UNIVERSITAT DE LLEIDA

ESCOLA POLITÈCNICA SUPERIOR

ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

Resolución de los problemas MaxSAT y MinSAT mediante Programación Lineal Entera

Autor: Andreu Belenguer Seuma

Director: Carlos José Ansótegui Gil

Septiembre 2011

Índice general

Índice de cuadros

Listings

Capítulo 1

Introducción

El problema MaxSAT es una generalización del problema de satisfactibilidad booleana. La idea es que en ocasiones no todas las restricciones de un problema se pueden cumplir, por lo que dividimos la instancia en dos grupos: las restricciones o cláusulas que deben ser satisfechas (*hard*), y las que pueden cumplirse o no (*soft*). En este último grupo, podemos asignar distintos pesos a las cláusulas (*weights*), donde el peso es el grado de penalización asociado a incumplir la cláusula, de esta forma podemos indicar la importancia de cada cláusula. Dada una instancia MaxSAT, queremos encontrar una asignación que cumpla todas las cláusulas *hard*, y que a su vez minimice la suma de los pesos de las cláusulas *soft* incumplidas. Dicha asignación de valores se considera el óptimo en este contexto. El problema MinSAT es una variante del problema MaxSAT que busca maximizar la suma de los pesos de las cláusulas *soft* incumplidas.

El problema MaxSAT es un problema natural combinatorio que puede ser utilizado en distintos ámbitos como pueden ser: *combinatorial auctions*, planificación, creación de horarios, *FPGA routing*, instalación de paquetes de software, etc. Sin embargo los resolutores actuales no han tenido tanto éxito como los resolutores SAT del problema de la satisfacibilidad booleana en el campo industrial. El problema MaxSAT es NP-hard, por lo tanto el objetivo es producir resolutores eficientes que puedan tratar problemas reales aunque tengan gran tamaño y permitan resolverlos en un tiempo razonable dada su complejidad.

Un problema de programación lineal entera (LIP) es un problema de optimización sobre una función lineal objetivo, sujeto a unas restricciones en forma de igualdades o desigualdades lineales. La programación lineal entera puede ser aplicada a varios campos de estudio, se utiliza de forma extendida en ámbitos económicos, pero puede ser utilizada en problemas de ingeniería.

En este proyecto presentamos una traducción de los problemas MaxSAT y MinSAT a LIP y comprobamos como se comporta el resolutor CPLEX, un resolutor de estado del arte LIP, comparado con los resolutores de estado del arte

MaxSAT. Para evaluar los resultados se han usado los resultados obtenidos en la MaxSAT Evaluation 2010.

1.1. Objetivos

Los objetivos de este proyecto son los siguientes:

- Diseñar traducciones de los problemas MaxSAT y MinSAT a LIP.
- Implementar un resolutor de problemas MaxSAT y MinSAT que utilice la herramienta de programación lineal entera CPLEX.
- Evaluar el rendimiento de nuestro resolutor sobre los problemas de la MaxSAT Evaluation 2010, comparándolo con los resolutores del estado del arte.

1.2. Estructura del documento

En el capítulo 1 describimos brevemente los problemas estudiados en el proyecto y los objetivos. A continuación en el capítulo 2 se detallan los problemas y presentamos el resolutor de problemas LIP CPLEX. En el capítulo 3 desarrollamos una traducción de problemas MaxSAT y MinSAT a un problema LIP tras lo cual mostramos su implementación en el capítulo 4. En el capítulo 5 comparamos los resultados obtenidos por el resolutor con otros resolutores MaxSAT y mostramos los resultados de MinSAT, finalmente en el capítulo 6 valoramos los resultados obtenidos y buscamos posibles vías de trabajo futuro.

Capítulo 2

Estado del arte

2.1. Problema MaxSAT y MinSAT

El problema Weighted Partial MaxSAT es una generalización del problema de satisfacibilidad booleana. La idea es que en ocasiones no todas las restricciones de un problema se pueden cumplir, por lo que dividimos la instancia en dos grupos: las restricciones o cláusulas que deben ser satisfechas (*hard*), y las que pueden cumplirse o no (*soft*). En este último grupo, podemos asignar distintos pesos a las clausulas (*weights*), donde el peso es el grado de penalización asociado a incumplir la cláusula, de esta forma podemos indicar la importancia de cada cláusula. Para clasificar los tipos de instancia MaxSAT, diremos que una instancia es *weighted* cuando tenga pesos asignados a las cláusulas, y diremos que se trata de una instancia *partial* cuando las cláusulas esten divididas en *hard* y *soft*. Dada una instancia *weighted partial* MaxSAT, queremos encontrar una asignación que cumpla todas las cláusulas *hard*, y que a su vez minimice la suma de los pesos de las cláusulas *soft* incumplidas. Dicha asignación de valores se considera el óptimo en este contexto.

Una estrategia directa a la hora de resolver un problema Weighted Partial MaxSAT es transformarlo a un problema Partial MaxSAT donde cada cláusula *soft* C con peso w sea sustituida por w copias de C, y utilizar un resolutor Partial MaxSAT sobre la instancia generada. Sin embargo, los pesos pueden ser arbitrariamente grandes y por lo tanto el tamaño de la codificación intratable por cualquier resolutor. Por lo tanto esta aproximación solo es factible para instancias con pesos relativamente pequeños. En lugar de eso, en la comunidad MaxSAT hay principalmente dos tipos de resolutores que tratan directamente con pesos: resolutores *depth-first branch and bound* WMaxSatz [11], MiniMaxSat [7], IncWMaxSatz [12], y resolutores basados en la prueba de satisfactibilidad: SAT4J [6], WBO y Msuncore [13], WPM1 [2] y WPM2 [3]. Analizando los resultados de la *Max-*

SAT evaluation [4] podemos concluir que, en general, los resolutores *branch and bound* son más competitivos en los problemas generados aleatoriamente, mientras que los resolutores basados en llamadas a resolutores de satisfactibilidad son mejores para problemas industriales y reales.

Definimos el problema MaxSAT de la siguiente forma. Considerando un conjunto infinito contable de *variables booleanas* $\mathcal{X}$, un literal l es o una variable $x_i \in \mathcal{X}$ o su negación $\neg x_i$. Una *cláusula* C es un conjunto finito de literales, también denotada como $C = l_1 \vee \cdots \vee l_r$, o como $\square$ para la cláusula vacía. Una *fórmula SAT* φ es un conjunto finito de cláusulas, también denotada como $\varphi = C_1 \wedge \cdots \wedge C_m$.

Una *cláusula weighted* (con peso) es una tupla (C, w), donde C es una cláusula y w puede ser tanto un número natural como infinito (indicando la penalización asociada a incumplir la cláusula C). Una cláusula es *hard* si su correspondiente peso es infinito, en cualquier otro caso la cláusula se considera *soft*.

Una *fórmula Weighted Partial MaxSAT* es un multiconjunto de cláusulas *weighted*

$$\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$$

donde las primeras cláusulas m son *soft* y las últimas cláusulas m' son *hard*. Dada una fórumla Weighted Partial MaxSAT φ, definimos las fórmulas SAT $\varphi_{soft} = \{C_1, \ldots, C_m\}$, $\varphi_{hard} = \{C_{m+1}, \ldots, C_{m+m'}\}$ y $\varphi_{plain} = \varphi_{soft} \cup \varphi_{hard}$.

El conjunto de variables que aparecen en una fórmula φ se denota como var(φ).

Una *asignación de verdad* es una función $I : X \to \{0, 1\}$ donde $X \subset \mathcal{X}$. Esta función puede ser extendida a literales, cláusulas, fórmulas SAT y fórmulas MaxSAT de la siguiente manera. Para los liteales, $I(\neg x_i) = 1 - I(x_i)$, si $x_i \in X$; en otro caso, $I(l) = l$. Para cláusulas, $I(l_1 \vee \ldots \vee l_r) = I(l_1) \vee \cdots \vee I(l_r)$, simplificado considerando $1 \vee C = 1$ y $0 \vee C = C$ y $\square = 0$. Para fórmulas SAT $I(C_1 \wedge \ldots \wedge C_r) = I(C_1) \wedge \cdots \wedge I(C_r)$, simplificado considerando $1 \wedge \varphi = \varphi$ y $0 \wedge \varphi = 0$ y $\emptyset = 1$. Para fórmulas MaxSAT, $I(\{C_1, w_1) \vee \ldots \vee (C_m, w_m)\} = \{(I(C_1), w_1), \ldots, (I(C_m), w_m)\}$, simplificado considerando $\{(1, w)\} \cup \varphi = \varphi$ y $\emptyset = 1$. Cabe fijarse que $I(\varphi)$ puede ser 0, 1, o una instanciación parcial de φ.

Decimos que una asignación de verdad I *satisface* un literal, cláusula o fórmula si le asigna 1, y la *falsifica* si le asigna 0. Una fórmula es *satisfactible* si existe una asignación de verdad que la satisface. En caso contrario es *insatisfactible*.

Dada una fórmula insatisfactible φ, un *core insatisfactible* (núcleo insatisfactible) φ_c es un subconjunto de cláusulas $\varphi_c \subseteq \varphi$ que también es insatisfactible. Un *core mínimo insatisfactible* es un *core insatisfactible* tal que cualquier subconjunto propio es satisfactible.

Dada una fórmula weighted partial MaxSAT φ y una asignación de verdad I : var(φ) $\to \{0, 1\}$, el *coste* de la asignación I sobre φ, es la suma de los pesos

de las clausulas falsificadas por I, por ejemplo:

$$\text{cost}(\varphi, I) = \sum_{\substack{(C_i, w_i) \in \varphi \\ I(C_i) = 0}} w_i$$

El *coste óptimo* de una fórmula es el mínimo coste de todas las posibles asignaciones:

$$\text{cost}(\varphi) = \text{mín}\, \text{cost}(\varphi, I) \mid I : \text{var}(\varphi) \to \{0, 1\}$$

Una *asignación óptima* es aquella con un coste óptimo.

