Universidad Simón Bolívar (*Sede De Sartenejas*)
Departamento de Computación y Tecnología de la Información
CI-5651 - *Diseño de Algoritmo I*
Enero - Marzo 2008

Fase 2 del Proyecto

(*BSP, Heurística de Decisión, Backtracking No Cronológico y Aprendizaje*)

Profesor:
Oscar Meza

Integrantes:
Ricardo Monascal, *03-36207*
Cristina Sanjuán, *03-36486*

Introducción:

En la primera fase del proyecto se implementó un resolvedor de SAT, con el objetivo de resolver instancias de SUDOKU por medio de una reducción de cada una de estas instancias a una instancia de SAT. Dicho resolvedor era capaz de resolver una cantidad grande de los casos de pruebas a los que fue expuesto, sin embargo hubo instancias que el resolvedor no pudo manejar en un tiempo razonable.

Realizando una revisión más profunda del anterior proyecto y re-implementandolo de una nueva manera, se logró correr todos los casos en un tiempo mucho menor a un segundo, comparable con los tiempos obtenidos por el resolvedor *zchaff*. El resolvedor hasta el punto de la entrega pasada estaba basado en el algoritmo DPLL, incluido el análisis de implicaciones o *Boolean Constraint Propagation*. Los resultados obtenidos (con este enfoque re-implementado) apuntan a que dicha organización es suficiente para obtener buenos resultados con instancias de SAT tan pequeñas (Para un SUDOKU clásico, la reducción arroja 729 variables y un poco mas de 11988 cláusulas). Hay que recordar que las instancias de SAT utilizadas para poner a prueba los revolvedores actualmente pueden tener millones de variables y miles de millones

de cláusulas. Nuestra sospecha, respaldada por el esfuerzo que hacen la mayoría de los revolvedores eficientes actuales, es que para estas instancias de SAT gigantes no basta solo BSP.

Entonces se decidió incorporar algunas de las técnicas modernas para revolvedores de SAT, como lo son una heurística de decisión (se decide que variable, entre las libres, se va a asignar en cada momento, basado en alguna función de *conveniencia*), el backtracking no cronológico (se permite retroceder múltiples niveles en la *recursión*) y el aprendizaje de cláusulas (se permite la incorporación de cláusulas que son redundantes a la formula original, pero pueden ayudar a encontrar implicaciones antes ignoradas).

En el presente informe se exponen con más detalle las técnicas implementadas. En primer lugar el diseño general de la aplicación (Estructuras de datos y algoritmos utilizados), luego los detalles de la implementación, instrucciones para la operación de la aplicación, una breve descripción del estado actual del programa y terminando con algunas conclusiones y recomendaciones para trabajo futuro.

Diseño de la aplicación:

Con fines de comparación se han implementado varios revolvedores SAT diferentes que difieren entre sí en uno o varios aspectos, como la heurística de decisión o la incorporación de aprendizaje. Sin embargo, todos están basados en el algoritmo DPLL (Davis-Putnam-Logemann-Loveland) que sigue la siguiente estructura básica:

```
DPLL () {

        inicializar_entorno ()
        if (estado != DESCONOCIDO) return estado
        while (true) {
                if (!decidir_proxima_variable ()) {
                        if (No se puede deshacer) return INSATISFACIBLE
                        while (true) {
                                estado = deshacer ()
                                if (estado == INSATISFACIBLE) return INSATISFACIBLE
                                else if (estado == SATISFECHO) return SATISFECHO
                                else if (estado == DESCONOCIDO) break
                        }
```

```
                    } else {
                            empilar_evento(Asignacion de Decision)
                            estado = asignar_variable ()
                            if (estado == SATISFECHO) return SATISFECHO
                            else if (estado == CONFLICTO) {
                                    if (No se puede deshacer) return INSATISFACIBLE
                                    while (true) {
                                            estado = deshacer ()
                                            if (estado == INSATISFACIBLE) return INSATISFACIBLE
                                            else if (estado == SATISFECHO) return SATISFECHO
                                            else if (estado == DESCONOCIDO) break
                                    }
                            }
                    }
                    while (true) {
                            estado = propagar ()
                            if (estado == CONFLICTO) {
                                    if (No se puede deshacer) return INSATISFACIBLE
                                    while (true) {
                                            estado = deshacer ()
                                            if (estado == INSATISFACIBLE) return INSATISFACIBLE
                                            else if (estado == SATISFECHO) return SATISFECHO
                                            else if (estado == DESCONOCIDO) break
                                    }
                            } else if (estado == SATISFECHO) return SATISFECHO
                            else break
                    }
            }

}
```

