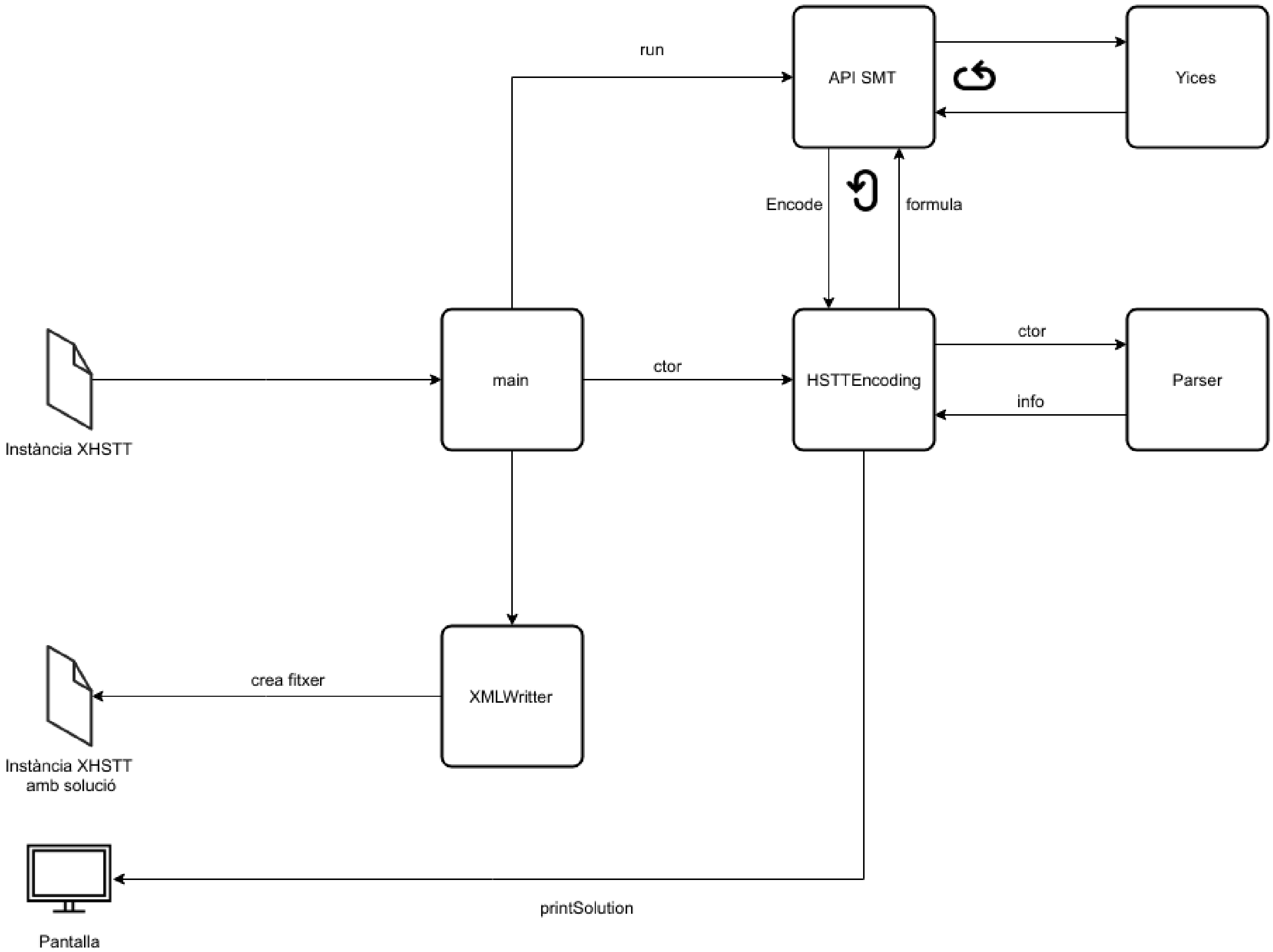

Figura 7.1: Esquema de processos

7.2.2 Model de dades

El model de dades correspondria al següent diagrama:

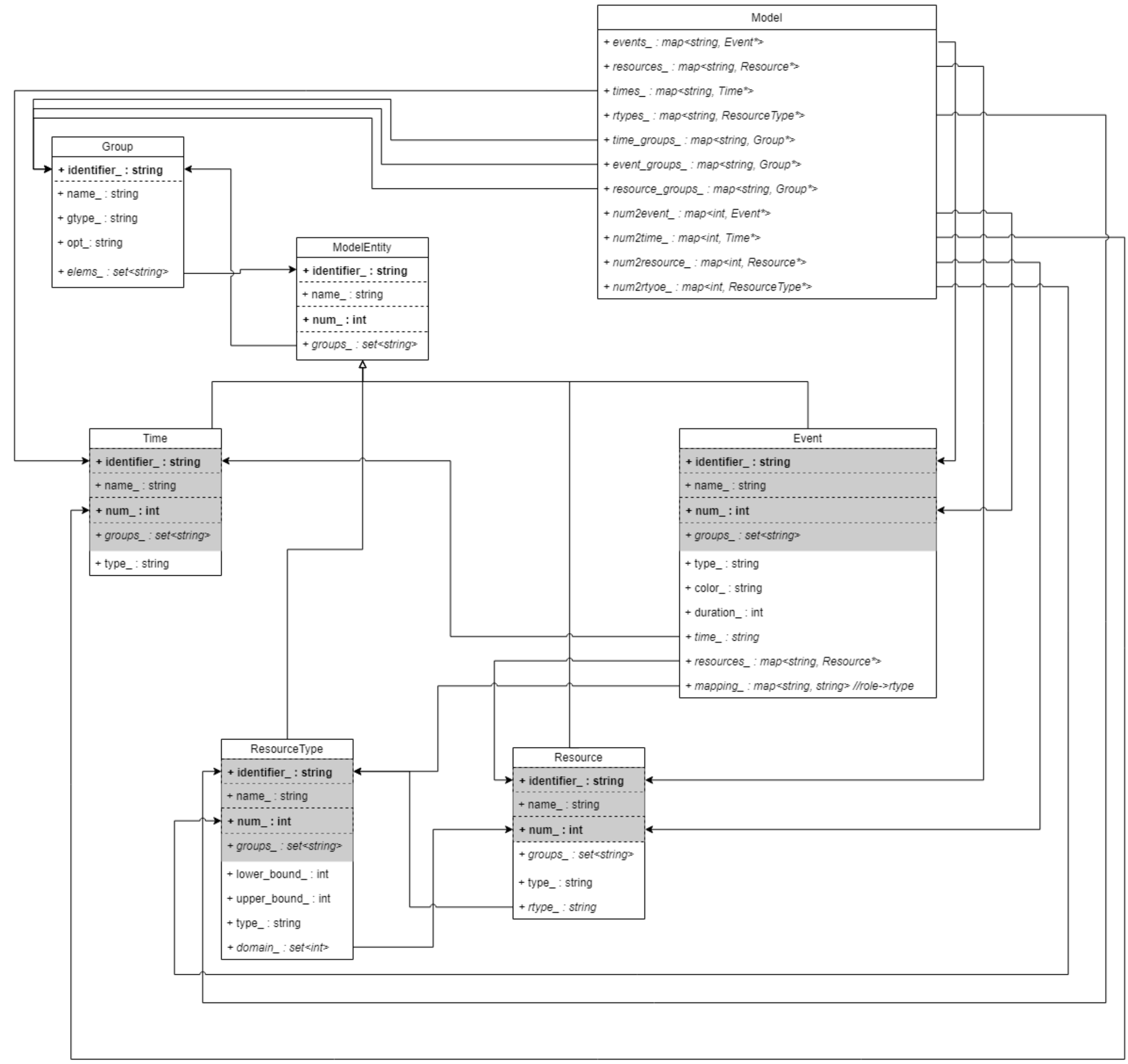

Figura 7.2: Model de Dades

Com es pot veure en la figura anterior, la classe *Model* guarda la informació de la instància (sense incloure les restriccions). D'aquesta heretaràn tots els encodings que es puguin arribar a fer, posat que sempre serà el mateix. Per fer-ho s'ha creat la classe *ModelEntity* de la qual hereten les classes *Event*, *Resource*, *ResourceType* i *Time*, representant respectivament a les assignatures, els recursos, els tipus de recursos i els espais de temps disponibles. Aquestes classes mantenen tota la informació necessària per a representar a cada tipus d'element. La classe *ModelEntity* inclou un identificador unic de tipus *string* i un numero enter unic per cada un dels tipus existents. Hi poden haver un *Event* i un *Resource* amb el mateix número, pero mai hi haura dos elements del mateix tipus amb el mateix numero.

A més s'ha creat una classe *Group* per tal de mantenir els diferents grups d'elements d'elements que preveu XHSTT.

7.2.3 Model d'objectes

El model d'objectes correspondria al següent diagrama:

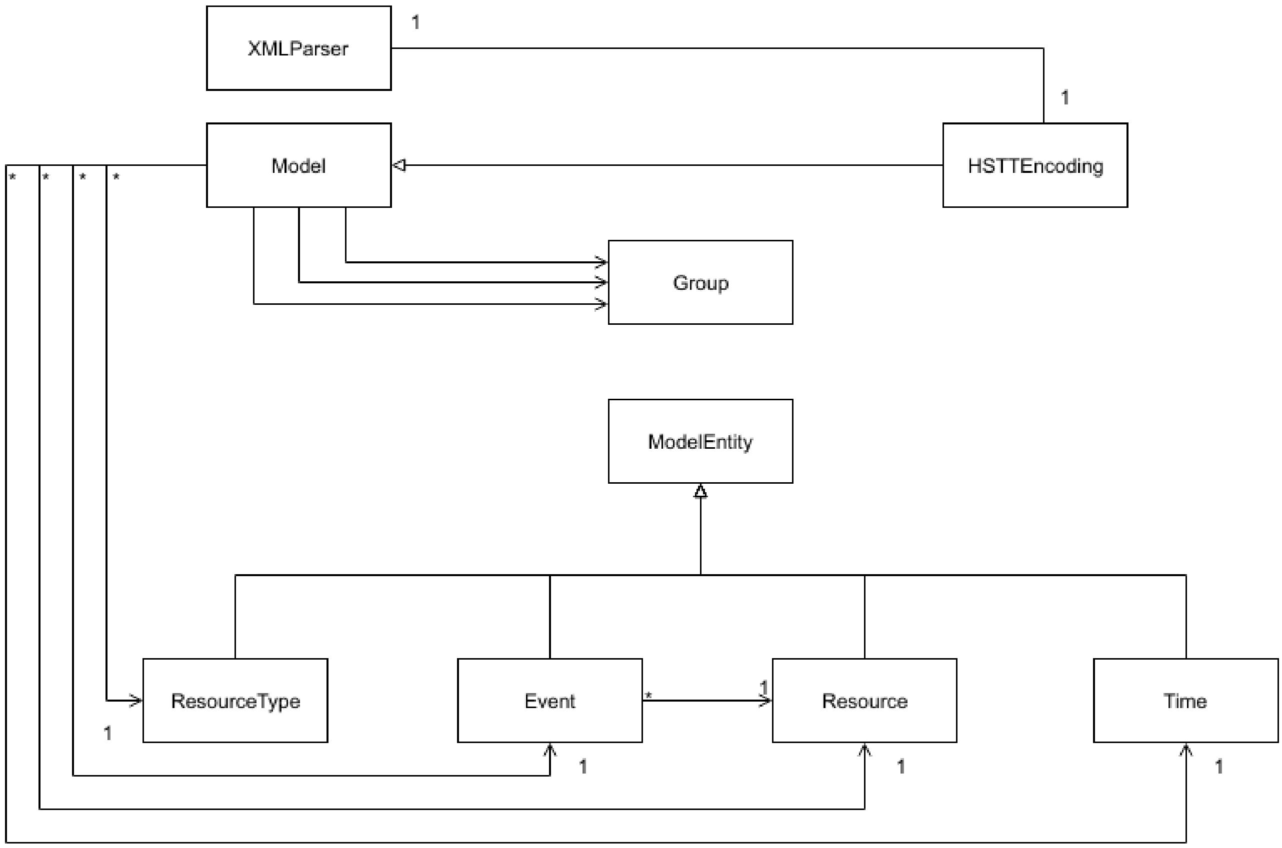

Figura 7.3: Model d'objectes

El programa consistirà en un objecte HSTTEncoding al qual donarem el nom del fitxer que volem llegir i treballarà amb ell. El HSTTEncoding crearà l'objecte XMLParser per tal de llegir el fitxer i es comunicaràn entre ells mentre XMLParser vagi llegint el fitxer enviant la informació llegida al HSTTEncoding. Al heretar de Model, aquest pot guardar totes les dades, que consistiran en diferents tipus de *ModelEntity* i diferents *Groups*. Des de aquí HSTTEncoding s'encarregarà de resoldre la instància comunicant-se amb la API SMT.

8. Implementació i proves

La implementació del generador s'he fet en C++, i està format per un conjunt de blocs definits:

- *Parser* del fitxer en format XHSTT.
  S'encarrega de llegir el fitxer XHSTT i extreure'n la informació de la instància continguda en el fitxer XML en format XHSTT passat per paràmetre. És important que aquest fitxer tingui l'estructura establerta en el format XHSTT.

- Bloc de codificació de la instància a SMT.
  En aquest apartat utilitzem la API SMT per tal de codificar la instància a SMT i passar-la al *solver* corresponent, el Yices 2.