Cabe fijarse que cuando w es finito, la tupla (C, w) es equivalente a tener w copias de la cláusula $(C, 1)$ en nuestro multiconjunto.

El *problema Weighted Partial MaxSAT* para una fórmula φ es el problema de encontrar una *asignación óptima*. Si el coste óptimo es infinito entonces el subconjunto de cláusulas hard es insatisfactible, y decimos que la fórmula es *insatisfactible*. El *problema weighted MaxSAT* es un problema *weighted partial MaxSAT* con el conjunto de cláusulas hard vacío. El *problema partial MaxSAT* es un problema weighted partial MaxSAT donde los pesos de todas las cláusulas soft son iguales. El *problema MaxSAT* es el problema Partial MaxSAT cuando no hay clausulas hard. El *problema SAT* sería equivalente a un problema Partial MaxSAT donde no hay clausulas soft.

MaxSAT Evaluation

La MaxSAT evaluation es un evento en el que se comprueba el estado del arte en el campo de los resolutores MaxSAT. El evento se realiza todos los años siendo la evaluación 2011 la 6ª. Los objetivos de la competición son los siguientes:

- Comprobar el rendimiento de los resolutores existentes.
- Identificar problemas difíciles de resolver.
- Identificar técnicas exitosas de resolución.
- Fomentar que los investigadores mejoren sus resolutores y envien nuevos problemas difíciles de resolver.
- Aumentar el conocimiento de cara a nuevas MaxSAT evaluation

De cara a comprobar el rendimiento de nuestro desarrollo lo compararemos con los resolutores presentados a la evaluación del año 2010, que son los siguientes:

- WPM1 : Carlos Ansótegui, Maria Luisa Bonet, Jordi Levy

- PM2 : Carlos Ansótegui, Marıa Luisa Bonet, Jordi Levy
- akmaxsat : Adrian Kuegel
- akmaxsat_ls : Adrian Kuegel
- SAT4J MAXSAT 2.2.0 : Daniel Le Berre
- QMaxSAT : Miyuki Koshimura , Tong Zhang
- IncMaxSatz : Han Lin , Kaile Su, Chu Min Li
- IncWMaxSatz : Han Lin , Kaile Su, Chu Min Li, Josep Argelich
- Maxsat_Power : Abdorrahim Bahrami , Seyed Rasoul Mousavi, Maryam Farshchian
- LS_Power : Abdorrahim Bahrami , Seyed Rasoul Mousavi, Maryam Farshchian
- LS_Power version 1.0
- WMaxsat_Power version 1.0
- LSW_Power : Abdorrahim Bahrami , Seyed Rasoul Mousavi, Maryam Farshchian
- LSW_Power version 1.0
- wbo 1.4a : Vasco Manquinho , Joao Marques-Silva, Jordi Planes
- wbo 1.4b : Vasco Manquinho , Joao-Marques Silva and Jordi Planes

Formato del problema y salida del resolutor

Para poder presentar un problema a la MaxSAT evaluation los problemas deben seguir el siguiente formato:

- Problema MaxSAT en formato DIMACS, podemos ver un ejemplo en el *listing* 2.1
    - El fichero puede empezar con comentarios, estos se marcan empezando la línea con el caracter 'c'.

```
c
c comments Max-SAT
c
p cnf 3 4
1 -2 0
-1 2 -3 0
-3 2 0
1 3 0
```

Listing 2.1: Problema MaxSAT en formato dimacs

```
c
c comments Weighted Max-SAT
c
p wcnf 3 4
10 1 -2 0
3 -1 2 -3 0
8 -3 2 0
5 1 3 0
```

Listing 2.2: Problema Weighted MaxSAT

- A continuación encontramos la línea `p cnf nbvar nbclauses` indicando que la instancia está en formato CNF. `nbvar` indica el número de variables que aparecen en el fichero. `nbclauses` indica el número de cláusulas contenidas en el fichero.
- Por último aparecen las cláusulas, cada línea indica una cláusula en forma de secuencia de números distintos de cero comprendidos entre -nbvar y +nbvar terminando la línea con el caracter 0 como marca de fin de cláusula. Los números positivos indican la correspondiente variable mientras que los negativos indican la negación de la variable.

- Problema Weighted MaxSAT, podemos ver un ejemplo en el *listing* 2.2

  En los problemas Weighted MaxSAT la línea de parámetros tiene la forma `p wcnf nbvar nbclauses` indicando que se trata de un problema con pesos. Los pesos de cada cláusula vienen determinados por el primer entero que aparece en cada línea. El peso de cada cláusula es un entero mayor o igual a 1 y menor que 2^{63}

- Problema Partial MaxSAT, podemos ver un ejemplo en el *listing* 2.3

- En los problemas Partial MaxSAT la línea de parámetros tiene la forma `p wcnf nbvar nbclauses top`. El peso asociado a cada cláusula es el primer

```
c
c comments Partial Max-SAT
c
p wcnf 4 5 15
15 1 -2 4 0
15 -1 -2 3 0
1 -2 -4 0
1 -3 2 0
1 1 3 0
```

Listing 2.3: Problema Partial MaxSAT

```
Example of Weigthed Partial Max-SAT formula:
c
c comments Weigthed Partial Max-SAT
c
p wcnf 4 5 16
16 1 -2 4 0
16 -1 -2 3 0
8 -2 -4 0
4 -3 2 0
3 1 3 0
```

Listing 2.4: Problema Weighted Partial MaxSAT

número que aparece en la línea. Los pesos deben ser mayor o igual que uno y menor que 2^{63}. Las cláusulas hard tienen como peso el indicado en la línea de parámetros como `top` y las cláusulas soft tienen peso 1. Para que el formato sea correcto debemos asegurarnos que el peso `top` sea siempre mayor que la suma de los pesos de todas las cláusulas soft.

- Problema Weighted Partial MaxSAT, podemos ver un ejemplo en el *listing* 2.4

  En los problemas Weighted Partial MaxSAT, la línea de parámetros tiene la forma `p wcnf nbvar nbclauses top`. Los pesos deben ser mayores o igual que 1 y menores que 2^{63}. Las cláusulas hard tienen como peso `top` y las cláusulas soft tienen peso menor que `top`. Para que el formato sea correcto se debe asegurar que el peso `top` sea siempre mayor que la suma de los pesos de las cláusulas soft.

Los resolutores deben sacar mensajes por la salida estándar que permitan comprobar los resultados. El formato de salida está inspirado en el formato DIMACS. Los mensajes de salida deben mostrarse siempre por la salida estándar o la salida de error sin usar ficheros de salida. Solo la salida estándar se usa para comprobar

```
c -----------------
c My Max-SAT Solver
c -----------------
o 10
o 7
o 6
o 5
s OPTIMUM FOUND
v -1 2 3 -4 -5 6 -7 8 9 10 -11 -12 13 -14 -15
```

Listing 2.5: Salida de un resolutor para un problema MaxSAT

```
c --------------------------
c My Weighted Max-SAT Solver
c --------------------------
o 481
o 245
o 146
o 145
o 144
o 143
s OPTIMUM FOUND
v -1 2 3 -4 -5 6 -7 8 9 10 -11 -12 13 -14 -15 16 -17 18 19 20
```

Listing 2.6: Salida de un resolutor para un problema Weighted MaxSAT

los resultados pero también se guarda un registro de la salida de error. Podemos ver un ejemplo de salida correcta de un resolutor en los listings 2.5 y 2.6

Los mensajes que puede escribir el resolutor por la salida estándar son los siguientes.

- Comentario (línea 'c ')
  - Estas líneas empiezan por el carácter 'c' seguido de un espacio.
  - Son líneas opcionales que puede aparecen en cualquier orden en la salida del resolutor.
  - Contienen información adicional que el autor del resolutor quiere remarcar.
  - Se aconseja evitar las líneas de comentarios que pueden ser útiles en un entorno interactivo pero no aportan en una ejecución masiva de pruebas.
- Solución óptima actual (línea 'o ')

- Estas líneas empiezan por el caracter 'o' seguido por un espacio.
- A continuación aparece un entero que representa la mejor solución encontrada por el momento, por ejemplo el mínumo número de cláusulas insatisfechas por la solución actual en un problema MaxSAT o el mínimo de la suma de los pesos de las cláusulas insatisfechas para un problema Weighted MaxSAT.
- Son líneas obligatorias y los resolutores deben escribirlas en la salida en cuanto encuentren una nueva solución.
- El entorno de pruebas tomará como solución óptima la última línea 'o ' que haya sacado el resolutor por el flujo de salida.

- Solución (línea 's ')
  - Esta línea empieza por el caracter 's' seguido de un espacio.
  - Solo se permite una línea de este tipo en la salida y es obligatoria.
  - Esta línea indica la solución del resolutor, debe tener uno de los siguientes posibles valores:

    s OPTIMUM FOUND Esta línea se debe escribir en la salida cuando la última línea 'o ' es la solución óptima al problema al dar una asignación completa de las variables de la fórmula.

    s UNSATISFIABLE Esta línea se debe escribir en la salida cuando se comprueba que el conjunto de cláusulas hard es insatisfactible.

    s UNKNOWN Esta línea se debe mostrar en cualquier otro caso, por ejemplo cuando no hemos encontrado la solución óptima.
  - Si no aparece ninguna línea de solución se asume una solución de tipo UNKNOWN.

- Valores (línea 'v ')
  - Estas líneas empiezan por el carácter 'v' seguido de un espacio.
  - Se permite más de una línea 'v' pero en ese caso se considerará el resultado de juntarlas.
  - Si el resolutor encuentra una solución óptima (escribe 's OPTIMUM FOUND') debe mostrar también una asignación de verdad para las variables de la instancia que será usada para comprobar si la solución es correcta. Es decir debe proporcionar un listado de literales no complementarios que al ser interpretados a cierto nieguen el mínimo número de cláusulas soft en un problema MaxSAT o minimicen la suma de los pesos de las cláusulas insatisfechas para un problema Weighted MaxSAT.