Figura 1. Algoritmo DPLL

De cómo se implementen las funciones decidir_proxima_variable(), y deshacer() dependerá el funcionamiento del algoritmo. Por ejemplo, el algoritmo DPLL original puede obtenerse haciendo que decidir_proxima_variable() retorne siempre la ultima variable que no ha agotado sus dos asignaciones posible y que deshacer() solo deshaga las implicaciones ocurridas por asignaciones un nivel inmediatamente anterior.

En cuanto a las estructuras de datos, para cada variación de la aplicación en general, se conserva un tipo de datos ***cláusula*** que tiene la cantidad de variables en dicha cláusula, un arreglo con los índices de las variables pertenecientes a dicha cláusula, la cantidad de variables que quedan sin asignar y si dicha cláusula ya ha sido satisfecha o no. Para las variables no se tiene una estructura de datos especial, sino solo un arreglo de enteros que representa el valor de la asignación de dicha variables (que puede ser TRUE, FALSE, o SIN_ASIGNAR). Para el manejo de la pseudo-recursión, que se implementa de manera iterativa se tiene una pila de eventos, donde cada evento tiene un tipo, un

índice y un valor. Los eventos pueden ser de tipo ASIG_BACKTRACK, que equivale a una decisión sobre una variable libre, de tipo ASIG_PROPAGACION que equivale a la asignación de una variable en respuesta a una implicación del BSP. En ambos casos el índice del evento, es el índice de la variable asignada y el valor del evento, el valor que le fue asignado. Un evento también puede ser de tipo SAT_CLAUSULA, que equivale a la satisfacción de una determinada cláusula, en cuyo caso el índice del evento, dependiendo de si el aprendizaje de cláusulas está activado o no, es el índice de la cláusula en cuestión ó la dirección de memoria de la misma, el valor del evento es irrelevante. En el caso en el que el aprendizaje de cláusulas esté activado, se guardan las cláusulas aprendidas en una lista, independiente del conjunto de cláusulas originales (por esto es importante que se empile la dirección de la cláusula y no su índice).

Para la heurística de decisión se implementaron tres métodos. El primero emula el comportamiento de la selección lineal, es decir la escogencia de la primera variable no asignada con el valor FALSE (luego se verá como se le asigna luego TRUE cuando la primera asignación lleva a un conflicto). El segundo método es el propuesto por los creadores de GRASP, el cual toma la variable y la polaridad tal que al asignársele dicho valor satisfaga directamente la mayor cantidad de cláusulas. El tercer método es el propuesto por los creadores de *chaff*, que ellos denominan VSIDS (Variable State Independant Decaying Sum), para la cual es estrictamente necesaria la incorporación del aprendizaje de cláusulas. A grosso modo, dicho método lleva dos contadores por cada variable (uno por cada polaridad) y cada vez que se aprenda una nueva cláusula se le suma un determinado valor a cada variable que participe de la misma en su polaridad correspondiente. A la hora de decidir la próxima variable a asignar, se toma aquella variable y polaridad que tenga el contador más alto. Con el objetivo de evitar el elitismo de variables que fueron muy utilizadas y luego ya no, se dividen periódicamente todos los contadores por alguna constante. En nuestro caso, el valor que sumamos es uno (1) y la constante por la que dividimos es dos (2). Dado el corto tiempo de ejecución para estas instancias pequeñas de SAT, se divide

siempre que se decida el valor de una nueva variable. (Cambios en estas constantes produjeron resultados idénticos, así que nos quedamos con uno solo).