- Bloc de escriptura del resultat a XHSTT.
  En aquest apartat creem un fitxer amb la instància treballada XHSTT i hi afegim la solució trobada al final.

L'usuari només tindrà un binari, anomenat *hstt2smt*, amb el qual interactuarà per resoldre una instància que passarà per fitxer. Aquest s'encarregarà de fer tota la feina i mostrarà per pantalla el resultat, a part també creara un fitxer en la carpeta *output* amb la instància treballada, afegint-hi la solució trobada en el format XHSTT Aquest binari serà el resultat de compilar el generador amb la API SMT i les llibreries que es necessiten per poder treballar amb XML i amb SMT.

8.1. Codificació

En aquesta secció es descriu la implementació de la codificació a SMT del generador d'horaris. Posat que SMT no permet clàusules violables, per a la codificació d'aquesta s'utilitzarà el cost i una variable auxiliar per cada clàusula *soft*, que s'afegira a la seva corresponent clàusula de forma condicional, negant la clàusula inicial, de manera que si no es compleix la clàusula inicial, la variable auxiliar hagi de ser certa. Al acabar la codificació de la instància afegirem una clàusula pseudo-booleana amb totes les variables auxiliars amb els seus pesos, comparada amb un upperbound. Posat que la funció d'optimització serà la de minimitzar el cost total, el optimitzador anirà modificant el upperbound intentant reduir la funció objectiu, que també és el resultat exacte de la pseudo-booleana.

Per exemple, si tenim les *soft-clauses* següents: SC_1, SC_2, SC_3 amb costos respectivament: W_1, W_2,

W_3. Crearíem les variables auxiliars Aux_1, Aux_2 i Aux_3. Amb això crearíem les següents clàusules:

$$\neg SC_1 \rightarrow Aux_1$$
$$\neg SC_2 \rightarrow Aux_2$$
$$\neg SC_3 \rightarrow Aux_3$$

Hi aplicaríem Àlgebra de Bool per convertir a CNF i convertiríem les clàusules que són tipus $\neg A \rightarrow B$ a $\neg(\neg A) \vee B$ i d'aquí eliminaríem la doble negació, quedant $A \vee B$:

$$\neg SC_1 \vee Aux_1$$
$$\neg SC_2 \vee Aux_2$$
$$\neg SC_3 \vee Aux_3$$

Així doncs, afegint la clàusula pseudo-booleana amb les variables auxiliars i els pesos, ens quedarien les següents clàusules:

$$SC_1 \vee Aux_1$$
$$SC_2 \vee Aux_2$$
$$SC_3 \vee Aux_3$$
$$Aux_1 * W_1 + Aux_2 * W_2 + Aux_3 * W_3 <= UPPERBOUND$$

Aquesta codificació està pensada per resoldre instàncies en què tots els recursos estan assignats a algun event, com ara les instàncies de Brazil que són en les que ens hem enfocat al desenvolupar el generador. Aquesta particularitat ens permet simplificar l'espai de cerca degut a la reducció de la combinatòria del problema, i permet una codificació més simple i directe de les restriccions de que es compon.

8.1.1 Model

El model està pensat per treballar a centrant-nos en els Events, els quals podríem definir com una reunió a la que es presenten diferents recursos i la nostre feina és la de decidir en quins espais de temps dels disponibles fiquem cada reunió d'aquestes.

Les variables del model són les següents:

- $Xt_{0,0}...Xt_{|Events|-1,|Times|-1}$
  Per cada event tenim tantes variables com espais de temps tingui la instancià. Cada una d'aquestes variables indica si un event en un espai de temps, s'està celebrant o no. Doncs si per l'event e i l'espai t, $Xt_{e,t}$ és certa, vol dir que en el temps t, e s'està celebrant.

- $Xs_{0,0}...Xs_{|Events|-1,|Times|-1}$
  Per cada event tenim tantes variables com espais de temps tingui la instància. Cada una de aquestes variables indica si un event comença en l'espai de temps representat. Un event comença en un espai de temps si no es donava en l'espai de temps anterior i/o si l'espai de temps és la primera hora del dia. Doncs si per l'event e i l'espai t, $Xt_{e,t}$ és certa, vol dir que en el temps t, e comença i que cap més variable $Xs_{e,i}$ serà certa i que la variable $Xt_{e,t}$ serà certa.

- $Xd_{0,1,0}...Xd_{|Events|-1,event.duration,|Times|-1}$
  Per cada event i cada possible duració d'aquest tenim una variable. Per exemple, si per a un event de duració 5, es defineixen fins a 5 conjunts de $|Times|$ variables. Cada variable ens indica si comença una lliçó de la durada i en l'espai de temps que representa la variable.

8.1.2 Clàusules de Channeling

Per poder dotar les variables del model de semàntica caldrà afegir certes clàusules. A continuació s'enumeren:

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

$$\forall e \in 0...|Events|-1\ \forall i \in 0...|Times|-1$$
$$exactily\_one(\neg Xs_{e,i} \vee \{\forall j \in 1...e.duration\ Xd_{e,j,i}\})$$

- Si un event té lloc a t però no a t-1, és que comença:

$$\forall e \in 0...|Events|-1\ \forall i \in 0...|Times|-1$$
$$\neg Xt_{e,i} \vee Xt_{i-1} \vee Xs_i$$

- Si un event comença amb duració d, llavors té lloc en d hores consecutives:

$$\forall e \in 0...|Events|-1\ \forall din1...e.duration\ \forall i \in 0...|Times|-1\ \forall j \in i...i+d-1$$
$$\neg Xd_{e,d,i} \vee Xt_{e,j}$$

8.1.3 Restriccions XHSTT

Assign Times Constraint

Restricció per imposar que a tots els events se'ls hi assigni temps. Per tant, per a cada event e hem de imposar que del conjunt de variables Xte, i, n'hi hagi exactament *e.duration* que s'evaluin a Cert:

$$\forall e \in Events\ exactly_k(\{Xt_{e,0}...Xt_{e,|Times|-1}\}, e.duration)$$

Split Events Constraint

Aquesta restricció limita la manera en com es fragmenten els events. És a dir, sobre el nombre de sessions en què es fragmenta i la durada d'aquestes.
Per a la restricció sobre el nombre de sessions en què es fragmenta un event, utilitzarem les tags de *MinimumAmount* i *MaximumAmount* per fer les següents clàusules:

$$\forall e \in Events\ at\_most\_k(\{Xs_{e,0}...Xs_{e,|Times|-1}\}, MaximumAmount)$$
$$\forall e \in Events\ at\_least\_k(\{Xs_{e,0}...Xs_{e,|Times|-1}\}, MinimumAmount)$$

Pel que fa el control de la durada de les sessions nomes cal negar totes les variables Xd de cada event que corresponguin a duracions q no estiguin compreses entre el rang definit per les tags *MinimumDuration* i *MaximumDuration*:

$$\forall e \in Events\ \forall d \notin MinimumDuration...MaximumDuration \wedge d \in 1...e.duration$$
$$\forall t \in 0...|Times| - 1 \quad \neg Xd_{e,d,t}$$

Distribute Split Constraint

Restricció que limita el nombre de events d'una duració determinada, per tant limita la cardinalitat de variables Xd. d ve donada per la tag *Duration* i el màxim i mínim venen donats per les tags *Maximum* i *Minimum*:

$$\forall e \in Events\ at\_most\_k(\{Xd_{e,d,0}....Xd_{e,d,|Times|-1}\}, max) \qquad si\ max < \frac{e.duration}{d}$$
$$\forall e \in Events\ at\_most_k(\{Xd_{e,d,0}....Xd_{e,d,|Times|-1}\}, min) \qquad si\ min > 0$$

Prefer Times Constraint

Restricció que indica en quins espais de temps no es poden programar certes sessions. Per exemple, pot servir per indicar que no es pot programar una sessió de dues hores en la última hora del dia.