- Un literal se denota como un entero que identifica la variable y la negación de una variable se denota como el símbolo menos seguido del entero que representa la variable.
- La línea de solución debe definir el valor de todas las variables del problema. El órden de los literales no importe.
- Si el resolutor no escribe una línea de asignación de valores o la línea tiene formato incorrecto se considerará una solución de tipo 'UNKNOWN'.

El problema MinSAT es un problema de optimización similar al problema MaxSAT [5]. Partiendo de una instancia MaxSAT cualquiera φ, podemos definir un problema MinSAT φ' cambiando el peso de las cláusulas hard de forma que en lugar de ser ∞ pase a ser $-\infty$ y cambiar su función objetivo *coste óptimo* de forma que el *coste óptimo* de una fórmula MinSAT sea el máximo coste de todas las posibles asignaciones, es decir:

$$\begin{aligned} \varphi'_{hard} &= \{(C_1, -\infty), \ldots, (C_n, -\infty)\} \\ \text{cost}(\varphi') &= \text{máx}\, \text{cost}(\varphi', I) \end{aligned}$$

Si $\text{cost}(\varphi') = -\infty$ decimos que la formula MinSAT φ' es insatisfactible.

2.2. Problema LIP

Un *problema de programación lineal* se define como el problema de *maximizar o minimizar una función lineal sujeta a restricciones lineales*. Las restricciones pueden ser tanto igualdades como desigualdades. Podemos escribir un problema de programación lineal como:

$$\text{mín}\, c_1x_1 + c_2x_2 + c_3x_3 + \cdots + c_nx_n$$

sujeto a:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x3 + \cdots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x3 + \cdots + a_{2n}x_n &\leq b_2 \\ \vdots &\quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x3 + \cdots + a_{mn}x_n &\leq b_m \end{aligned}$$

$$\begin{aligned} 0 \leq x_j &\leq x_{l,j} \\ b_i &\geq 0 \end{aligned}$$

Pudiendose escribir en notación matemática reducida como:

$$\text{mín} \sum_{j=1}^{n} c_j x_j$$

sujeto a:

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j &\leq b_i && \text{(i = 1, 2, . . . , m)} \\ x_j &\geq 0 && \text{(j = 1,2, . . . , n)} \\ b &\geq 0 \end{aligned}$$

Cabe notar las siguientes características del problema:

- La función objetivo es lineal
- Las restricciones son lineales

Y las siguientes convenciones:

- Las variables se definen como positivas
- Todos los elementos b son positivos

Un *problema lineal de programación entera* o Linear Integer Programing (LIP) es un caso particular de problema de programación lineal en que las variables de decisión deben tomar valor entero para que la solución sea válida.

$$x_j \in \mathbb{Z} \qquad \text{(j = 1,2 . . . , n)}$$

Otra restricción posible es hacer que las variables tomen valor booleano para simular un comportamiento de decisión.

$$\begin{aligned} x_j &\in \mathbb{Z}_2 && \text{(j = 1,2 . . . , n)} \\ Z_2 &= \{0, 1\} \end{aligned}$$

2.3. IBM ILOG CPLEX Optimization Studio

IBM ILOG CPLEX Optimization Studio (normalmente llamado simplemente CPLEX) es un paquete de software de optimización.

El IBM ILOG CPLEX Optimizer resuelve problemas grandes [14] de programación entera y programación lineal usando los siguientes métodos [1]:

- Variante *primal* o *dual* del método *simplex*
- Variante *primal* o *dual* del método *barrier interior point*

También resuelve problemas convexos y no-convexos de programación cuadrática y problemas con restricciones cuadráticas convexas.

2.3.1. Comandos CPLEX

En este apartado describimos los comandos CPLEX necesarios para introducir los problemas MaxSAT una vez transformados a problemas de programación lineal entera con restricción de valor booleano para las variables.

Para este fin vamos a analizar un programa simple en CPLEX mostrado en el *listing* 2.7 que contiene todos los comandos necesarios para introducir cualquier problema que podamos generar.

En la sentencia de la línea 1 asignamos el valor 0 al parámetro `mip tolerances mipgap`. Este parámetro indica el margen de precisión permitida para que una solución se acepte como válida. En nuestro caso queremos obtener el coste óptimo de la fórmula MaxSAT original de forma que todos nuestros programas tendrán este parámetro con valor 0, es decir forzamos que las soluciones sean exactas.

En la línea 2 indicamos que vamos a empezar a introducir el problema, en la línea siguiente ponemos un nombre arbitrario al problema. El problema termina en la línea 27 con la sentencia `End`. Como se ha explicado en la sección 2.2 un problema de programación lineal entera consta de una función lineal objetivo así como ciertas restricciones lineales en forma de igualdades o desigualdades. Para introducir esta función primero debemos indicar cual es el objetivo a buscar con los comandos `Maximize` o `Minimize` tal como se ve en la línea 4. A continuación introducimos la función con un nombre cualquiera `[nombre]:` seguido de un sumatorio de variables con sus respectivos coeficientes, los nombres de las variables también son arbitrarios. Podemos ver un ejemplo en la línea 5. Para introducir las restricciones lineales, indicamos con el comando `Subject To` que se van a empezar a introducir, seguido de tantas restricciones como sea necesario. Para introducir cada restricción igual que se ha hecho para la función objetivo, necesitamos dar un nombre arbitrario a cada una, seguido de la desigualdad lineal que corresponda. Podemos ver el conjunto de restricciones del ejemplo de la línea 7

```
set mip tolerances mipgap 0
enter
problem_test
Maximize
obj:+ 5 b4 + 8 b5 + 7 b6 + 6 b7 + 5 b8 + 6 b9
Subject To
c1: + x2 >= 1
c2: - x1 >= 0
c3: + x3 >= 1
c4: + b4 + x1 >= 1
c5: + b4 - x2 >= 0
c6: + b4 - x3 >= 0
c7: - b4 - x1 + x2 + x3 >= -1
c8: + b5 - x1 >= 0
c9: - b5 + x1 >= 0
c10: + b6 + x1 >= 1
c11: - b6 - x1 >= -1
c12: + b7 - x2 >= 0
c13: - b7 + x2 >= 0
c14: + b8 + x1 >= 1
c15: - b8 - x1 >= -1
c16: + b9 - x3 >= 0
c17: - b9 + x3 >= 0

Binaries
 x1 x2 x3 b4 b5 b6 b7 b8 b9
End
Optimize
DISPLAY solution variables -
```

Listing 2.7: Ejemplo simple de programa CPLEX

a la línea 23. Para restringir las variables a valores booleanos tenemos el comando `Binaries` en la línea 25 seguido de las variables que queramos restringir. En nuestro caso todas las variables del problema serán siempre booleanas.

Una vez introducido el problema lazamos la optimización con el comando `Optimize` mostrado en la línea 28. Para recuperar los resultados ejecutamos el comando `DISPLAY solution variables -` que muestra el valor que toman todas las variables para la solución encontrada.

Tuning del CPLEX

Además de resolver el problema CPLEX nos permite lanzar un tuning de parámetros para encontrar la mejor configuración para resolver el problema. En lugar de utilizar el comando `Optimize` podemos lanzar un tuning del problema utilizando el comando `Tune`.

2.3.2. Salida CPLEX

En este apartado describimos brevemente como obtener el resultado al problema a partir de la salida de un programa CPLEX. Para este fin utilizaremos como ejemplo la salida mostrada en el *listing* 2.8 correspondiente a ejecutar el programa del *listing* 2.7.

Nos fijaremos en las líneas 25 - 33 que se generan al ejecutar el comando `DISPLAY solution variables -` como se indica en el apartado 2.3.1. En estas líneas se muestran todas las variables que toman valor 1 y a continuación el mensaje `All other variables in the range 1-9 are 0.` de forma que todas las variables mostradas tendrán valor 1 y el resto de variables del programa toman valor 0.

```
1
2  IBM ILOG CPLEX Optimization Studio Academic Research Edition
3  Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer
       12.2.0.0
4    with Simplex, Mixed Integer & Barrier Optimizers
5  5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55
6  Copyright IBM Corp. 1988, 2010.  All Rights Reserved.
7
8  Type 'help' for a list of available commands.
9  Type 'help' followed by a command name for more
10 information on commands.
11
12 CPLEX> New value for mixed integer optimality gap tolerance: 0
13 CPLEX> Enter name for problem: Enter new problem ['end' on a
       separate line terminates]:
14 CPLEX> Tried aggregator 1 time.
15 MIP Presolve eliminated 17 rows and 9 columns.
16 All rows and columns eliminated.
17 Presolve time =    0.00 sec.
18
19 Solution pool: 1 solution saved.
20
21 MIP - Integer optimal solution:  Objective =  2.9000000000e+01
22 Solution time =    0.00 sec.  Iterations = 0  Nodes = 0
23
24 CPLEX> Incumbent solution
25 Variable Name           Solution Value
26 b4                            1.000000
27 b6                            1.000000
28 b7                            1.000000
29 b8                            1.000000
30 b9                            1.000000
31 x2                            1.000000
32 x3                            1.000000
33 All other variables in the range 1-9 are 0.
34 CPLEX>
```

Listing 2.8: Salida CPLEX ejemplo

Capítulo 3

Traducción a LIP

En este capitulo se describe el algoritmo utilizado para transformar cualquier problema MaxSAT o MinSAT a un problema de programación lineal entera restringido a variables de tipo booleano.

