A Hybrid Approach to Solve SAT and UNSAT Problems

F. Silva, J. Neves, S. Dias, L. Zárate, and M. Song

***Abstract*—This work involves the construction of a hybrid approach to solve SAT and UNSAT problems. The solution combines techniques found in Stalmarck and DPLL algorithms for solving propositional satisfiability problems. We extend the Stalmarck rules in order to reduce computational cost during the deduction phase. We applied our solution to SAT and UNSAT instances. From the results it was found that this approach showed good results for UNSAT industrial instances. In some cases, our approach obtained gains of 45% to 60% in terms of efficient when compared to existing approaches. We compared our solution against the best known SAT solvers such as ZChaff and RSat. The techniques and methods involved are described in the article.**

***Index Terms*— DPLL, SAT, Stalmarck, UNSAT.**

I. Introdução

DIVERSOS trabalhos têm contribuído para a evolução dos solucionadores reduzindo o tempo de processamento e aumentando o tamanho e complexidade do problema a ser resolvido [1, 2]. Como pode-se observar em [3], anualmente competições são realizadas com o objetivo de buscar solucionadores cada vez mais eficiente para a solução de problemas da natureza deste trabalho.

Atualmente são inúmeras as aplicações SAT, dentre as quais cita-se: a verificação de equivalência em circuitos combinatórios [4], a identificação e remoção de redundâncias em circuitos combinatórios [5], a verificação de microprocessadores [6], a utilização em teoria da complexidade [7], em arquitetura de computadores - alocação de registradores [8], em testes de algoritmos de criptografia [9] e na análise de DNA para tratamento de doenças [10]. Além destes, existem trabalhos que procuram realizar a análise, correção e verificação de sistemas utilizando diferentes técnicas. Por exemplo, em [11] os autores propõem uma abordagem para realizar a verificação formal de um sistema distribuído modelando a comunicação como um programa concorrente, instanciando o sistema distribuído utilizando encadeamentos e filas atômicas. Já em [12] os autores realizam uma análise automática de imagens para identificação de falhas em plantações.

F. Silva, J. Neves, S. Mariano, L. Zárate, M. Song are with the Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, Minas Gerais, Brasil, flavio.silva@hotmail.com, {juliocesar.neves, sergiomariano}@gmail.com, {zarate, mark}@pucminas.br

Basicamente, um problema SAT consiste na atribuição de valores às variáveis de uma expressão lógica disposta em forma normal conjuntiva. Uma simples abordagem para o problema é dispor tais variáveis na forma de uma árvore binária de decisão, percorrendo-se os caminhos à procura de uma solução. Entretanto, esta proposta é inviável na prática: considerando-se uma expressão com n variáveis tem-se 2^n caminhos distintos a serem percorridos. Em problemas reais, que muitas vezes são modelados com milhões de variáveis, a abordagem é impraticável.

Um problema denominado satisfazível, também conhecido como problema SAT (*satisfiable* ou satisfatível), é aquele caracterizado por uma formulação de expressões proposicionais. Esta tem uma importância teórica uma vez que esta foi a primeira vez que provou ser um problema NP-Completo [7]. Neste caso, modela-se o sistema como um problema SAT e com o uso de verificadores SAT torna-se possível realizara análise, a verificação e a geração de casos de teste para sistemas de hardware e/ou software, também contribuindo para a eliminação e correção de erros nas fases iniciais de um projeto quando o custo de corrigi-los é ainda baixo.

Muitas abordagens têm sido propostas com o objetivo de resolver tal problema, como por exemplo, abordagens baseadas em DPLL (*Davis–Putnam–Logemann–Loveland*) – atualmente a base de várias soluções [13]. As principais soluções propostas compartilham das mesmas heurísticas, como o aprendizado de cláusulas de conflito, backtracking e outros. A maioria sendo aplicada para problemas SAT.

O objetivo deste artigo é apresentar uma abordagem híbrida para resolver problemas SAT baseados em DPLL e métodos de Stalmarck que também é uma abordagem para a resolução de problemas UNSAT (*unsatisfiable* ou não satisfatível), o que diferencia este trabalho dos demais.