La función deshacer() influye también mucho en el funcionamiento del resolvedor. De aquí viene la razón por la que solo hace falta asignarle a una variable un valor (por decisión, no propagación) en uno solo de sus dos valores posibles. La función deshacer, deshace asignaciones de propagación y satisfacciones de cláusulas hasta que se consiga una asignación de decisión. En este momento la misma función de deshacer puede asignar a la variable el valor contrario al que tenía (si no había agotado ya sus opciones) y empilar un nuevo evento de asignación por decisión. Sin embargo, se puede introducir el backtacking no cronológico a esta estrategia con muy poco esfuerzo. En vez de empilar el valor contrario de la variable como otra decisión, se empila como una propagación más, de esta manera la próxima vez que se llame a deshacer(), esta asignación será deshecha y se seguirá deshaciendo hasta encontrar alguna asignación de decisión (Nótese que este es exactamente el comportamiento que tiene el backtracking no cronológico, sin siquiera la necesidad de guardar un nivel para cada evento, ni guardar de manera explícita un grafo de implicaciones para calcular a que nivel se debe saltar).

El aprendizaje de cláusulas es como el propuesto para *GRASP*, se toman todas las variables asignadas por una decisión, anteriores a la ocurrencia del conflicto y se agrega una nueva cláusula conformada por el complemento de cada una de dichas asignaciones. Nótese ahora que cuando deshacer() empila la segunda rama de búsqueda de una variable de decisión como una asignación de propagación no lo hace sin sentido, ya que en la cláusula nueva la ultima variable de decisión es precisamente la única que quedaría sin asignar, y por BSP su valor es implicado.

Detalles de la implementación:

Como se ha dicho anteriormente, con fines de comparación se han implementado un conjunto de resolvedores SAT, todos basados en el algoritmo DPLL, pero que difieren en algunos aspectos, como la heurística de decisión de variables libre, o la incorporación de un sistema de aprendizaje de cláusulas.

En primer lugar se implementó una versión *Simple*, que emula el comportamiento del algoritmo DPLL original + BCP, y corresponde a la re-implementación de la primera fase del proyecto. En esta versión, la función que decide que variable libre ha de ser la próxima a asignar, toma la primera variable libre (sin asignar) ordenadas por su índice o etiqueta. La función que deshace implementa un pseudo-bactracking no cronológico, donde se aplica la misma técnica explicada anteriormente para dicha técnica, con la salvedad de que no se aprenden cláusulas de conflictos. De esta manera la asignación de propagación que se hace con la última variable asignada por su valor complemento, no proviene del resultado de un BSP, sino de la incorporación de una regla nueva. Si se llega a un conflicto, la última variable asignada (como mínimo) es incorrecta, por lo tanto debe implicarse que dicha variable adopte el valor de su complemento.

En segundo lugar, para otra versión se incorporó una heurística de decisión diferente. En este caso se incorporó la misma heurística que se utiliza en el resolvedor GRASP, la cual consta de lo siguiente: *Se calcula para cada variable y cada polaridad la cantidad de cláusulas que quedarían directamente satisfechas al asignar a dicha variable la polaridad correspondiente. Se escoge la variable y polaridad con el mayor valor (que satisfaga directamente la mayor cantidad de cláusulas)*. La función que deshace es la misma que se describió para de caso *Simple*.