3.1. Reificación de cláusulas soft

Para realizar la transformación de un problema MaxSAT a un problema de programación lineal entera con restricción de variables booleanas el primer paso será transformar el problema MaxSAT a otro problema MaxSAT que cumpla la restricción de que todas las cláusulas *soft* deben ser unitarias, es decir tener un solo literal. Además este nuevo literal debe ser una nueva variable que no pertenezca al conjunto de variables del problema original. Llamaremos a los nuevos literales unitarios b_i. Para que la transformación sea equivalente tenemos que asegurarnos que b_i se cumple si y solo si se cumpla la cláusula original y añadir esta restricción en forma de cláusulas hard.

Partiendo de un problema MaxSAT:

$$\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$$

Sustituimos las cláusulas soft a nuevas variables unitarias reemplazando cada cláusula soft (C_i, w_i) por (b_i, w_i).

$$\varphi_k = \{b_1, \ldots, b_m, c_{m+1}, \ldots, C_{m+m'}\}$$

Añadimos a las cláusulas hard la conversión a CNF de la condición $(b_i \leftrightarrow C_i)$.

$$\varphi_k = \{(b_1, w_1), \ldots, (b_m, w_m), C_{m+1}, \ldots, C_{m+m'}, CNF(b_1 \leftrightarrow C_1), \ldots, CNF(b_m \leftrightarrow C_m)\}$$

Como veremos a continuación, podemos limitar las condiciones añadidas dependiendo si se trata de un problema MaxSAT o MinSAT. En el caso de un problema MaxSAT es suficiente con que se cumpla la condición $b_i \rightarrow C_i$.

$$\varphi_k = \{(b_1, w_1), \ldots, (b_m, w_m), C_{m+1}, \ldots, C_{m+m'}, CNF(b_1 \rightarrow C_1), \ldots, CNF(b_m \rightarrow C_m)\}$$

En el caso MinSAT es suficiente con que se cumpla la condición $C_i \rightarrow b_i$

$$\varphi_k = \{(b_1, w_1), \ldots, (b_m, w_m), C_{m+1}, \ldots, C_{m+m'}, CNF(C_1 \rightarrow b_1), \ldots, CNF(C_m \rightarrow b_m)\}$$

Llamaremos a esta traducción alternativa traducción *weak*.

3.2. Función objetivo y restricciones lineales

Finalmente pasaremos el problema transformado a un problema de programación lineal entera con variables restringidas a valores booleanos. Para que los problemas sean equivalente codificaremos las cláusulas soft como la función objetivo del nuevo problema, de forma que los pesos de cada clausula pasaran a ser los coeficientes del sumatorio de la función objetivo y las variables unitarias b_i serán las variables del sumatorio:

$$\varphi_{soft} = \{(b_1, w_1), \ldots, (b_n, w_n)\} \Rightarrow \text{máx} \sum_{i=1}^{n} w_1 b_1 + \cdots + w_n b_n$$

En caso de tener un problema MinSAT, en lugar de maximizar buscaremos minimizar:

$$\varphi_{soft} = \{(b_1, w_1), \ldots, (b_n, w_n)\} \Rightarrow \text{mín} \sum_{i=1}^{n} w_1 b_1 + \cdots + w_n b_n$$

Cada cláusula hard de pasará a ser una restricción del nuevo problema. Como las cláusulas tienen forma de disyunción de literales tendremos que aplicarles la siguiente transformación:

$$C_i = \{l_1, \ldots, l_r\} \Rightarrow l_1 + \ldots l_r >= 1 - |C_{i_{neg}}|$$

Sabiendo que:

$$C_{i_{neg}} = \{l_i \in C_i \mid l_i = \neg x_i, x_i \in X\}$$

Podemos ver que la variante *weak* de la traducción MaxSAT es válida ya que al buscar maximizar la función objetivo sólo con asegurar que si una cláusula b_i no se satisface la cláusula original tampoco se satisface es suficiente para calcular el óptimo.

En el caso MinSAT como buscamos minimizar la función objetivo, tenemos que asegurar que si b_i se satisface, la cláusula original también se satisface.

Capítulo 4

Implementación en CPLEX

El transformación anterior ha sido implementada de forma que recoja el problema Weighted Partial MaxSAT en el formato de la MaxSAT Evaluation (apartado 2.1), transforme la fórmula Weighted Partial MaxSAT φ a otra equivalente φ' con cláusulas soft unitarias codificando el formato original en forma de cláusulas hard tal como se ha explicado en la sección 3.1. Una ver hecho esto genera un problema de programación lineal entera con variables restringidas a valor booleano (sección 2.2) en fomato CPLEX (apartado 2.3.1), ejecuta CPLEX para resolver el problema e interpreta la salida de CPLEX (apartado 2.3.2) para encontrar el óptimo.

Esta transformación del problema se ha codificado en el lenguaje de programación Python (versión $\geq$ 2.3) en forma de una clase *pwmaxsat* que modela el problema y permite transformarlo a un problema de programación lineal entera en formato CPLEX, así como ejecutar CPLEX y recoger los resultados en el formato de la MaxSAT evaluation. También se ha desarrollado un resolutor MaxSAT de línea de comandos *test_exec_cplex.py* que permite lanzar de forma sencilla la resolución de un problema utilizando la clase *pwmaxsat*.

4.1. Instalación de CPLEX

IBM ofrece una versión completa gratuita de CPLEX mediante el programa IBM's Academic Initiative program [9] para enseñanza e investigación. Para esto es necesario registrarse como usuario en la página web.

Una vez registrados como usuarios podemos descargarnos el software CPLEX. Accediendo por la página de descargas [8] del IBM's Academic Initiative program. Una vez en la página usando el enlace "Download from the Software Catalog"podemos buscar el software CPLEX y descargarlo. Una vez descargado el ejecutable de instalación se nos pedirá aceptar la licencia y la ruta completa donde

queremos instalar CPLEX.

Además del software es necesario descargarse una licencia para su utilización, podemos realizar este paso desde la página de obtención de licencias de IBM [10] usando el enlace 'ILOG Optimization Key Request'. Antes de utilizar CPLEX tendremos que inicializar la variable de entorno *ILOG_LICENSE_FILE* para que contenga la ruta a la licencia que nos hemos descargado. Podemos inicializar esta variable en los ficheros */.bashrc* o */.bash_profile* en sistemas Unix si utilizamos la shell bash.

Una vez realizada la instalación podemos ejecutar CPLEX vía los siguientes ejecutables dependiendo de la arquitectura y sistema operativo que tengamos (sistema de 32 o 64 bits).

- *RUTA_DE_INSTALACION*/cplex/bin/x86_sles10_4.1/cplex
- *RUTA_DE_INSTALACION*/cplex/bin/x86-64_sles10_4.1/cplex

4.2. Clase pwmaxsat

Para representar el problema Partial Weighted MaxSAT se ha creado una clase Python llamada *pwmaxsat* y almacenada en el fichero *transat.py*. Esta clase nos permite leer y almacenar el problema así como transformarlo a un problema de programación lineal entera en formato CPLEX.

4.2.1. Variables miembro

Las clausulas de la formula Partial Weighted MaxSAT se han almacenado usando la siguientes estructuras de datos:

L_soft_clauses es el conjunto de cláusulas soft. Usamos una lista python para almacenar todas las clausulas soft. Modelamos cada cláusula soft como una lista de 2 elementos siendo el primero el peso de la cláusula y el segundo una lista de literales.

Modelamos los literales de las clausulas del problema como números enteros, sabiendo que si el número está en positivo nos estamos refiriendo a la variable, y negativo a la negación de la variable.

```
L_soft_clauses =
[
    [2, [1, -2] ],
    [5, [2, 3] ]
]
```

Listing 4.1: Ejemplo de la variable miembro L_soft_clauses

```
L_hard_clauses =
[
    [4, -3],
    [2, -3]
]
```

Listing 4.2: Ejemplo de la variable miembro L_hard_clauses

Por ejemplo las siguientes cláusulas soft:

$$\begin{aligned} \varphi_{soft} &= \{(C_1, w_1), (C_2, w_2)\} \\ C_1 &= x_1 \vee \neg x_2 \\ w_1 &= 2 \\ C_2 &= x_2 \vee x_3 \\ w_2 &= 5 \end{aligned}$$

Pueden ser modeladas como se ha indicado en el *listing* 4.1.

L_hard_clauses es el conjunto de cláusulas hard. No es necesario almacenar el peso ya que el peso de una cláusula hard es siempre infinito. Almacenamos en una lista python para cada cláusula una lista con los literales que la conforman.

Por ejemplo las siguientes cláusulas hard:

$$\begin{aligned} \varphi_{hard} &= \{(C_3, \infty), (C_4, \infty)\} \\ C_3 &= x_4 \vee \neg x_3 \\ C_4 &= x_2 \vee \neg x_3 \end{aligned}$$

Pueden ser modeladas como se ha indicado en el *listing* 4.2.

Además de las cláusulas, la clase *transat* almacena cierta información adicional sobre el problema:

n_vars Tipo entero. Almacena el número de variables del problema.

n_clauses Tipo entero. Almacena el número de cláusulas del problema.

n_top Tipo entero. Almacena la suma de los pesos de todas las cláusulas soft. Sirve para generar un fichero de salida con el problema en formato dimacs.

filename Tipo string. Almacena el nombre del fichero del que hemos leído el problema.

first_uni_clause Tipo entero. Indica el número de variable a partir del cual empezamos a generar nuevas variables unitarias sobre el problema original.

is_uni Tipo booleano. Indica si el problema ha sido transformado a uno equivalente con cláusulas soft unitarias. Podemos controlar si el problema está transformado antes de lanzar CPLEX.