Na Seção II será descrito o problema SAT e a definição de tautologia e contradição. Na Seção III apresenta as abordagens de DPLL e *Stalmarck*. Na Seção IV a abordagem híbrida para problemas SAT. Na seção V apresentamos os resultados da solução híbrida e na Seção VI as conclusões e trabalhos futuros.

II. O Problema SAT

O problema satisfazível (problema SAT) pode ser declarado como: dada uma expressão booleana E, decida se há alguma atribuição às variáveis em E, de modo que E seja verdade.

Basicamente, o problema é descrito como uma fórmula proposicional, normalmente expressa em uma forma normal,

que é composta por variáveis proposicionais, operadores lógicos (∧ e ∨) e o operador unário ¬. Podem-se considerar as variáveis proposicionais que assumem apenas valores verdadeiros ou falsos.

Uma fórmula *f* é dita estar em CNF (*conjunctive normal form* ou forma conjuntiva normal) se está definida sobre as conjunções de cláusula. Em cada cláusula, no máximo, os conectivos ($c_1 \wedge c_2 \wedge c_3 \wedge ... \wedge c_n$) são apresentados. O operador ¬ não pode ser repetido (como ¬ ¬) e é aplicado apenas a variáveis proposicionais. Observe que *f* é satisfazível apenas se todas as cláusulas forem consideradas como verdadeiras. Caso contrário, ou seja, não satisfazível, ocorre quando pelo menos, uma cláusula é avaliada como falsa.

Uma expressão booleana é satisfazível se houver pelo menos uma atribuição booleana para as variáveis que avaliam a expressão CNF como verdadeira. O problema pertence a uma classe específica de problemas de decisão, ou seja, problemas que admitem apenas uma resposta sim ou nenhuma. Basicamente, a tarefa envolvendo um solucionador SAT é encontrar uma atribuição de valores para as variáveis booleanas para satisfazer todas as cláusulas. Encontrar essa atribuição resolve-se então o problema.

Para avaliar dois conceitos de *f* são necessários: tautologia e contradição. A tautologia pode ser definida como uma expressão proposicional em que o resultado de sua tabela verdade contém apenas valor de verdade. Em outras palavras, a tautologia é qualquer expressão proposicional cujo valor lógico é sempre verdadeiro. Pelo contrário, a contradição pode ser definida como qualquer expressão proposicional que a sua tabela verdade contém apenas valores falsos, ou seja, para qualquer expressão proposicional cujo valor lógico é sempre falso.

III. Abordagem para solucionadores SAT

Esta seção descreve as abordagens DPLL e *Stalmarck* para resolver problemas SAT.

A. Metodologia DPLL

O primeiro algoritmo denotado DP (*Davis–Putnam*) resolve problemas SAT, mas limitado a uma resolução de espaço exponencial [14]. As melhorias no algoritmo DP resultaram na abordagem DPLL que procura reduzir o uso exponencial de memória.

Uma simples solução para problemas SAT é pesquisar atribuições de valores para variáveis representadas como uma árvore de decisão binária. Dada uma expressão com *n* variáveis, 2^n caminhos distintos podem ser gerados.

A solução DPLL consiste em um procedimento de busca em profundidade, com *backtracking,* através de todos os caminhos possíveis. Basicamente, este é composto pelas seguintes três etapas:

1. BCP (*Boolean Constraint Propagation* ou Propagação de restrição booleana) - se houver uma cláusula com apenas um literal, ou seja, uma cláusula de unidade, este literal deve fazer parte da solução. As cláusulas contendo o literal podem ser eliminadas e aquelas que contêm a sua forma de negação podem ser simplificados.
2. Eliminação pura de literal - as cláusulas contendo literais puros podem ser eliminadas. O literal é considerado puro se, em cada cláusula, aparecer apenas na sua forma positiva ou negativa.
3. Pesquisa para trás: considerando Φ uma fórmula na *CNF*, aplique o *BCP* e a eliminação literal pura em Φ. Escolha aleatoriamente uma variável *x* e execute o algoritmo para $\Phi(x)$. Se uma solução for encontrada, o problema é provado ser satisfazível. Caso contrário, execute o algoritmo para $\Phi(x)$. Se uma solução for encontrada, o problema é satisfazível. Se nenhuma solução foi encontrada, o problema é UNSAT.