Definim el conjunt Ta com el conjunt de slots de temps els quals la restricció ens diu que són preferits. Neguem totes les variables Xd de duració d que representin espais de temps no continguts dins de Ta.

$$\forall e \in Events\ \forall t \in Times \wedge t \notin Ta$$
$$(\neg Xd_{e,d,t})$$

Spread Events Constraint

Aquesta restricció posa límits en el nombre de sessions de cada event que es poden celebrar en certs grups d'espais de temps definits com a dies.

Definim Tg com el conjunt grups d'espais de temps definits per la restricció.

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

Avoid Clashes Constraint

Aquesta restricció imposa que certs recursos no poden tenir assignats més d'un event al mateix temps. Aquí aprofitem que els recursos estàn assignats d'un inici, i definim E_r com el conjunt de events als quals ha de assistir un recurs r i imposem que per cada espai de temps, com a molt un d'els events de E pot estar programat:

$$\begin{aligned}&\forall r \in Resources\ \forall t \in Times\\ &\qquad\qquad at\_most\_one(\{Xt_{e,t} | e \leftarrow E_r\})\end{aligned}$$

Avoid Unavailable Times Constraint

Restricció que indica que hi ha certs espais de temps durant les quals no podem utilitzar certs recursos. T és conjunt d'espais de temps no disponibles i E_r el conjunt d'events als que ha d'assistir un recurs r:

$$\forall r \in Resources\ \forall t \in T\ \forall e \in E_r \quad (\neg Xt_{e,t})$$

Limit Idle Times Constraint

Restricció que limita el nombre d'espais de temps lliure entre dos espais de temps ocupats a l'horari de certs recursos. Per tal de codificar aquesta restricció és necessari afegir variables auxiliars al model. Per cada espai de temps que pot ser forat s'introdueix una variable auxiliar que serà certa si l'espai realment és un forat.

El conjunt Tg representa el conjunt de grups d'espais de temps que tractem. S'introdueix la variable $idle_i$ per cada recurs que defineix si aquest té un forat en l'horari en l'espai de temps i. Per tal de considerar un espai de temps com a forat, cal que en algun moment abans i després hi hagi programada alguna activitat. Així que es defineixen dos conjunts: A_i representant els espais de temps posterior (*After*) de i i B_i representant els espais de temps anteriors (*Before*) de i.

Llavors, per a cada recurs r es fan les següents clàusules:

$$
\begin{array}{lr}
\forall g \in Tg\ \forall t \in g\ \forall e \in E_r & \\
\quad (\neg Idle_i \vee \neg Xt_{e,t}) & si\ \ (B_t \neq \emptyset\ \& A_t \neq \emptyset) \\
\forall g \in Tg\ \forall t \in g & \\
\quad (\neg Idle_t \vee \{\forall b \in B_t\ \forall e \in E_r\ \ Xt_{e,b}\}) & si\ \ (B_t \neq \emptyset) \\
\forall g \in Tg\ \forall t \in g & \\
\quad (\neg Idle_t \vee \{\forall a \in A_t\ \forall e \in E_r\ \ Xt_{e,a}\}) & si\ \ (A_t \neq \emptyset) \\
\forall g \in Tg\ \forall t \in g\ \forall b \in B_t\ \forall a \in A_t & \\
\quad \forall e1 \in E_r\ \forall e_2 \in E_r\ \forall e_3 \in E_r & \\
\quad (Xt_{e_1,t} \vee \neg Xt_{e_2,b} \vee Xt_{e_3,a} \vee Idle_t) & si\ \ (B_t \neq \emptyset\ \& A_t \neq \emptyset)
\end{array}
$$

Un cop definides i lligades les variables auxiliars l'únic que queda és, per cada recurs, imposar les restriccions de cardinalitat:

$$
\begin{array}{l}
\forall r \in Resources \\
\quad at\_most\_k(\{Idle_t | t \leftarrow Times\}, max) \\
\quad at\_least\_k(\{Idle_t | t \leftarrow Times\}, min)
\end{array}
$$

Cluster Busy Times Constraint

Restricció que imposa límits sobre el nombre de grups d'espais de temps (que podem entendre com a dies) en què un recurs pot estar ocupat. En aquesta restricció introduirem una variable $Busy_g$ per a cada recurs que ens indicarà si el recurs està ocupat en el grup g o no.

Per a cada recurs r es fan les clàusules següents:

$$
\begin{array}{l}
\forall g \in Tg \\
\quad (\neg Busy_g \vee (\forall e \in E_r\ \forall t \in g \quad Xt_{e,t})) \\
\forall g \in Tg\ \forall t \in g\ \forall e \in E_r \\
\quad (\neg Xt_{e,t} \vee Bsuy_g)
\end{array}
$$

Un cop definides i lligades les variables auxiliars l'únic que queda és, per cada recurs, imposar les restriccions de cardinalitat:

$$
\begin{array}{l}
\forall r \in Resources \\
\quad at\_most\_k(\{Busy_g | g \leftarrow Tg\}, max) \\
\quad at\_least\_k(\{Busy_g | g \leftarrow Tg\}, min)
\end{array}
$$

8.2. Proves

Les proves del programa desenvolupat s'han dividit en dos fases: el parser i el generador automàtic d'horaris sencer. Això s'ha fet així per assegurar que quan es comences a dissenyar el model, les dades rebudes per aquest fossin sempre correctes. Així doncs, abans de començar a fer la codificació en sí, s'ha fet i provat el parser.

Per fer les proves tant del parser com del programa sencer s'ha utilitzat el mètode de la caixa negra[7]. Aquest mètode consisteix en examinar el funcionament del programari que està sent provat sense tenir en compte el com està implementat, com un tot. La idea està en provar els casos d'ús de forma independent al codi intern en sí i comparar els resultats amb els que s'esperaven.