También tenemos ciertas variables miembro que modifican el comportamiento del solver de la clase:

ty_cplex_exec Tipo string. Valores posibles 'P', 'F'. Indica si la comunicación con CPLEX se realiza mediante pipes unix (opción 'P') o usando ficheros temporales (opción 'F'). Por defecto se utiliza la opción 'F'.

nodel_tmp Tipo booleano. Indica si queremos conservar los ficheros temporales generados durante la ejecución de CPLEX. Permite ver el fichero generado que hemos pasado a CPLEX así como la salida de CPLEX para detectar posibles errores. Por defecto falso.

tune Tipo booleano. Indica si queremos realizar un tuning del cplex sobre el problema en lugar de resolver el problema. Por defecto falso.

minsat Tipo booleano. Indica si se trata de un problema MinSAT en lugar de MaxSAT. Por defecto falso.

left Tipo booleano. Indica si queremos aplicar la transformación de la fórmula original usando una implicación de un solo sentido. $\leftarrow$ en lugar de doble sentido $\leftrightarrow$. Por defecto falso.

tuning Tipo lista de strings. Almacena un posible listado de parámetros de tuning para cplex. Por defecto vacío.

4.2.2. Métodos

Los principales métodos de la clase son:

```
comment_line = re.compile(r'^c.*|^\s*$')
parameter_line_pwms = re.compile(r'^p\s+' +
                            r'wcnf\s+' +
                            r'(\d+)\s+' +
                            r'(\d+)\s+' +
                            r'(\d+)\s*$'
                            )
parameter_line_wms = re.compile(r'^p\s+' +
                            r'wcnf\s+' +
                            r'(\d+)\s+' +
                            r'(\d+)\s*$'
                            )
parameter_line_uwms = re.compile(r'^p\s+' +
                            r'cnf\s+' +
                            r'(\d+)\s+' +
                            r'(\d+)\s*$'
                            )
clause_line = re.compile(r'^(\d+)\s+' +
                         r'(.+$)'
                         )
clause_line_uwms = re.compile(r'(^.+$)')
```

Listing 4.3: Expresiones regulares para leer el formato Dimacs

Método constructor *__init__*

Recibe por parámetro un nombre de fichero y opcionalmente los valores de las variables de configuración del solver indicados anteriormente. Lee el problema mediante el método *read_pwmaxsat* y almacena los parámetros en las variables correspondientes. Si existe el fichero *tuning.conf* lee los parámetros de tuning a pasar a CPLEX.

Método *read_pwmaxsat*

Recibe por parámetro una ruta a un fichero en formato dimacs, retorna los valores de las variables n_vars, n_clauses, n_top, L_soft_vars y L_hard_vars.

Este método permite leer problemas Partial Weighted MaxSAT, Weighted MaxSAT, Partial MaxSAT o Unweighted MaxSAT en formato dimacs usando el módulo estándar de Python *re* para construir expresiones regulares. Podemos ver las expresiones regulares usadas en el *listing* 4.3.

Usamos la expresión regular *comment_line* para saltarnos las posibles líneas de comentario o vacías al principio del fichero Dimacs. A continuación leemos los parámetros del problema MaxSAT en cualquiera de los distintos formatos que se han descrito en el apartado 2.1:

parameter_line_pwms Lee parámetros Partial Weighted MaxSAT con forma `p wcnf VARS CLAUSES TOP` por ejemplo `p wcnf 3 4 90` siendo 3 el número de variables del problema, 4 el número de cláusulas y 90 el peso que se asigna a las cláusulas hard.

parameter_line_wms Lee parámetros Weighted MaxSAT con forma `p wcnf VARS CLAUSES` por ejemplo `p wcnf 3 4` siendo 3 el número de variables del problema, 4 el número de cláusulas.

parameter_line_uwms Lee parámetros Unweighted MaxSAT con forma `p cnf VARS CLAUSES` por ejemplo `p cnf 3 4` siendo 3 el número de variables del problema, 4 el número de cláusulas.

Si no encontramos una línea de parámetros con alguno de los formatos descritos se lanza un error de sintáxis en forma de excepción.

Una vez sabemos qué tipo de problema estamos leyendo empezamos a leer las cláusulas del problema, en caso de que el problema sea Partial Weighted MaxSAT o Weighted MaxSAT usaremos la expresión regular *clause_line*, en caso de que sea un problema Unweighted MaxSAT sin pesos usaremos la expresión regular *clause_line_uwms*.

clause_line Lee líneas de cláusula con un entero indicando su peso.

clause_line_uwms Lee líneas de cláusula sin peso.

Leemos las variables como una serie de números enteros separados por espacio con el número 0 marcando el final de la cláusula. Para los problemas Unweighted MaxSAT leemos todas las cláusulas como cláusulas soft con peso 1. Para los problemas Weighted MaxSAT leemos todas las cláusulas como cláusulas soft con su peso indicado. En el caso de los problemas Partial Weighted MaxSAT, tenemos que comparar el peso de las cláusulas con el valor `TOP` leído como parámetro del problema. En caso que el peso sea igual al `TOP` añadiremos la cláusula como hard, en otro caso la leeremos como una cláusula soft con peso igual que en el caso anterior.

Una vez realizados estos pasos retornamos el problema leído como una tupla con `(VARS, CLAUSES, TOP, L\_soft\_clauses, L\_hard\_clauses)`.

Método *to_uni_pwmaxsat*

Sin parámetros de entrada. Si ejecutamos este método sobre un objeto de la clase pwmaxsat en el que se ha leido un problema MaxSAT, este método transforma el problema a uno equivalente con cláusulas soft unitarias. Podemos ver el código en el *listing* 4.4.

```
# Change all soft clauses to unitary vars
# encode format as hard clauses

self.first_uni_var = self.n_vars + 1
uni_var = self.first_uni_var

for soft_clause in self.L_soft_clauses:
    # -b1 || x1 || x2 || x3
    cond1 = []
    cond1.append( -uni_var )

    for var in soft_clause[1]:
        cond1.append( var )

        # b1 || -xN
        if not self.left:
            condx = []
            condx.append( uni_var )
            condx.append( -var )
            self.L_hard_clauses.append(condx)
            self.n_clauses += 1

    # -b1 || x1 || x2 || x3
    self.L_hard_clauses.append(cond1)
    self.n_clauses += 1

    # Modify the original soft clause to unitary
    soft_clause[1] = [uni_var]

    # Next uni var
    uni_var += 1
    self.n_vars += 1

# Mark it's done
self.is_uni = True
```

Listing 4.4: Algoritmo de transformación a clausulas soft unitarias

Podemos ver cómo por cada cláusula soft de la lista L_soft_clauses, recorremos todos sus literales y añadimos las condiciones que hemos definido en el apartado 3.1 como cláusulas hard en la lista L_hard_clauses. Una vez añadidas las condiciones sustituimos la cláusula soft original por una nueva variable unitaria que no aparecía en el problema original. Guardamos en la variable *first_uni_var* de la clase la primera variable unitaria para poder saber que cualquier variable mayor o igual a esta será una variable unitaria generada. Esto es importante para tener localizadas qué variables pertenecen al problema original y cuales no.

Método *get_char_var_abs*

Este método auxiliar se utiliza para generar los nombres de las variables que le pasamos a CPLEX a partir de la representación interna que utilizamos. Nuestras variables internamente se representan con un número entero y CPLEX requiere que las variables contengan letras para poder diferenciarlas de valores numéricos. También hemos diferenciado entre variables del problema original y variables generadas para transformar el problema en un problema con cláusulas soft unitarias. Para este fin tenemos el miembro *first_uni_var* que nos marca que todas las variables iguales o mayores a este valor son generadas y no forman parte del problema original.

El método recibe un número entero N y retorna un string de caracteres formado por la concatenación del caracter x con el número N en valor absoluto, xN si la variable es del problema original (es decir, menor que el miembro *first_uni_var* y retorna un string formado por la concatenación del caracter b con el número N en valor absoluto, bN si la variable es una variable unitaria generada.

Método *create_cplex*

Usamos este método para generar un problema en formato CPLEX a partir del problema MaxSAT pasado a cláusulas unitarias. El formato de salida de CPLEX es el indicado en el apartado 2.3.1. Para generar el fichero de salida como string vamos cada línea en una lista Python `cplex_file` para hacer más sencillo el tratamiento.

Para introducir las cláusulas soft en forma de función a minimizar o maximizar recorremos la lista L_soft_clauses y ponemos cada variable unitaria multiplicada su peso asociado como los sumandos de la función a maximizar o minimizar. Usamos el método *get_char_var_abs* para convertir las variables a un formato que entienda CPLEX. En el *listing* 4.5 podemos ver como generamos cada sumando usando el peso `soft_clause[0]` y la representación de la variable unitaria `self.get_char_var_abs(soft_clause[1][0])`.

```
for soft_clause in self.L_soft_clauses:
    cplex_file.append( '%s %i %s' % (
                    '+',
                    soft_clause[0],
                    self.get_char_var_abs(soft_clause[1][0])
                    ) )
```

Listing 4.5: Transformación de clausulas soft a función objetivo cplex

```
num = 1
for hard_clause in self.L_hard_clauses:
    exp = []
    total = 1
    for var in hard_clause:
        if var > 0:
            exp.append( '+ %s' % self.get_char_var_abs(var) )
        else:
            total -= 1
            exp.append( '- %s' % self.get_char_var_abs(var))

    cplex_file.append( 'c%i: ' % num )
    cplex_file.append( '%s >= %i' % (
                        '\n'.join(exp),
                        total ) )

    num += 1
```

Listing 4.6: Transformación de cláusulas hard a restricciones cplex

Para introducir las cláusulas hard que son las restricciones a las que está sujeta la función objetivo, por cada una de ellas añadiremos una restricción tal como se ha explicado en el apartado 3.2 en el formato indicado en el apartado 2.3.1. Para este fin inicializamos la variable `total = 1`, recorremos cada variable de la cláusula y en caso que sea positiva generamos un sumando `+ var`, en caso que sea negativa generamos un sumando `- var` y decrementamos el valor de la variable `total` en 1. Una vez terminamos este proceso para todas las variables generamos la restricción como el sumatorio de los sumandos generados menor o igual al valor calculado de la variable `total`. Podemos ver este proceso en el *listing* 4.6, donde cabe fijarse que usamos una variable `num` incremental para dar un nombre distinto a cada restricción.