A seguir, o esboço do DPLL é apresentado. Considerando Φ uma fórmula *CNF* e Π o conjunto de termos atribuídos como verdade, que está inicialmente vazio:

função DPLL(Φ, Π)
(1) se todas as cláusulas de Φ são verdadeiras
    então retorne verdadeiro;
(2) se uma ou mais cláusulas de Φ é falsa
    então retorne falso;
(3) se existe alguma única cláusula *c* em Φ
    então retorne DPLL (atribua (*c*, Φ), $\Pi \wedge$ c);
(4) se existe algum literal puro *c* em Φ
    então retorne DPLL (atribua (*c*, Φ), $\Pi \wedge$ c);
(5) *c*: = pesquisaliteral (Φ);
(6) retorne DPLL (atribua (*c*, Φ), $\Pi \wedge c$) ou
    DPLL (atribua ($\neg c$, Φ), $\Pi \wedge \neg c$);

Importante destacar que a maioria das ferramentas SAT são baseadas no DPLL. Basicamente, os desenvolvimentos são implementações diferentes para os mecanismos descritos.

B. A Abordagem Stalmarck

A abordagem de *Stalmarck* verifica tautologia em uma expressão proposicional [15]. O algoritmo difere da DPLL por confiar na busca em largura na pesquisa do espaço da solução. Por isso, um importante e preliminar passo é mudar uma instância do CNF em uma implicação (um processo de transformação) [16].

O método de Transformação consiste em converter uma expressão de CNF em uma IE (*implication expression* ou expressão de implicação). O processo de transformação é uma função de complexidade linear [17]. O método aplica as seguintes regras de transformação:

1) A ∨ B em ¬ A → B
2) A ∧ B em ¬ (A → ¬ B)
3) ¬ ¬A em A
4) ¬ A em A → 0

O uso de regras de transformação resulta em uma expressão que contém variáveis, parênteses e a implicação conectiva (→). Considere a seguinte expressão CNF: ($a \vee b$) ∧ ($\neg a$). Aplicando as regras de transformação, obtém-se:

- (4): ($a \vee b$) ∧ ($a \rightarrow 0$)
- (1): ($\neg a \rightarrow b$) ∧ ($a \rightarrow 0$)
- (4): (($a \rightarrow 0$) → b) ∧ ($a \rightarrow 0$)
- (2): ¬ ((($a \rightarrow 0$) → b) → ¬ ($a \rightarrow 0$))
- (4): ¬ ((($a \rightarrow 0$) → b) → (($a \rightarrow 0$) → 0))

- (4): ((((*a* → 0) → *b*) → ((*a* → 0) → 0)) → 0)

Uma expressão $A \rightarrow B$ pode ser definida como $C \equiv A \rightarrow B$ ou $C \equiv \neg A \vee B$. A nova variável C representa o resultado de $A \rightarrow B$ e o triplo (C, A, B) representa $C \equiv A \rightarrow B$.

O processo de transformação converte a expressão CNF $(a \vee b) \wedge (\neg a)$ em $((((a \rightarrow 0) \rightarrow b) \rightarrow ((a \rightarrow 0) \rightarrow 0)) \rightarrow 0)$. Convertendo a expressão de implicação em triplos resultados:

– $X_1 \equiv (a \rightarrow 0)$
– Expressão: $(((X_1 \rightarrow b) \rightarrow (X_1 \rightarrow 0)) \rightarrow 0)$
– $X_2 \equiv (X_1 \rightarrow 0)$
– $X_3 \equiv (X_1 \rightarrow b)$
– Expressão: $((X_3 \rightarrow X_2) \rightarrow 0)$
– $X_4 \equiv (X_3 \rightarrow X_2)$
– Expressão: $(X_4 \rightarrow 0)$
– $X_5 \equiv (X_4 \rightarrow 0)$
– Expressão: X_5

Note que o problema descrito como $(a \vee b) \wedge (\neg a)$ é agora representado pelas triplas $(X_5, X_4, 0)$, (X_4, X_3, X_2), (X_3, X_1, b), $(X_2, X_1, 0)$ e $(X_1, a, 0)$.

Conforme observado, a aplicação das regras de transformação cria novas variáveis para representar as implicações (X_1, X_2, X_3, X_4 e X_5). Pode-se observar que essas novas variáveis não são as apresentadas no problema original, mas apenas as representações do conectivo lógico (→). O problema original agora está reduzido ao X_5.