Aquest mètode busca trobar errors en:

- Funcions incorrectes o inexistents
- Errors de interfície
- Errors en les estructures de dades
- Errors en el comportament o en el rendiment
- Errors de inicialització i terminació

9. Resultats

En aquesta secció es realitzarà una comparativa de rendiment amb els diferents encodings diferents en la API SMT i es veuràn els resultats donats per el generador per les diferents instàncies de Brazil que es poden trobar en el repositori públic.

El format XHSTT disposa d'un avaluador d'horaris online anomenat HSeval[8]. Aquest, ens serveix per avaluar la bondat d'un horari i, a més, permet mostrar-lo de forma gràfica. A continuació es mostraran els resultats obtinguts amb la instància *BrazilInstance1.xml* en la web i en el generador en sí, que també mostra els horaris resultants per pantalla, com es pot veure a la figura 9.4.

Per aconseguir l'informe de resultats només cal pujar el fitxer xml amb la solució, seleccionar la opció *HTML Report* i fer click a *Submit*. El resultat es pot veure a la figura 9.1. Per aconseguir la representació gràfica dels horaris, en comptes de seleccionar *HTML Report*, triem la opció *HTML Timetables*. Aquesta opció generarà un horari per cada Recurs (en les instàncies de Brazil serien un per cada professor, com veiem a la figura 9.2, i un per cada classe, com es pot veure en la figura 9.3).

Els resultats mostrats a continuació, són els obtinguts de optimitzar la *BrazilInstance1.xml*.

Solution Group TFG_IsmaelElHabri_2019

This solution group is Solution obtained with a SMT modelling powered by yices. It was contributed by Ismael El Habri on 3182019.

Solution of instance BrazilInstance1_XHSTT-v2014 (cost 0.00088)

Distribute Split Events Constraint	Constraint name	Point of application	Calculation	Inf.	Obj.
DistributeSplit_1	At least 1 double lesson(s)	T8-S1	1 * Linear(1 too few in T8-S1)		1
Total (1 point)					1

Limit Idle Times Constraint	Constraint name	Point of application	Calculation	Inf.	Obj.
noIDLETimesT	No IDLE times for teachers	T1	3 * Linear(1 too ? in Th)		3
noIDLETimesT	No IDLE times for teachers	T2	3 * Linear(4 too ? in Tu, We)		12
noIDLETimesT	No IDLE times for teachers	T3	3 * Linear(1 too ? in Mo)		3
noIDLETimesT	No IDLE times for teachers	T6	3 * Linear(2 too ? in Mo, Th)		6
noIDLETimesT	No IDLE times for teachers	T7	3 * Linear(2 too ? in Fr)		6
noIDLETimesT	No IDLE times for teachers	T8	3 * Linear(1 too ? in Mo)		3
Total (6 points)					33

Cluster Busy Times Constraint	Constraint name	Point of application	Calculation	Inf.	Obj.
MaxNofDaysConstraint_T_days_2	Not more than 2 days with lessons	T2	9 * Linear(gr_Mo + gr_Tu + gr_We + gr_Th - 2)		18
MaxNofDaysConstraint_T_days_2	Not more than 2 days with lessons	T5	9 * Linear(gr_Tu + gr_We + gr_Th + gr_Fr - 2)		18
MaxNofDaysConstraint_T_days_2	Not more than 2 days with lessons	T7	9 * Linear(gr_Mo + gr_We + gr_Th + gr_Fr - 2)		18
Total (3 points)					54

Summary	Inf.	Obj.
Distribute Split Events Constraint (1 point)		1
Limit Idle Times Constraint (6 points)		33
Cluster Busy Times Constraint (3 points)		54
Grand total (10 points)		88

Figura 9.1: Informe d'optimalitat obtingut amb HSEval

T6 (allTeachers)

Mo	Tu	We	Th	Fr
T6-S2	T6-S3		T6-S2	
T6-S2	T6-S3		T6-S2	
	T6-S1			
T6-S3	T6-S1		T6-S1	
T6-S3			T6-S1	

Limit Idle Times Constraint	Constraint name	Point of application	Calculation	Inf.	Obj.
noIDLETimesT	No IDLE times for teachers	T6	3 * Linear(2 too ? in Mo, Th)		6
Total (1 point)					6

Figura 9.2: Horari d'un professor de *BrazilInstance1.xml*

S1 (allClasses)

Mo	Tu	We	Th	Fr
T7-S1	T1-S1	T2-S1	T1-S1	T7-S1
T7-S1	T1-S1	T2-S1	T2-S1	T7-S1
T3-S1	T6-S1	T7-S1	T2-S1	T4-S1
T3-S1	T6-S1	T4-S1	T6-S1	T4-S1
T8-S1	T2-S1	T8-S1	T6-S1	T3-S1

S2 (allClasses)

Mo	Tu	We	Th	Fr
T6-S2	T2-S2	T4-S2	T6-S2	T3-S2
T6-S2	T2-S2	T4-S2	T6-S2	T3-S2
T2-S2	T1-S2	T8-S2	T1-S2	T5-S2
T2-S2	T1-S2	T8-S2	T5-S2	T5-S2
T3-S2	T5-S2	T2-S2	T5-S2	T4-S2