Luego se tienen tres versiones adicionales que incorporan el aprendizaje de cláusulas. Las cláusulas aprendidas se guardan en una lista aparte de las cláusulas originales (leídas del archivo CNF), en atención a este cambio de estructura, cuando ocurre un evento de tipo SAT_CLAUSULA, que recordemos es cuando se ha satisfecho una cláusula. En las dos versiones anteriores, los eventos

de tipo SAT_CLAUSULA tenían en el índice, el índice de la cláusula en el orden en que fue leída del archivo. En atención a que ahora existen dos estructuras de datos que contienen cláusulas y una de ellas no tiene acceso de tiempo constante (Para alcanzar el i-ésimo elemento de una lista, hay que pasar necesariamente por los primeros i-1), se asigna como índice del evento, la dirección de memoria de dicha cláusula. Esto hace que la estructura en la que se encuentra la cláusula satisfecha es indiferente y el acceso a cualquiera de dichas cláusulas es constante. Nótese que en las funciones de propagación (que implementa BSP), incluyendo calcular los efectos de una asignación específica de una variable, la búsqueda de una cláusula unitaria, entre otros, el acceso a la lista de cláusulas es lineal, ya que deben recorrerse todas las cláusulas. Sin embargo una estructura de tipo lista es ideal para secuencias de objetos a los que se les va accediendo en un determinado orden. El tamaño de las cláusulas aprendidas, en otras palabras la cantidad de variables que incluyen, es ilimitado. De todas formas la cláusula más grande posible solo tendría 729 variables. La cantidad de cláusulas que pueden ser aprendidas es también ilimitada, dependiente únicamente de la cantidad de memoria disponible.

La primera versión que incluye aprendizaje, es idéntico al *Simple* en cuanto a la heurística de decisión y la manera de deshacer cuando hay conflicto. La única diferencia es (a nivel de concepto, en implementación es idéntico), que cuando ocurre un conflicto y la última variable de decisión es asignada el complemento de su valor actual a manera de propagación, se hace como consecuencia de lo que arrojaría el BSP sobre la nueva cláusula aprendida. La segunda versión es equivalente a su vez a la que implementa la heurística de decisión del GRASP. LA tercera y última versión implementa la heurística de decisión que proponen los creadores de *chaff*. Dicha decisión está basada en una heurística que ellos llaman VSIDS (Variable State Independant Decaying Sum), en donde para cada variable y cada polaridad se mantiene un contador que está activo durante toda la ejecución del programa. Cada vez que se aprenda una cláusula nueva, a cada variable que participe de dicha cláusula se le incrementa el contador en su polaridad correspondiente. De esta manera se escoge siempre la variable y

polaridad con el contador más elevado. De igual manera, periódicamente se debe ir dividiendo el valor de todos los contadores por una cierta constante. Como ya se mencionó antes el valor que se suma a los contadores es uno (1), la constante que divide es dos (2) y la periodicidad es siempre que se asigne una nueva variable de decisión. La razón por la que ésta es una buena heurística es por que se va a intentar primeramente asignar las variables que ocasionaron el conflicto en dirección a no volver a ocasionarlo.

El mayor problema encontrado a la hora de la implementación fue en realidad al intentar continuar con esta fase del proyecto, partiendo de lo que fue la primera entrega, lo cual nos fue imposible. Es por esto que se decidió primero re-implementar de otra manera el equivalente a la primera fase del proyecto. Esto nos permitió luego poder implementar las heurísticas de decisión, el bactracking no cronológico de manera relativamente rápida, sin embargo la re-implementación de dicha primera fase tomó una cantidad considerable de tiempo para que funcionara como nosotros queríamos.

Instrucciones de operación:

La aplicación completa, en todas sus versiones, deberá copiarse en un mismo directorio. Desde la consola del sistema, en el directorio donde se colocó la aplicación se debe ejecutar la instrucción '*make*', la cual se encarga de compilar todos los elementos necesarios de la aplicación. Para ejecutar cualquiera de las diferentes versiones se debe cambiar de directorio al que corresponda a la versión deseada y ejecutar una instrucción de la forma '*./proyecto_II <archivo de entrada> <plantilla de salida>*', donde *<archivo de entrada>* es un archivo que contenga un conjunto de instancias de SUDOKU en el formato acordado y *<plantilla de salida>* es un trozo de nombre da archivo que se va a utilizar para completar los nombres de los diferentes tipos de archivos de salida, como son los tableros individuales (*<plantilla de salida>*_*<numero de instancia>*.txt), los tableros traducidos a SAT (SAT_*<plantilla de salida>*_*<numero de instancia>*.txt) y los archivos de solucion, tanto para nuestro resolvedor 'SATurn' (solucion_SATurn_*<plantilla de salida>*