Una vez tenemos todo el fichero de salida en la variable `cplex_file` en forma de lista de líneas lo retornamos todo en un string separando las líneas con el caracter salto de línea y usando el método `join` de Python.

```
# Find unused filename
while True:
    inst_name = os.path.basename(self.filename)
    rand_num = random.randint(0,999999999999)
    self.tmp_input_file = '/tmp/transat_%s_%i.in' % (inst_name,
        rand_num)
    self.tmp_output_file = '/tmp/transat_%s_%i.out' % (inst_name
        , rand_num)
    self.tmp_errput_file = '/tmp/transat_%s_%i.err' % (inst_name
        , rand_num)
    if os.path.isfile(self.tmp_input_file) or\
       os.path.isfile(self.tmp_output_file) or\
       os.path.isfile(self.tmp_errput_file):
        # Continue search
        pass
    else:
        # Found
        break
```

Listing 4.7: Generar nombres de ficheros temporales únicos

Método *exec_cplex*

El método *exec_cplex* se encarga de generar un problema CPLEX llamando al método *create_cplex* y ejecutar CPLEX para que lo resuelva. Una vez se ha terminado de ejecutar CPLEX recupera los datos y llama al método *parse_cplex_output* para interpretar los resultados y mostrarlos por la salida estándar en el formato de la MaxSAT Evaluation tal como se indica en el apartado 2.1.

Para realizar esta comunicación utilizamos ficheros temporales donde escribimos el problema de entrada para CPLEX y recogemos los resultados de salida mediante redirecciones UNIX. Para poder realizar esta comunicación sin que haya problemas en ejecuciones en paralelo del programa debemos asegurarnos que los nombres de fichero usados son únicos y no existian. Para asegurarnos usamos el método `random.randint` para generar un entero aleatorio y el método `os.path.isfile` para asegurarnos que el fichero no existia anteriormente y repetimos el proceso en caso que ya existiera. Podemos ver este proceso en el *listing* 4.7.

Una vez generados los nombres de fichero escribimos el contenido del problema en el fichero de entrada y como podemos ver en el *listing* 4.8 llamamos a CPLEX y redireccionamos la salida a los otros dos ficheros correspondientes.

Método *parse_cplex_output*

En este método leemos la salida de CPLEX en el formato indicado en el apartado 2.3.2, interpretamos la salida y transformamos la solución al formato de sa-

```
cplex_call = 'cat␣%s␣|␣%s␣1>␣%s␣2>␣%s'
    % ( self.tmp_input_file,
        self.cplexbin,
        self.tmp_output_file,
        self.tmp_errput_file)

os.system( cplex_call )
```

Listing 4.8: Llamada a CPLEX utilizando redirección de ficheros

```
        var_line    = re.compile(r'%s(.+)\s+1.000000\s*$' %
            VAR_NEW )
        ori_line    = re.compile(r'%s(.+)\s+1.000000\s*$' %
            VAR_ORI )
        unsat_line  = re.compile(r'^\s*MIP␣-␣Integer␣infeasible
            .\s*$')
        cplex_error = re.compile(r'^\s*CPLEX␣Error.*:(.*)$')
        is_sol_line = re.compile(r'^\s*Solution␣pool:\s+([0-9]+)
            \s+solution[s]*␣saved.\s*$')
```

Listing 4.9: Expresiones regulares para leer la salida de CPLEX

lida de la MaxSAT Evaluation (apartado 2.1).

Para leer la salida usamos las expresiones regulares del módulo estándar `re`. En el *listing* 4.9 podemos ver las expresiones regulares usadas.

var_line Lee las líneas con las variables unitarias generadas que toman valor 1 en CPLEX.

ori_line Lee las líneas con las variables originales del problema que toman valor 1 en CPLEX.

unsat_line Línea que indica que el problema es insatisfactible.

cplex_error Línea de error al ejecutar CPLEX.

is_sol_line Línea que indica que se ha encontrado solución al problema.

Usamos dos estructuras de datos de tipo Set,

- `var_set` para almacenar las variables unitarias generadas que toman valor positivo en CPLEX.
- `ori_var_set` para almacenar las variables originales del problema que toman valor positivo en CPLEX.

```
# Calculate optimum value
value = 0
for soft_clause in self.L_soft_clauses:
    if soft_clause[1][0] not in var_set:
        value += soft_clause[0]
```

Listing 4.10: Cálculo del óptimo a partir de los resultados de CPLEX

```
# Show original variable values
var_values = []
for var in range(1, self.first_uni_var):
    if var in ori_var_set:
        var_values.append( str(var) )
    else:
        var_values.append( str(-var) )
```

Listing 4.11: Generar asignación de verdad a partir de los resultados de CPLEX

Tal como se indica en el apartado 2.1 hay tres posibles resultados, en nuestro caso corresponden a los siguientes casos:

s UNSATISFIABLE Cuando la expresión regular `unsat_line` encuantra que el problema es insatisfactible.

s UNKNOWN Cuando se encuentra algún error o no se encuentra la línea de solución encontrada.

s OPTIMUM FOUND Si la expresión `is_sol_line` encuentra que hay resultados y la expresión regular `unsat_line` no encuentra que el problema sea insatisfactible.

En caso que hayamos encontrado solución óptima realizaremos el cálculo mostrado en el *listing* 4.10. En este cálculo usamos el Set `var_set` para comprobar si las variables unitarias de las cláusulas soft aparecen con valor 1 en la solución. Si no aparecen, hay que sumar el coste por incumplir la cláusula. De este sumatorio obtenemos el coste óptimo.

Para que la solución sea válida hay que proporcionar también una asignación de verdad para todas las variables del problema original. Como sabemos hasta qué número las variables son del problema original al tener almacenada la variable `first_uni_var` en la clase podemos generar la asignación de verdad para todas estas variables mirando si aparecen en el Set `ori_var_set` para saber si toman valor normal o deben aparecen negadas en la solución. Podemos ver este proceso en el *listing* 4.11.

```
$ ./test_exec_cplex.py --help
Usage: test_exec_cplex.py [options] WCNF_FILE

Options:
  -h, --help       show this help message and exit
  -n, --nodel_tmp  Keep temporary files
  -t, --tune       Tune CPLEX parameters
  -m, --minsat     Solve minsat problem
  -l, --left       Encode translation as C1 <- b1 instead of C1
     <-> b1
  -u TIME, --timelimit=TIME
                        Total execution time in seconds for
                           tuning
  -c TIME, --conf_timelimit=TIME
                        Execution time in seconds for tuning for
                            one
                        configuration
```

Listing 4.12: Ayuda para ejecución del resolutor

Una vez realizados estos cálculos escribimos la solución al problema por la salida estándar.

4.3. Ejecución del resolutor por línea de comandos

Para poder ejecutar de forma sencilla el resolutor se ha creado un script en Python `./test_exec_cplex` que utiliza la clase `pwmaxsat` y ofrece ciertos parámetros de línea de comandos para controlar su ejecución. Podemos ver la ayuda en el *listing* 4.12.

Al ejecutar el script debe indicarse la ruta al fichero de problema MaxSAT que queremos resolver. Además están disponibles las siguientes opciones:

-h, –help Muestra la ayuda.

-n, –nodel_tmp No eliminar los archivos temporales al terminar la ejecución. Permite ver el fichero de entrada pasado a CPLEX y la salida de CPLEX.

-t, –tune Ejecuta el tuning de parámetros de CPLEX en lugar de resolver el problema.

-m, –minsat Resuelve el problema MinSAT en lugar de MaxSAT.

-l, –left Aplica la transformación alternativa del problema explicada en el apartado 3.1.

-u TIME, –timelimit=TIME Configura el tiempo total de ejecución del tuning. Se indica en segundos.

-c TIME, –conf_timelimit=TIME Configura el tiempo por configuración del tuning. Se indica en segundos.

Capítulo 5

Resultados experimentales

En este apartado mostraremos los resultados experimentales de ejecutar nuestro resolutor para una serie de problemas MaxSAT de la MaxSAT Evaluation 2010. Para contextualizar los resultados los compararemos con los resultados obtenidos por varios resolutores MaxSAT para esos mismos problemas. Hemos ejecutado dichos problemas también para MinSAT para comprobar como se comporta aunque no tengamos resultados para compararlo. La máquina donde se han ejecutado las pruebas tiene como sistema operativo Rocks Cluster 4.0.0 Linux 2.6.9, y sus especificaciones son procesador AMD Opteron 248 Processor, 2 GHz, memoria ram 512 MB y cache 1024 KB.

Los problemas se han clasificado por categoría así como por su origen, *crafted* para problemas que han sido codificados manualmente como prueba de rendimiento o *industrial* para problemas reales industriales codificados como problema MaxSAT. Estos a su vez vienen clasificados en varias familias de problemas según su origen o características.

Para realizar los experimentos se ha dado un tiempo de 2 horas por problema a cada resolutor y los resultados obtenidos son el tiempo de resolución y si el resolutor ha sido capaz de encontrar una solución al problema. Esta información se muestra en formato de tabla con las dos primeras columnas indicando la familia de problemas y el número de problemas que contiene la familia y el resto de columnas mostrando los resultados de cada resolutor en forma **Media de tiempo(nº de problemas resueltos)**.