Figura 9.3: Horari d'unes classes de *BrazilInstance1.xml*

```
c stats 40;40;19.3577;-1;220434;970173;-1;-1;-1;-1;-1;-1;-1;-1;
c lb/ub 40 40
v
Class S3
+------------+------------+------------+------------+------------+
|Mo          |Tu          |We          |Th          |Fr          |
+------------+------------+------------+------------+------------+
|Mo_1 - T3-S3|Tu_1 - T6-S3|We_1 - T5-S3|Th_1 - T7-S3|Fr_1 - T4-S3|
+------------+------------+------------+------------+------------+
|Mo_2 - T8-S3|Tu_1 - T6-S3|We_1 - T5-S3|Th_1 - T7-S3|Fr_1 - T4-S3|
+------------+------------+------------+------------+------------+
|Mo_2 - T8-S3|Tu_3 - T5-S3|We_3 - T4-S3|Th_3 - T5-S3|Fr_3 - T3-S3|
+------------+------------+------------+------------+------------+
|Mo_4 - T6-S3|Tu_3 - T5-S3|We_4 - T7-S3|Th_4 - T1-S3|Fr_3 - T3-S3|
+------------+------------+------------+------------+------------+
|Mo_4 - T6-S3|Tu_5 - T1-S3|We_4 - T7-S3|Th_4 - T1-S3|Fr_5 - T7-S3|
+------------+------------+------------+------------+------------+

Class S1
+------------+------------+------------+------------+------------+
|Mo          |Tu          |We          |Th          |Fr          |
+------------+------------+------------+------------+------------+
|Mo_1 - T7-S1|Tu_1 - T1-S1|We_1 - T2-S1|Th_1 - T1-S1|Fr_1 - T7-S1|
+------------+------------+------------+------------+------------+
|Mo_1 - T7-S1|Tu_1 - T1-S1|We_1 - T2-S1|Th_2 - T2-S1|Fr_1 - T7-S1|
+------------+------------+------------+------------+------------+
|Mo_3 - T3-S1|Tu_3 - T6-S1|We_3 - T7-S1|Th_2 - T2-S1|Fr_3 - T4-S1|
+------------+------------+------------+------------+------------+
|Mo_3 - T3-S1|Tu_3 - T6-S1|We_4 - T4-S1|Th_4 - T6-S1|Fr_3 - T4-S1|
+------------+------------+------------+------------+------------+
|Mo_5 - T8-S1|Tu_5 - T2-S1|We_5 - T8-S1|Th_4 - T6-S1|Fr_5 - T3-S1|
+------------+------------+------------+------------+------------+

Class S2
+------------+------------+------------+------------+------------+
|Mo          |Tu          |We          |Th          |Fr          |
+------------+------------+------------+------------+------------+
|Mo_1 - T6-S2|Tu_1 - T2-S2|We_1 - T4-S2|Th_1 - T6-S2|Fr_1 - T3-S2|
+------------+------------+------------+------------+------------+
|Mo_1 - T6-S2|Tu_1 - T2-S2|We_1 - T4-S2|Th_1 - T6-S2|Fr_1 - T3-S2|
+------------+------------+------------+------------+------------+
|Mo_3 - T2-S2|Tu_3 - T1-S2|We_3 - T8-S2|Th_3 - T1-S2|Fr_3 - T5-S2|
+------------+------------+------------+------------+------------+
|Mo_3 - T2-S2|Tu_3 - T1-S2|We_3 - T8-S2|Th_4 - T5-S2|Fr_3 - T5-S2|
+------------+------------+------------+------------+------------+
|Mo_5 - T3-S2|Tu_5 - T5-S2|We_5 - T2-S2|Th_4 - T5-S2|Fr_5 - T4-S2|
+------------+------------+------------+------------+------------+

Violated Soft Clauses (violated/total): (8/37) ----------- Total cost: 40
s OPTIMUM FOUND
o 40
```

Figura 9.4: Resultat de la instància 1 des de el generador d'horaris.

9.1. Comparativa d'encodings

En aquest apartat es provaran tots els encodings de les clàusules de cardinalitat (bàsicament *at_most_one* i *at_most_k*) disponibles en la API SMT i es mirarà com afecten aquests en el rendiment del generador d'horaris. Totes les proves realitzades en aquesta secció seran de resolució. S'ha utilitzat un timeout de 30 minuts.

9.1.1 Comparativa per temps d'execució

Encoding Cardinalitat: Sorter

Aquí es presenta una taula amb els temps d'execució amb els diferents encodings disponibles del *at_most_one* combinats amb el encoding de cardinalitat *sorter*:

Encoding	BrazilInstance1	BrazilInstance2	BrazilInstance3
Quadràtic	1.669	11.257	16.832
Logarítmic	1.625	11.784	19.444
Ladder	1.78	12.281	20.74
Heule	1.609	12.062	16.881

Figura 9.5: Taula de resultats de temps de les instàncies de la 1 a la 3

Encoding	BrazilInstance4	BrazilInstance5	BrazilInstance6
Quadràtic	15:11.96	48.887	01:09.23
Logarítmic	24:49.42	48.092	01:08.44
Ladder	9:42.25	1:22.49	01:19.53
Heule	Timeout	52.955	02:19.89

Figura 9.6: Taula de resultats de temps de les instàncies de la 4 a la 6

Utilitzant una *Sorting network* per a la codificació de les restriccions de cardinalitat, en les instàncies petites, a la figura 9.5, es pot observar que la codificació més lenta és la de Ladder, tot i que les diferencies de temps són molt petites, excepte en la instància 3, on la diferència entre la més ràpida (la codificació quadràtica) i el la més lenta és de pràcticament 4 segons. La resta d'instàncies d'aquesta taula tarden massa poc com per poder observar grans diferencies de temps.

En les instàncies grosses, la figura9.5 es pot observar una mica de tot. En la 4 el millor temps s'aconsegueix amb el encoding ladder, pero aquest aconsegueix mals resultats amb les instancies 5 i 6. En general, els temps aconseguits amb la instància 4 són molt diferents entre els encodings, mentre que en la resta de instàncies en general son més semblants. Es pot observar que el Heule en general dona mals resultats en les instàncies grosses, fent saltar la limitació de temps de 30 minuts en la instància 4 i duplicant el temps d'execució en la instància 6.

Sorpren generalment que el encoding quadràtic obtingui resultats tant competitus, sent el millor o molt proper al millor en totes les instàncies menys la 4, on aconsegueix igualment el segon lloc, pero molt lluny del millor temps.

Encoding Cardinalitat: Totalizer

Aquí es presenta una taula amb els temps d'execució amb els diferents encodings disponibles del *at_most_one* combinats amb el encoding de cardinalitat *totalizer*:

Encoding	BrazilInstance1	BrazilInstance2	BrazilInstance3
Quadràtic	1.604	11.048	19.215
Logarítmic	1.611	11.252	17.846
Ladder	1.728	10.899	16.713
Heule	1.681	12.233	17.377

Figura 9.7: Taula de resultats de temps de les instàncies de la 1 a la 3

Encoding	BrazilInstance4	BrazilInstance5	BrazilInstance6
Quadràtic	09:52.03	49.606	01:08.74
Logarítmic	12:56.72	48.908	01:08.23
Ladder	Timeout	49.556	01:24.78
Heule	Timeout	01:14.19	01:25.67

Figura 9.8: Taula de resultats de temps de les instàncies de la 4 a la 6

Amb la codificació *Totalizer* per a les restriccions de cardinalitat, en les instàncies petites, a la figura 9.7, es pot veure que, a diferència del apartat anterior, la codificació Ladder és la que dóna millors resultats, mentre que abans era la més lenta en les codificacions petites. Però altre cop, aquestes instàncies son petites i es resolen molt ràpidament, així que les diferències acostumen a estar dins d'un marge d'error acceptable.

Pel que fa les instàncies grosses, es torna a observar que s'obtenen molt bons resultats amb l'encoding quadràtic, on és el millor en la instància 4 per molta diferència, i està molt aprop de ser-ho en la resta de instàncies, en l'aspecte que s'està comparant en aquest apartat, el temps d'execució. En la instància 4 es pot observar, a més, que l'encoding Ladder fa saltar la limitació de temps i l'encoding de Heule, d'els que acaben, dona els pitjors resultats amb diferència. Aquest (l'encoding de Heule) també és el que dona pitjors resultats en la resta de instàncies de la taula.