_<*numero de instancia*>.txt) como para el resolvedor de dominio público *zchaff* (solucion_zchaff_<*plantilla de salida*>_<*numero de instancia*>.txt). Para todos estos archivos <*numero de instancia*> es el orden de aparición de dicho tablero, de entre los tableros válidos, en <*archivo de entrada*>.

Estado Actual:

El estado actual de la aplicación es *totalmente operativo*, o por lo menos no se encontró ningún caso para el cual la aplicación falle. El único problema viene dado con algunas instancias de tablero utilizadas que corren por demasiado tiempo en algunas de las versiones implementadas.

Conclusiones y recomendaciones:

Para esta fase del proyecto, se implementó una mejora al resolvedor SAT que se había entregado como primera fase, las cuales incluyen heurísticas de decisión, backtracking no cronológico, y aprendizaje de cláusulas. Sin embargo se ha demostrado que para las instancias pequeñas de SAT que genera la reducción de un tablero clásico (729 variables y un poco más de 11988 cláusulas), tomando en cuenta que los revolvedores actuales resuelven instancias de SAT con millones de variables y miles de millones de cláusulas, la incorporación de dichas técnicas (salvando el backtracking no cronológico) no muestra mejora alguna. Este se debe a la estructura de la fórmula proporcionada y del overhead que implican dichas técnicas. A continuación se muestra una tabla con los tiempos de corrida para cada una de las versiones implementadas y *zchaff*, tomando 60 casos de prueba, repartidos en 6 niveles de dificultad. Las corridas se hicieron en unas de las maquinas del Laboratorio Docente de Computación (*baraka*) tomando como *timeout* para las corridas 10 minutos. Lo cual es razonable, ya que los revolvedores actuales, para resolver instancias gigantes de SAT utilizan un *timeout* que varía entre 1/2 hora y 3 horas. Además, como se podrá ver en la tabla, todas las versiones resuelven la instancia en menos de medio segundo o en

más de 10 minutos, de lo que se puede intuir que probablemente dichas instancias se tarden incluso mucho más que esta cota.