5.1. Resultados MaxSAT

Instance set	#	Maxsat_Power	akmaxsat_ls	LS_Power	IncMaxSatz	akmaxsat	WMaxSatz-2009	WMaxSatz+	CPLEX	CPLEX-weak	PM2	WPM1	wbo-1.4a	SAT4JMAXSAT2.2.0
MAXCUTDIMACS_MOD	62	205.49(53)	**187.40(53)**	210.16(53)	211.80(53)	59.22(52)	178.14(52)	161.56(52)	379.04(24)	1197.08(22)	206.93(10)	0.02(4)	0.01(4)	0.57(2)
MAXCUTSPINGLASS	5	7.03(3)	12.00(3)	12.60(3)	29.06(3)	13.82(3)	**5.34(3)**	5.87(3)	56.15(3)	97.28(3)	3.69(2)	0.31(2)	15.29(2)	47.58(1)
bipartitemaxcut-140-630-0.7	50	**170.74(50)**	198.95(50)	208.75(50)	431.73(50)	255.92(50)	521.04(50)	589.62(50)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
bipartitemaxcut-140-630-0.8	50	**124.94(50)**	145.37(50)	151.99(50)	311.33(50)	197.19(50)	402.70(50)	449.02(50)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
Total	167	**156**	156	156	156	155	155	155	27	25	12	6	6	3

Cuadro 5.1: MaxSAT Crafted. Media de tiempo en segundos.

Instance set	#	wbo-1.4a	PM2	WPM1	SAT4JMAXSAT2.2.0	CPLEX-weak	CPLEX	IncMaxSatz	LS_Power	Maxsat_Power	WMaxSatz+	WMaxSatz-2009	akmaxsat	akmaxsat_ls
CircuitDebuggingProblems	9	139.35(8)	**36.13(8)**	22.43(5)	10.20(2)	2841.87(2)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
SeanSafarpour	68	**224.45(40)**	267.56(39)	241.15(38)	2728.36(10)	238.44(4)	270.84(1)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
Total	77	**48**	47	43	12	6	1	0	0	0	0	0	0	0

Cuadro 5.2: MaxSAT Industrial. Media de tiempo en segundos.

Instance set	#	CPLEX-weak	CPLEX
JobShop	4	0.00(0)	0.00(0)
MAXCLIQUERANDOM	96	**111.16(96)**	112.44(96)
MAXCLIQUESTRUCTURED	62	**1041.71(39)**	940.62(38)
MAXONE3SAT	80	**32.88(80)**	**32.88(80)**
MAXONESTRUCTURED	60	**705.63(57)**	689.28(56)
PSEUDOmiplib	4	**62.13(4)**	62.23(4)
frb	25	**724.09(10)**	724.34(10)
min-enckbtree	54	**291.22(54)**	374.65(54)
Total	385	**340**	338

Cuadro 5.3: Partial MaxSAT Crafted. Media de tiempo en segundos.

Instance set	#	PM2	QMaxSat	wbo-1.4b-wcnf	SAT4J	CPLEX-weak	CPLEX	IncWMaxSatz	WPM1	wbo-1.4a-wcnf	WMaxSatz+	WMaxSatz-2009	akmaxsat	akmaxsat_ls	WMaxsat_Power
CircuitTraceCompaction	4	740.97(4)	**158.53(4)**	451.57(2)	380.86(3)	0.00(0)	0.00(0)	0.00(0)	2873.46(1)	0.00(0)	3744.31(1)	3461.81(1)	0.00(0)	0.00(0)	0.00(0)
HaplotypeAssembly	6	34.56(5)	0.00(0)	195.52(3)	0.00(0)	1178.08(2)	1176.94(2)	0.00(0)	33.81(5)	**10.32(5)**	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
PROTEIN_INS	12	77.55(2)	312.62(3)	0.15(1)	1554.94(4)	1.92(1)	2.21(1)	0.22(1)	51.18(1)	10.09(1)	0.24(1)	0.25(1)	**3804.45(5)**	2992.39(4)	0.26(1)
bcp-fir	59	19.90(58)	48.22(29)	227.52(39)	8.27(10)	**110.83(59)**	111.29(59)	796.16(43)	16.10(54)	132.41(37)	576.99(8)	568.46(8)	6.42(8)	6.37(8)	0.02(4)
bcp-hipp-yRa1SU	38	319.30(30)	269.27(30)	**170.65(37)**	569.75(10)	0.00(0)	0.00(0)	0.00(0)	29.85(11)	100.67(10)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
bcp-hipp-yRa1simp	17	87.52(15)	192.23(16)	**88.90(16)**	835.15(14)	10.51(5)	10.55(5)	265.71(6)	20.96(9)	37.87(10)	16.27(5)	16.30(5)	171.50(5)	176.61(5)	0.04(2)
bcp-msp	64	2290.53(21)	107.02(25)	63.37(11)	98.72(12)	**666.26(32)**	668.66(32)	340.36(24)	19.06(5)	47.55(3)	677.47(11)	702.98(11)	357.28(20)	290.03(20)	0.05(1)
bcp-mtg	40	5.50(40)	**0.26(40)**	117.34(30)	206.95(27)	1013.30(24)	838.50(23)	408.21(14)	76.82(11)	9.73(14)	3064.84(4)	2866.11(4)	63.72(3)	74.72(3)	0.00(0)
bcp-syn	74	11.18(38)	422.57(35)	70.38(24)	570.03(25)	**36.15(71)**	36.18(71)	342.19(34)	3.06(30)	40.88(31)	320.49(23)	318.86(23)	60.15(31)	48.15(22)	0.11(11)
pbo-mqcnencdr	84	1159.05(64)	217.85(76)	217.47(54)	**556.23(84)**	2694.85(5)	2714.15(5)	3464.71(18)	441.28(7)	175.64(14)	2895.55(43)	2781.06(42)	0.00(0)	0.00(0)	0.00(0)
pbo-mqcnlogencdr	84	1170.94(84)	98.58(76)	126.86(66)	**96.02(84)**	1876.41(6)	1874.64(6)	2818.31(47)	279.63(30)	138.31(25)	943.84(64)	951.64(64)	0.00(0)	0.00(0)	0.00(0)
pbo-routing	15	1.27(15)	40.85(15)	95.49(15)	421.50(15)	305.44(15)	130.38(15)	22.96(5)	0.95(15)	**0.60(15)**	5.67(5)	5.68(5)	4800.95(4)	4811.88(4)	0.00(0)
Total	497	**376**	349	298	288	220	219	192	179	165	165	164	76	66	19

Cuadro 5.4: Partial MaxSAT Industrial. Media de tiempo.

Instance set	#	akmaxsat_ls	akmaxsat	IncWMaxSatz	WMaxSatz+	WMaxSatz-2009	WMaxsat_Power	LSW_Power	wpm2-me2011-v1.sh	CPLEX	CPLEX-weak	SAT4J	wbo1.4a-wcnf	WPM1-924-v1
RAMSEY	48	**12.04(37)**	15.98(36)	7.55(36)	18.10(36)	18.17(36)	19.13(36)	20.24(36)	98.74(36)	0.00(0)	0.00(0)	17.62(36)	0.06(34)	22.01(35)
WMAXCUTDIMACS_MOD	62	**80.77(60)**	89.48(60)	234.07(59)	304.81(59)	308.88(59)	324.20(59)	328.46(59)	0.22(3)	1412.04(31)	1076.18(29)	915.18(3)	0.01(4)	0.12(4)
WMAXCUTSPINGLASS	5	**13.07(4)**	19.27(4)	34.61(4)	51.13(4)	51.16(4)	53.33(4)	468.43(3)	0.00(0)	228.77(4)	318.77(4)	11.88(1)	0.00(0)	0.00(0)
frb	34	419.09(14)	377.24(14)	134.00(14)	21.20(9)	21.21(9)	21.84(9)	21.70(9)	224.16(16)	616.09(19)	**885.69(20)**	1731.50(8)	140.56(4)	0.00(0)
Total	149	**115**	114	113	108	108	108	107	55	54	53	48	42	39

Cuadro 5.5: Weighted MaxSAT Crafted. Media de tiempo en segundos.

Instance set	#	CPLEX	CPLEX-weak	IncWMaxSatz	akmaxsat_ls	akmaxsat	WMaxSatz-2009	WMaxSatz+	SAT4JMAXSAT2.2.0	WPM1-924-v1	wbo-1.4b-wcnf	wbo1.4a-wcnf	wpm2-me2011-v1.sh
AUCTIONSAUC_PATHS	88	**0.24(88)**	**0.24(88)**	8.21(88)	22.47(88)	22.90(88)	627.67(81)	627.71(81)	977.27(46)	22.14(33)	91.09(7)	0.00(0)	0.00(0)
AUCTIONSAUC_SCHEDULING	84	**0.81(84)**	**0.81(84)**	218.96(84)	506.39(83)	566.18(83)	88.29(84)	88.02(84)	723.19(77)	7.74(80)	260.46(51)	0.00(0)	3197.31(8)
PSEUDOmiplib	12	180.74(3)	305.98(3)	**1426.58(5)**	3.63(2)	2.20(2)	245.28(3)	244.12(3)	590.52(4)	33.90(3)	39.19(3)	0.00(0)	398.21(2)
WCSPSPOT5DIR	21	**202.44(18)**	**202.44(18)**	1131.04(5)	1.22(4)	0.56(4)	11.07(2)	11.05(2)	7.60(3)	1.40(5)	98.82(6)	1.00(5)	172.49(9)
WCSPSPOT5LOG	21	534.06(8)	3.02(6)	0.63(4)	813.28(5)	772.15(5)	15.10(2)	18.76(2)	18.89(3)	42.99(6)	128.87(6)	10.45(6)	**551.47(9)**
min-encplanning	56	556.99(52)	500.51(52)	94.91(38)	101.49(39)	95.85(39)	213.98(50)	214.92(50)	**3.57(56)**	13.10(54)	10.81(56)	8.88(22)	163.01(39)
min-encwarehouses	18	1.02(18)	**0.93(18)**	1248.43(18)	0.18(1)	0.16(1)	0.31(1)	0.30(1)	1.47(1)	1628.11(3)	136.39(4)	0.00(0)	159.24(1)
random-net	78	**1029.10(59)**	606.65(53)	1173.93(1)	725.57(19)	743.29(19)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	203.97(7)	22.36(55)	0.00(0)
Total	378	**330**	322	243	241	241	223	223	190	184	140	88	68

Cuadro 5.6: Weighted Partial MaxSAT Industrial. Media de tiempo en segundos.