Aumentar o número de variáveis é um aspecto negativo do processo de *Stalmarck*. Mesmo assim, é um processo eficiente para verificar tautologia [15].

O método de resolução proposto por *Stalmarck* é uma prova por contradição. A expressão f é assumida como falsa. No caso de uma contradição a ser encontrada, a expressão é uma tautologia. Esta situação ocorre quando um terminal triplo (1, 1, 0), (0, *X*, 1) e (0, 0, *Y*) é obtido.

Considere, como exemplo, a expressão $a \rightarrow (b \rightarrow a)$, equivalente as triplas $X_1 \equiv b \rightarrow a$, $X_2 \equiv a \rightarrow X_1$. Observe que a variável X_2 representa o resultado da expressão inicial assumida como falsa (0). A partir desta decisão, uma série de ações é desencadeada com base nas triplas. Essas ações são os resultados da aplicação das sete regras de *Stalmarck* descritas a seguir.

Considerando uma tripla genérica (R, X, Y) onde $R \equiv X \rightarrow Y$ ou $R \equiv \neg X \vee Y$, são tomadas as seguintes decisões:

Regra 1: $(0, X, Y) \Rightarrow X = 1$ e $Y = 0$
Regra 2: $(R, X, 1) \Rightarrow R = 1$
Regra 3: $(R, 0, Y) \Rightarrow R = 1$
Regra 4: $(R, 1, Y) \Rightarrow R = Y$
Regra 5: $(R, X, 0) \Rightarrow R = \neg X$
Regra 6: $(R, R, Y) \Rightarrow R = 1$ e $Y = 1$
Regra 7: $(R, X, X) \Rightarrow R = 1$

As sete regras acima são responsáveis por escolher os valores corretos a serem atribuídos às variáveis para resolver o problema [18, 19, 20, 21, 22, 23, 24].

O exemplo a seguir ilustra a aplicação das sete regras. Considerando a expressão $a \rightarrow (b \rightarrow a)$, representada pelas triplas (X_2, a, X_1) e (X_1, b, a):

- assumindo que $X_2 = 0$, pode ser aplicado a Regra 1 em (X_2, a, X_1), portanto, resultando em $a = 1$ e $X_1 = 0$,
- por causa disso, obtém-se a tripla $(0, b, 1)$, que é um terminal triplo.
- pode-se deduzir que, considerando a expressão equivalente a falhas falsas, a uma contradição. Então, a expressão é uma tautologia.

Note que as sete regras não podem ser sempre aplicadas. Como, por exemplo, a situação em que é obtida uma tripla (1, *X*, 1). Neste caso não há nenhuma regra a ser aplicada, uma vez que dois valores possíveis podem ser atribuídos à variável *X*. A situação é definida como um dilema [25, 26].

A atividade a ser realizada é verificar os dois valores possíveis para a variável *X*. É importante definir um conjunto S_1 das atribuições geradas quando $X = 1$ e um conjunto S_0 das atribuições geradas quando $X = 0$. Se uma contradição for encontrada de ambas as formas, então uma tautologia é alcançada. Se a contradição for encontrada em um dos dois caminhos, por exemplo, S_1, então as atribuições S_0 são válidas. Se nenhuma contradição foi encontrada em ambos os caminhos, então $S_1 \cap S_0$ [27,28].

C. Saturação x Complexidade

A saturação é um termo relacionado à extensão do método do dilema. Existem problemas cuja solução pode ser encontrada, aplicando apenas as sete regras, por exemplo, a → (b → a). Observe que esta expressão leva a uma solução (tautologia) sem qualquer aplicação do processo de dilema. Estes problemas são considerados como 0-saturação. Pode-se compreender 0-saturação como uma classe de problemas cuja resolução é dada pela avaliação de um único caminho que atribui valores a variáveis.

O problema no qual a solução não pode ser encontrada apenas pelas sete regras, ou seja, usando a abordagem de dilema, são considerados *n*-saturação ($1 \leq n \leq \infty$) onde *n* refere-se ao número de variáveis simultaneamente trazidas para o dilema.