Simple	Heurística Grasp	Aprendizaje + Simple	Aprendizaje + Heurística Grasp	Aprendizaje + Heurística Chaff	zchaff
0 min. 0.118 seg.	0 min. 0.110 seg.	0 min. 0.113 seg.	0 min. 0.109 seg.	0 min. 0.107 seg.	0 min. 0.118 seg.
0 min. 0.115 seg.	0 min. 0.112 seg.	0 min. 0.110 seg.	0 min. 0.110 seg.	0 min. 0.109 seg.	0 min. 0.115 seg.
0 min. 0.114 seg.	0 min. 0.115 seg.	0 min. 0.113 seg.	0 min. 0.108 seg.	0 min. 0.108 seg.	0 min. 0.114 seg.
0 min. 0.112 seg.	0 min. 0.439 seg.	0 min. 0.106 seg.	0 min. 0.107 seg.	0 min. 0.104 seg.	0 min. 0.112 seg.
0 min. 0.114 seg.	0 min. 0.111 seg.	0 min. 0.107 seg.	0 min. 0.108 seg.	0 min. 0.105 seg.	0 min. 0.114 seg.
0 min. 0.112 seg.	0 min. 0.111 seg.	0 min. 0.107 seg.	0 min. 0.105 seg.	0 min. 0.104 seg.	0 min. 0.112 seg.
0 min. 0.112 seg.	0 min. 0.109 seg.	0 min. 0.106 seg.	0 min. 0.119 seg.	0 min. 0.104 seg.	0 min. 0.112 seg.
0 min. 0.135 seg.	0 min. 0.111 seg.	0 min. 0.107 seg.	0 min. 0.105 seg.	0 min. 0.108 seg.	0 min. 0.135 seg.
0 min. 0.114 seg.	0 min. 0.113 seg.	0 min. 0.111 seg.	0 min. 0.107 seg.	0 min. 0.104 seg.	0 min. 0.114 seg.
0 min. 0.115 seg.	0 min. 0.112 seg.	0 min. 0.108 seg.	0 min. 0.108 seg.	0 min. 0.109 seg.	0 min. 0.115 seg.
0 min. 0.111 seg.	0 min. 0.122 seg.	0 min. 0.108 seg.	0 min. 0.106 seg.	0 min. 0.107 seg.	0 min. 0.111 seg.
0 min. 0.114 seg.	0 min. 0.114 seg.	0 min. 0.112 seg.	0 min. 0.110 seg.	0 min. 0.109 seg.	0 min. 0.114 seg.
0 min. 0.113 seg.	0 min. 0.108 seg.	0 min. 0.108 seg.	0 min. 0.106 seg.	0 min. 0.106 seg.	0 min. 0.113 seg.
0 min. 0.118 seg.	0 min. 0.112 seg.	0 min. 0.121 seg.	0 min. 0.109 seg.	0 min. 0.105 seg.	0 min. 0.118 seg.
0 min. 0.110 seg.	0 min. 0.111 seg.	0 min. 0.107 seg.	0 min. 0.104 seg.	0 min. 0.103 seg.	0 min. 0.110 seg.
0 min. 0.111 seg.	0 min. 0.113 seg.	0 min. 0.105 seg.	0 min. 0.105 seg.	0 min. 0.104 seg.	0 min. 0.111 seg.
0 min. 0.139 seg.	0 min. 0.119 seg.	0 min. 0.108 seg.	0 min. 0.103 seg.	0 min. 0.105 seg.	0 min. 0.139 seg.
0 min. 0.109 seg.	0 min. 0.110 seg.	0 min. 0.106 seg.	0 min. 0.106 seg.	0 min. 0.104 seg.	0 min. 0.109 seg.
0 min. 0.111 seg.	0 min. 0.113 seg.	0 min. 0.105 seg.	0 min. 0.105 seg.	0 min. 0.105 seg.	0 min. 0.111 seg.
0 min. 0.111 seg.	0 min. 0.113 seg.	0 min. 0.108 seg.	0 min. 0.105 seg.	0 min. 0.104 seg.	0 min. 0.111 seg.
0 min. 0.121 seg.	0 min. 0.236 seg.	0 min. 0.117 seg.	0 min. 0.230 seg.	0 min. 0.107 seg.	0 min. 0.121 seg.
0 min. 0.120 seg.	0 min. 0.206 seg.	0 min. 0.118 seg.	0 min. 0.196 seg.	0 min. 0.117 seg.	0 min. 0.120 seg.
0 min. 0.113 seg.	0 min. 0.282 seg.	0 min. 0.112 seg.	0 min. 0.281 seg.	0 min. 0.106 seg.	0 min. 0.113 seg.
0 min. 0.116 seg.	0 min. 0.214 seg.	0 min. 0.112 seg.	0 min. 0.208 seg.	0 min. 0.118 seg.	0 min. 0.116 seg.