Comparativa sorter vs totalizer

Al comparar els dos encodings de les restriccions de cardinalitat, es pot veure que generalment els temps no canvien gaire. En les que més diferència es nota són les següents:

- **Instància 3:** Ofereix millors resultats utilitzant el sorter amb el *at_most_one* quadràtic o el totalizer amb el *at_most_one* ladder. El quadràtic utilitza 3 segons més amb el totalizer, però generalment és més rapid en aquesta instància.
- **Instància 4:** Generalment és més ràpid el totalizer, pero els millors resultats absoluts en aquesta instància són amb el sorter combinat amb el *at_most_one* ladder, encoding amb el qual salta el timeout utilitzant el totalizer. Els temps obtinguts amb el totalizer i el *at_most_one* quadràtic,

però, son bastant propers (10 segons de diferència en una instància que tarda pràcticament 10 minuts).

Vist això i comparant les taules es pot veure que generalment la combinació d'encodings que ofereix millors temps d'execució és el totalizer amb el *at_most_one* quadràtic, i per tant, les pròximes proves utilitzaràn aquesta combinació d'encodings.

9.1.2 Comparativa de variables i clàusules generades

Comparativa Sorter vs Totalizer Aquestes comparatives es faràn amb l'encoding *at_most_one* quadràtic:

Instància	Sorter	Totalizer
BrazilInstance1	222991	220114
BrazilInstance2	1776829	1769961
BrazilInstance3	2187085	2179450
BrazilInstance4	7112715	7099821
BrazilInstance5	6385494	6372080
BrazilInstance6	8702195	8687131

Instància	Sorter	Totalizer
BrazilInstance1	978757	969534
BrazilInstance2	6113127	6090809
BrazilInstance3	8853332	8828541
BrazilInstance4	24622271	24580215
BrazilInstance5	24752226	24709009
BrazilInstance6	30815832	30766834

Figura 9.9: Taula de resultats amb el nombre de variables i clàusules generades

Com es pot veure en la figura 9.9, l'encoding de les restriccions de cardinalitat genera més variables i més clàusules. Això explicaria el perquè generalment en la comparativa de temps d'execució s'aconseguien més ràpidament.

Comparativa *at_most_one* Aquestes comparatives es faràn amb l'encoding per restriccions de cardinalitat totalizer:

Instància	Quadràtic	Logarítmic	Ladder	Heule
BrazilInstance1	220,114	220,664	220,889	220,564
BrazilInstance2	1,769,961	1,771,386	1,772,611	1,772,461
BrazilInstance3	2,179,450	2,181,100	2,182,300	2,181,950
BrazilInstance4	7,099,821	7,102,496	timeout	7,105,171
BrazilInstance5	6,372,080	6,374,680	6,376,930	6,376,430
BrazilInstance6	8,687,131	8,690,356	8,693,031	8,692,656

Figura 9.10: Taula de resultats amb el nombre de variables generades

Instància	Quadràtic	Logarítmic	Ladder	Heule
BrazilInstance1	969,534	969,984	969,559	969,234
BrazilInstance2	6,090,809	6,090,534	6,087,609	6,087,459
BrazilInstance3	8,828,541	8,829,116	8,826,341	8,825,991
BrazilInstance4	24,580,215	24,578,115	timeout	24,571,315
BrazilInstance5	24,709,009	24,708,334	24,703,034	24,702,534
BrazilInstance6	30,766,834	30,766,834	30,759,459	30,759,084

Figura 9.11: Taula de resultats amb el nombre de clàusules generades

Com es pot veure en la taula corresponent a la figura 9.10 amb l'encoding quadràtic és el que tenim menys clàusules variables sempre, això és degut a que, com hem vist a l'apartat 4.4.4, no introdueix variables. Per altre banda podem veure que, sorprenentment no sempre és el que més clàusules té, intercalant-se aquesta posició amb l'encoding Logarítmic en funció de la instància.

L'encoding Ladder és el que, consitentment, té més variables. A la vegada, sembla ser el segon que genera menys clàusules, sent el primer l'encoding de Heule. Aquest, tot i no introduir tantes variables com d'altres, és consistentment el que té menys clàusules.

9.2. Resultats d'optimització

10. Conclusions

Durant el desenvolupament d'aquest treball s'han estudiat, repassat i aplicat diverses tècniques de programació per restriccions, sobretot les diferents codificacions de restriccions globals a CNF per SAT i SMT, que s'han acabat utilitzant en el generador d'horaris que s'ha creat.

Pel que fa el generador desenvolupat al llarg d'aquest treball, tot i estar molt lluny de la perfecció, ha permés posar a prova les diferents codificacions de les restriccions globals vistes en el treball i compleix amb els objectius marcats al principi del treball. El generador és capaç de resoldre el problema en un temps raonable (la instància que tarda més és la 4, que ens tarda menys de 10 minuts), excepte la instància 7, la qual al intentar resoldre, ens quedem sense memòria. Cal tenir en compte però que les instancies 4 i 6 són problemes de instituts reals de Brazil, i els temps d'execució resultants en aquestes instàncies són molt positius.

Pel que fa la planificació, la implementació del parser va durar bastant més del previst, i des de aquí es va haver de adaptar la planificació en funció d'això com es pot veure en el diagrama de Gantt de la figura

La realització d'aquest treball m'ha ajudat molt a aprofondir i consolidar molts dels coneixaments adquirits al llarg de la carrera, particularment els apartats referents al itinerari que he estudiat, el de computació. També m'ha ajudat a ficar a prova la meva organització i plantejament dels problemes, sobretot els problemes de satisfacció de restriccions, i el com es resolen. També m'ha ajudat a compendre una mica millor el funcionament del món de la recerca.

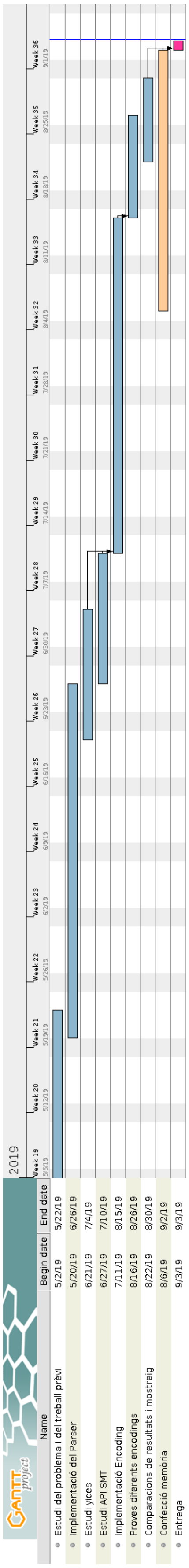

Figura 10.1: Diagrama de Gantt amb el treball realitzat en el projecte

11. Treball futur

En primer lloc seria interessant afegir encodings de les diferents restriccions del problema que no suporta actualment el generador d'horaris, posat que ens hem centrat en un sub-conjunt de les instàncies disponibles, les que tenen els recursos preassignats.