Instance set	#	wpm2-me2011-v1.sh	WPM1-924-v1	wbo1.4a-wcnf	CPLEX-weak	CPLEX	IncWMaxSatz	SAT4JMAXSAT2.2.0	akmaxsat	akmaxsat_ls	WMaxSatz+	WMaxSatz-2009	WMaxsat_Power
timetabling	32	**376.17(15)**	310.56(14)	251.55(8)	0.00(0)	0.00(0)	0.00(0)	283.44(4)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
upgradeability-problem	100	311.25(100)	37.72(100)	152.94(100)	**9.86(100)**	11.34(100)	192.27(90)	696.07(29)	1747.83(21)	1964.98(21)	0.00(0)	0.00(0)	0.00(0)
Total	132	**115**	114	108	100	100	90	33	21	21	0	0	0

Cuadro 5.7: Weighted Partial MaxSAT Industrial. Media de tiempo en segundos.

Los resultados han sido muy positivos, con la implementación en CPLEX por delante del resto de resolutores en los problemas Partial MaxSAT industrial así como en otras familias de problemas concretas dentro de otras categorías de problemas. Vemos que la variante *weak* de la transformación no varía los resultados significativamente aunque en general es más rápida que la transformación normal. Podemos decir que los resolutores branch and bound se comportan mejor en problemas Crafted y los resolutores de test de satisfactibilidad se comportan mejor en problemas Industrial.

MaxSAT

Podemos ver en las tabla de resultados MaxSAT Crafted 5.1 que nuestro resolutor se comporta mejor para estos problemas mejor que los resolutores de test de satisfactibilidad, pero queda por debajo de los resolutores branch and bound. En la tabla MaxSAT Industrial 5.2 como hemos comentado se comportan mejor los resolutores de test de satisfactibilidad y nuestro resolutor queda por debajo de estos y por encima de los branch and bound.

Partial MaxSAT

Podemos ver resultados similares en la tabla Partial MaxSAT Industrial 5.4 en cuanto a posicionamiento de nuestro resolutor, con la diferencia que para unas pocas familias concretas de problemas nuestro resolutor es el más rápido.

Weighted MaxSAT

En la tabla de resultados Weighted MaxSAT Crafted el resolutor se comporta mejor que los de test de satisfactibilidad pero queda por debajo de los branch and bound. Para una familia de problemas (2010instanceswms_craftedfrb) nuestro resolutor resuelve un mayor número de problemas.

Weighted Partial MaxSAT

En la tabla Weighted Partial MaxSAT Crafted 5.6 nuestro resolutor se comporta mejor que el resto de resolutores. Mientras que en los problemas Weighted Partial MaxSAT Industrial (tabla 5.7) aunque en una de las dos familias de problemas estudiadas no consigue resolver ningún problema (timetabling) en la otra familia de problemas (upgradeability problem) funciona más rápido que el resto de resolutores.

5.2. Resultados MinSAT

Instance set	#	MinSAT	MaxSAT
MAXCUTDIMACS_MOD	62	**0.75(62)**	379.04(24)
MAXCUTSPINGLASS	5	**32.68(3)**	56.15(3)
bipartitemaxcut-140-630-0.7	50	**0.76(50)**	0.00(0)
bipartitemaxcut-140-630-0.8	50	**0.77(50)**	0.00(0)
Total	167	**165**	27

Cuadro 5.8: MinSAT Crafted. Media de tiempo en segundos.

Instance set	#	MinSAT	MaxSAT
CircuitDebuggingProblems	9	404.50(1)	0.00(0)
SeanSafarpour	68	3642.69(2)	270.84(1)
Total	77	3	1

Cuadro 5.9: MinSAT Industrial. Media de tiempo en segundos.

Instance set	#	MinSAT	MaxSAT
JobShop	4	**635.51(4)**	0.00(0)
MAXCLIQUERANDOM	96	**0.46(96)**	112.44(96)
MAXCLIQUESTRUCTURED	62	**3.22(62)**	940.62(38)
MAXONE3SAT	80	36.27(80)	**32.88(80)**
MAXONESTRUCTURED	60	787.63(57)	689.28(56)
PSEUDOmiplib	4	**0.15(4)**	62.23(4)
frb	25	**1.42(25)**	724.34(10)
min-enckbtree	54	751.41(53)	374.65(54)
Total	385	**381**	338

Cuadro 5.10: Partial MinSAT Crafted. Media de tiempo en segundos.

En el caso de los problemas MinSAT no existen resultados de otros resolutores para comparar el rendimiento de forma que puede considerarse un primer estudio sobre el que pueden compararse otros resolutores MinSAT. Podemos ver en las tablas de resultados que MinSAT resulta más sencillo de resolver que MaxSAT en la mayoría de casos.

Instance set	#	MinSAT	MaxSAT
CircuitTraceCompaction	4	**11.82(1)**	0.00(0)
HaplotypeAssembly	6	1986.66(2)	**1176.94(2)**
PROTEIN_INS	12	**215.79(3)**	2.21(1)
bcp-fir	59	55.19(58)	111.29(59)
bcp-hipp-yRa1SU	38	**9.51(38)**	0.00(0)
bcp-hipp-yRa1simp	17	**2.21(17)**	10.55(5)
bcp-msp	64	**89.09(45)**	668.66(32)
bcp-mtg	40	**252.91(39)**	838.50(23)
bcp-syn	74	**0.82(74)**	36.18(71)
pbo-mqcnencdr	84	0.00(0)	2714.15(5)
pbo-mqcnlogencdr	84	4395.75(3)	**1874.64(6)**
pbo-routing	15	1395.96(4)	**130.38(15)**
Total	497	**284**	219

Cuadro 5.11: Partial MinSAT Industrial. Media de tiempo en segundos.

Instance set	#	MinSAT	MaxSAT
RAMSEY	48	0.00(0)	0.00(0)
WMAXCUTDIMACS_MOD	62	**0.77(62)**	1412.04(31)
WMAXCUTSPINGLASS	5	**116.37(4)**	228.77(4)
frb	34	**8.61(34)**	616.09(19)
Total	149	**100**	54

Cuadro 5.12: Weighted MinSAT Crafted. Media de tiempo en segundos.

Instance set	#	MinSAT	MaxSAT
AUCTIONSAUC_PATHS	88	**0.21(88)**	0.24(88)
AUCTIONSAUC_SCHEDULING	84	**0.46(84)**	0.81(84)
PSEUDOmiplib	12	**285.87(8)**	180.74(3)
WCSPSPOT5DIR	21	**0.84(21)**	202.44(18)
WCSPSPOT5LOG	21	**1.51(21)**	534.06(8)
min-encplanning	56	212.00(44)	556.99(52)
min-encwarehouses	18	1.21(18)	1.02(18)
random-net	78	**0.98(78)**	1029.10(59)
Total	378	**362**	330

Cuadro 5.13: Weighted Partial MinSAT Crafted. Media de tiempo en segundos.

Instance set	#	MaxSAT	MinSAT
timetabling	32	0.00(0)	0.00(0)
upgradeability-problem	100	11.34(100)	15.14(100)
Total	132	100	100

Cuadro 5.14: Weighted Partial MinSAT Industrial. Media de tiempo en segundos.

Capítulo 6

Conclusiones y trabajo futuro

Podemos ver en el apartado de resultados que el resolutor implementado en CPLEX se comporta muy satisfactoriamente en algunas familias de problemas. Quedaría pendiente comprobar como se comportan otros resolutores LIP como Gurobi o SCIP usando la misma traducción de problema.

Otro punto por estudiar son los algoritmos usados por CPLEX para resolver algunas familias de problemas eficientemente y ver como se podrían aplicar a resolutores MaxSAT, así como ver por qué CPLEX se comporta mucho peor en ciertas familias de problemas llegando a no resolver ninguno dado el mismo tiempo que otros resolutores.

En cuanto al problema MinSAT sería necesario comparar los resultados con los obtenidos por otros resolutores. Así como estudiar por qué en algunas familias de problemas parece ser mucho más rápido resolver el problema MinSAT que el problema MaxSAT.

En el resolutor se ha implementado un parámetro que permite realizar un tuning de los parámetros de CPLEX, para las familias de problemas que estan dando malos resultados con el resolutor se puede encontrar configuraciones alternativas de CPLEX que puedan dar mejores resultados.

Bibliografía

[1] *IBM ILOG Optimization and Supply Chain Applications Library*. http://www-01.ibm.com/software/websphere/products/optimization/library/.

[2] Carlos Ansotegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'09)*, 2009.

[3] Carlos Ansotegui, Maria Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial maxsat. In *Proc. the 24th National Conference on Artificial Intelligence (AAAI'10)*, 2010.

[4] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. The first and second Max-SAT evaluations. *Journal on Satisfiability*, 4:251–278, 2008.

[5] Adi Avidor and Uri Zwick. Approximating min k-sat*.

[6] Daniel Le Berre. Sat4jmaxsat. In *www.sat4j.org*.

[7] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In *Proc. of the 10th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'07)*, pages 41–55, 2007.

[8] IBM. Download software catalog. https://www.ibm.com/developerworks/university/software/get_software.html.

[9] IBM. Ibm's academic initiative program. http://www.ibm.com/academicinitiative.

[10] IBM. Ilog support. https://www.ibm.com/developerworks/university/support/ilog.htm.

[11] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Exploiting cycle structures in Max-SAT. In *SAT'09*, 2009.

[12] Han Lin, Kaile Su, and Chu Min Li. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *Proc. the 23th National Conference on Artificial Intelligence (AAAI'08)*, pages 351–356, 2008.

[13] Vasco Manquinho, João Marques-Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In *SAT'09*, pages 495–508, 2009.

[14] H. D. Mittelmann. Recent benchmarks of optimization software. *EURO XXII Prague, Czech Republic*, (22nd), July 2007.