0 min. 0.120 seg.	0 min. 0.165 seg.	0 min. 0.116 seg.	0 min. 0.160 seg.	0 min. 0.109 seg.	0 min. 0.120 seg.
0 min. 0.163 seg.	-	0 min. 0.160 seg.	-	0 min. 0.824 seg.	0 min. 0.163 seg.
0 min. 0.120 seg.	0 min. 0.153 seg.	0 min. 0.116 seg.	0 min. 0.148 seg.	0 min. 0.109 seg.	0 min. 0.120 seg.
0 min. 0.113 seg.	0 min. 0.124 seg.	0 min. 0.109 seg.	0 min. 0.119 seg.	0 min. 0.113 seg.	0 min. 0.113 seg.
0 min. 0.111 seg.	0 min. 0.114 seg.	0 min. 0.110 seg.	0 min. 0.111 seg.	0 min. 0.107 seg.	0 min. 0.111 seg.
0 min. 0.126 seg.	0 min. 0.897 seg.	0 min. 0.122 seg.	0 min. 0.890 seg.	0 min. 0.107 seg.	0 min. 0.126 seg.
0 min. 0.110 seg.	0 min. 0.264 seg.	0 min. 0.110 seg.	0 min. 0.198 seg.	0 min. 0.125 seg.	0 min. 0.110 seg.
0 min. 0.552 seg.	0 min. 15.338 seg.	0 min. 0.569 seg.	0 min. 15.059 seg.	0 min. 0.274 seg.	0 min. 0.552 seg.
0 min. 0.114 seg.	-	0 min. 0.118 seg.	-	0 min. 0.155 seg.	0 min. 0.114 seg.
0 min. 0.135 seg.	0 min. 0.167 seg.	0 min. 0.142 seg.	0 min. 0.155 seg.	0 min. 0.111 seg.	0 min. 0.135 seg.
0 min. 0.451 seg.	0 min. 42.465 seg.	0 min. 0.449 seg.	1 min. 0.268 seg.	0 min. 0.305 seg.	0 min. 0.451 seg.
0 min. 0.119 seg.	0 min. 0.172 seg.	0 min. 0.116 seg.	0 min. 0.170 seg.	0 min. 0.115 seg.	0 min. 0.119 seg.
0 min. 0.111 seg.	0 min. 0.169 seg.	0 min. 0.117 seg.	0 min. 0.142 seg.	0 min. 0.111 seg.	0 min. 0.111 seg.
0 min. 0.114 seg.	0 min. 0.177 seg.	0 min. 0.113 seg.	0 min. 0.163 seg.	0 min. 0.108 seg.	0 min. 0.114 seg.
0 min. 0.118 seg.	0 min. 0.205 seg.	0 min. 0.180 seg.	0 min. 0.197 seg.	0 min. 0.111 seg.	0 min. 0.118 seg.
0 min. 0.109 seg.	0 min. 0.161 seg.	0 min. 0.104 seg.	0 min. 0.148 seg.	0 min. 0.113 seg.	0 min. 0.109 seg.
0 min. 0.118 seg.	0 min. 0.132 seg.	0 min. 0.111 seg.	0 min. 0.124 seg.	0 min. 0.109 seg.	0 min. 0.118 seg.
0 min. 0.116 seg.	0 min. 0.197 seg.	0 min. 0.109 seg.	0 min. 0.182 seg.	0 min. 0.119 seg.	0 min. 0.116 seg.
0 min. 0.121 seg.	0 min. 0.170 seg.	0 min. 0.122 seg.	0 min. 0.168 seg.	0 min. 0.114 seg.	0 min. 0.121 seg.
0 min. 0.143 seg.	-	0 min. 0.136 seg.	-	0 min. 0.114 seg.	0 min. 0.143 seg.
0 min. 0.119 seg.	0 min. 0.218 seg.	0 min. 0.116 seg.	0 min. 0.212 seg.	0 min. 0.115 seg.	0 min. 0.119 seg.
0 min. 0.266 seg.	-	0 min. 0.256 seg.	-	0 min. 0.423 seg.	0 min. 0.266 seg.
0 min. 0.158 seg.	-	0 min. 0.153 seg.	-	0 min. 0.113 seg.	0 min. 0.158 seg.
0 min. 0.109 seg.	0 min. 0.128 seg.	0 min. 0.106 seg.	0 min. 0.120 seg.	0 min. 0.112 seg.	0 min. 0.109 seg.
0 min. 0.109 seg.	0 min. 0.132 seg.	0 min. 0.105 seg.	0 min. 0.126 seg.	0 min. 0.114 seg.	0 min. 0.109 seg.
0 min. 0.110 seg.	0 min. 0.132 seg.	0 min. 0.105 seg.	0 min. 0.125 seg.	0 min. 0.115 seg.	0 min. 0.110 seg.
0 min. 0.117 seg.	0 min. 0.503 seg.	0 min. 0.113 seg.	0 min. 0.507 seg.	0 min. 0.116 seg.	0 min. 0.117 seg.