Es podrien explorar els paradigmes de SMT i intentar codificar el problema amb les diferents teories que té, com ara els BitVectors.

També es podrien millorar les codificacions i el model en sí per utilitzar menys memòria i sobretot, per intentar aconseguir millors resultats al optimitzar el problema, comparant amb les diferents solucions disponibles per a cada instància.

12. Bibliografía

[1] Cristòfor Nogueira Gascons. "Generador d'horaris d'instituts". A: (2015). Disponible a: http://hdl.handle.net/10256/11507.

[2] Wikipedia contributors. *Bin packing problem — Wikipedia, The Free Encyclopedia.* 2019. URL: https://en.wikipedia.org/w/index.php?title=Bin_packing_problem&oldid=912207075.

[3] Tim B Cooper i Jeffrey H Kingston. "The Complexity of Timetable Construction Problems Technical Report Number 495". A: (febr. de 1995).

[4] Jeffrey H. Kingston. *High School Timetable Data Format Specification.* 2012. URL: http://www.it.usyd.edu.au/~jeff/cgi-bin/hseval.cgi?op=spec.

[5] Stephen A Cook. "The complexity of theorem-proving procedures". A: *Proceedings of the third annual ACM symposium on Theory of computing.* ACM. 1971.

[6] Olivier Bailleux i Yacine Boufkhad. "Efficient CNF Encoding of Boolean Cardinality Constraints". A: *CP*. 2003.

[7] Wikipedia contributors. *Black-box testing — Wikipedia, The Free Encyclopedia.* [Online; accessed 31-August-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Black-box_testing&oldid=909756532.

[8] Jeffrey H. Kingston. *The HSEval High School Timetable Evaluator.* 2012. URL: http://www.it.usyd.edu.au/~jeff/cgi-bin/hseval.cgi.

[9] Universitat de Twente. *International High School Timetabling competition.* 2018. URL: http://www.utwente.nl/ctit/hstt/.

[10] Aishwarya Agarwal. *Theory of computation — Decidable and undecidable problems — GeeksforGeeks.* URL: https://www.geeksforgeeks.org/theory-computation-decidable-undecidable-problems/.

[11] GeeksforGeeks. *Theory of computation — Decidable and undecidable problems — GeeksforGeeks.* URL: https://www.geeksforgeeks.org/np-completeness-set-1/.

[12] Arseny Kapoulkine. *pugixml.* URL: https://pugixml.org/.

[13] SRI International. *The Yices SMT Solver.* URL: https://yices.csl.sri.com/.

[14] Standard C++ Foundation. *Standard C++.* URL: https://isocpp.org/.

[15] Wikipedia contributors. *C++ — Wikipedia, The Free Encyclopedia.* [Online; accessed 29-August-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=C%2B%2B&oldid=912907129.

[16] The Qt Company. *Qt APIs & Libraries, Tools and IDE.* URL: https://www.qt.io/qt-features-libraries-apis-tools-and-ide/#ide.

[17] The LaTeX project. *LaTeX– A document preparation system.* URL: https://www.latex-project.org/.

[18] *Visual Studio Code.* URL: https://code.visualstudio.com/.

[19] The Linux Foundation. *Linux.* URL: https://www.linuxfoundation.org/projects/linux/.

[20] ArchWiki. *Arch Linux.* URL: https://wiki.archlinux.org/index.php/Arch_Linux.

[21] Zhe Liu. *Algorithms for Constraint Satisfaction Problems (CSPs).* Universitat de Waterloo. Disponible a: http://www.cs.toronto.edu/~fbacchus/Papers/liu.pdf. 1998.

[22] Mauricio Toro et al. "GELISP : A Framework to represent musical constraint satisfaction problems and search strategies". A: 86 (abr. de 2016). Disponible a: http://www.jatit.org/volumes/Vol86No2/17Vol86No2.pdf, pàg. 327 - 331.

[23] Wikipedia contributors. *Boolean satisfiability problem — Wikipedia, The Free Encyclopedia.* 2019. URL: https://en.wikipedia.org/w/index.php?title=Boolean_satisfiability_problem&oldid=911344299.

[24] Wikipedia contributors. *Maximum satisfiability problem — Wikipedia, The Free Encyclopedia.* 2019. URL: https://en.wikipedia.org/w/index.php?title=Maximum_satisfiability_problem&oldid=900651688.

[25] Clark Barrett i Cesare Tinelli. "Satisfiability modulo theories". A: *Handbook of Model Checking.* Springer, 2018, pàg. 305 - 343.

[26] Leonardo De Moura i Nikolaj Bjørner. "Satisfiability modulo theories: introduction and applications". A: *Communications of the ACM* 54.9 (2011), pàg. 69 - 77.

[27] Wikipedia contributors. *Satisfiability modulo theories — Wikipedia, The Free Encyclopedia.* 2019. URL: https://en.wikipedia.org/w/index.php?title=Satisfiability_modulo_theories&oldid=909772056.

[28] Carsten Sinz. "Towards an optimal CNF encoding of boolean cardinality constraints". A: *International conference on principles and practice of constraint programming.* Springer. 2005, pàg. 827 - 831.

[29] Roberto Asín et al. "Cardinality networks and their applications". A: *International Conference on Theory and Applications of Satisfiability Testing.* Springer. 2009, pàg. 167 - 180.

13. Manual d'usuari i instal·lació

13.1. Compilació

Hi ha un *Makefile* per a poder compilar el projecte. Per fer-ho cal tenir instal·lades les llibreries *pugixml*[1] i *yices2*[2]

Per a compilar de forma que s'obtingui el millor rendiment, no cal utilitzar cap paràmetre, però si es vol la versió de depuració, cal ficar el paràmetre *DEBUG:=1*.

- Per a compilar normalment: `make`
- Per a compilar amb capacitat de depurar: `make DEBUG:=1`

Això deixa l'executable en la carpeta ./bin/linux sóta el nom *hstt2smt*

13.2. Ús

El resultat de compilar com s'ha vist anteriorment, és un executable amb els següents paràmetres rellevants:

- fitxer: instància XHSTT que es vol resoldre.
- `-ub, --upper-bound`: per indicar un upper-bound al resoldre
- `--use-assumptions=0`: paràmetre heredat de la API SMT, s'ha de ficar a 0, perquè no s'han implementat assumpcions.
- `-o, --optimizer`: permèt triar l'optimitzador a utilitzar, si es tria *check* no optimitza. Opcions: `check, ub, bu, dico, native`

Per exemple, per a resoldre la primera instància de Brazil, anomanada *BrazilInstance1.xml* que es troba en la carpeta *Instances*, es pot fer (l'ordre dels pràmetres no importa):

```
./bin/linux/hstt2smt instances/BrazilInstance1.xml --use-assumptions=0 -o=check
```

[1] `https://pugixml.org/`
[2] `https://yices.csl.sri.com/`