0 min. 0.111 seg.	0 min. 0.119 seg.	0 min. 0.111 seg.	0 min. 0.113 seg.	0 min. 0.108 seg.	0 min. 0.111 seg.
0 min. 0.131 seg.	0 min. 0.520 seg.	0 min. 0.119 seg.	0 min. 0.505 seg.	0 min. 0.115 seg.	0 min. 0.131 seg.
0 min. 0.114 seg.	0 min. 0.181 seg.	0 min. 0.144 seg.	0 min. 0.172 seg.	0 min. 0.118 seg.	0 min. 0.114 seg.
0 min. 0.110 seg.	0 min. 0.297 seg.	0 min. 0.115 seg.	0 min. 0.287 seg.	0 min. 0.125 seg.	0 min. 0.110 seg.
0 min. 0.112 seg.	0 min. 0.146 seg.	0 min. 0.259 seg.	0 min. 0.135 seg.	0 min. 0.116 seg.	0 min. 0.112 seg.
0 min. 0.114 seg.	0 min. 0.293 seg.	0 min. 0.159 seg.	0 min. 0.258 seg.	0 min. 0.114 seg.	0 min. 0.114 seg.
0 min. 0.140 seg.	0 min. 0.413 seg.	0 min. 0.106 seg.	0 min. 0.395 seg.	0 min. 0.106 seg.	0 min. 0.140 seg.
0 min. 0.112 seg.	0 min. 0.202 seg.	0 min. 0.104 seg.	0 min. 0.136 seg.	0 min. 0.126 seg.	0 min. 0.112 seg.

Como se puede ver se obtuvieron para el resolvedor *Simple* y *Aprendizaje + Heurística Chaff*, esto se debe en gran medida a que para todas las versiones se implementó un mecanismo de backtracking no cronológico, en el caso de *Simple* no existe ningún overhead por incluir las técnicas propuestas y en el caso de *Aprendizaje + Heurística Chaff* la combinación de las técnicas junto con una heurística que funciona muy bien, compensa dicho overhead. Para ambos caso en los que se implementó la heurística de GRASP, se tienen casos que no corren a tiempo y los demás tienden a correr un poco más lento. Esto se debe a la incorporación de una heurística de decisión que probablemente no sea adecuada para la estructura y el tamaño de las formulas CNF proporcionadas. La versión *Aprendizaje + Simple*, funciona casi tan bien como las dos nombradas primeramente, sin embargo los tiempos de ejecución ligeramente mayores se deben al overhead que implica el aprendizaje de cláusulas, que sin una buena heurística que aproveche dicho aprendizaje, se hace apreciable.

Estamos satisfechos con los resultados obtenidos por nuestro resolvedor, que por lo menos para las instancias de SAT proporcionada se comporta de manera competitiva con el resolvedor *zchaff*, considerado como uno de los actuales revolvedores del estado del arte. Sin embargo, aún podrían incorporare algunas mejoras a nivel de implementación, como el reemplazo de los arreglos contiguos en memoria, por listas enlazadas (permitiendo un aprovechamiento mejor de la memoria disponible) y apuntadores a los elementos necesarios cuando

el orden de acceso deba ser constante. Ya se implementó una primera aproximación a esta idea con la lista de cláusulas aprendidas.

Referencias Bibliográficas:

[1] Joao P. Marques Silva y Karem A. Sakallah, *GRASP-A New Search Algorithm For Satisfiability*

[2] Mathew W. Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang y Sharad Malik, *Chaff: Engineering An Efficient SAT Solver*

[3] Lintao Zhing y Sharad Malik, *The Quest For Efficient Boolean Satisfiability Solvers*