Por exemplo, na seção anterior apenas avaliou-se uma variável *X*. Isso levou à análise simultânea de 2^1 caminhos. Os problemas resolvidos desta forma são designados como 1-saturação.

Em alguns casos, a solução não pode ser encontrada através da 1-saturação. Portanto, é necessário expandir a regra do dilema em 2 variáveis diferentes, ou seja, *X* e *Y*. Observe que 2 variáveis no dilema exigem a análise de 2^2 caminhos simultaneamente - referente a diferentes tarefas para as variáveis *X* e *Y* (00, 01,10,11).

O algoritmo de *Stalmarck* aumenta sucessivamente o método do dilema para n-saturação, onde n é o número de variáveis do problema. A solução proposta é O (n^{2k+1}), em que *n* é o número de variáveis de instância e *k* o nível de saturação.

Note que, para a situação extrema onde $k = n$ o algoritmo tem desempenho ruim. Mas de acordo com *Stalmarck*, na prática, a maioria dos problemas reais (industrial) pode ser resolvida dentro de 2-saturações [15]. A análise do algoritmo *Stalmarck* que compara a solução proposta e a abordagem DPLL pode ser encontrada em [29].

IV. ABORDAGEM HÍBRIDA

A seção anterior apresentou a abordagem *Stalmarck* e DPLL. Esta seção pretende apresentar uma ferramenta híbrida baseada em ambos os métodos.

A partir da análise anterior, pode-se observar que as expressões tautológicas se assemelham a uma contradição. Enquanto o primeiro tem apenas valores verdadeiros, o outro tem apenas valores falsos. Assim, o método *Stalmarck* também pode ser usado para verificar se uma expressão proposicional é uma contradição. Neste caso, um problema UNSAT.

Com base nessas observações, foi criada uma ferramenta, para lidar com instâncias UNSAT. Esta solução foi usada para verificar instâncias industriais UNSAT. A solução pressupõe que a expressão é SAT. Se uma contradição (terminal triplo) for encontrada, a expressão não é satisfatória, caso contrário, é confirmado o SAT.

Depois, converte este em uma expressão de implicação - um conjunto de triplas. Em seguida, o conjunto é submetido aos processos de decisão, dedução e saturação baseados na abordagem *Stalmarck*.

É importante notar que as sete regras definem uma relação de causa e consequência. Por exemplo, ao aplicar a regra 1, cada tripla ($R = 0$, $X =?$, $Y =?$) Pode ser reduzido para $X = 1$ e $Y = 0$.

Propomos algumas extensões, com base nas sete regras originais, para reduzir custos computacionais durante o processo de dedução:

- Regra 1: $(0, X, Y) \Rightarrow X = 1$ e $Y = 0$
  Extensões: $(R, 1,0) \Rightarrow R = 0$
- Regra 2: $(R, X, 1) \Rightarrow R = 1$
  Extensões: $(R, 0,1) \Rightarrow R = 1$, $(R, 1,1) \Rightarrow R = 1$
- Regra 3: $(R, 0, Y) \Rightarrow R = 1$
  Extensões: $(R, 0,0) \Rightarrow R = 1$, $(R, 0,1) \Rightarrow R = 1$

V. RESULTADOS: ABORDAGEM HÍBRIDA X ZCHAFF X RSAT

Esta seção apresenta os resultados alcançados pela ferramenta. Conforme destacado em [15], o método de *Stalmarck* apresenta resultados melhores para problemas industriais. Assim sendo, foi feita uma avaliação do desempenho da abordagem híbrida, nesta classe de problemas, em relação a dois solucionadores baseados em DPLL.

Para as análises, foram utilizadas cerca de duzentas instâncias, retiradas das competições SAT [3]. A Tabela I é uma amostra retirada do universo de testes. Nesta tabela, os tempos de execução estão expostos em segundos, sendo o limite para time out de 3.600s.

O primeiro solucionador escolhido foi o *ZChaff*, por ser o campeão da competição SAT (classe industrial) em 2004 e pelo fato da primeira etapa de resolução da abordagem híbrida ser baseada em seu código [3]. Já o solucionador *RSat* foi escolhido por ser eficiente para instâncias industriais UNSAT, sendo o campeão da competição SAT nesta categoria em 2007 [3].

A seguir apresentamos uma breve descrição das Instâncias presentes na Tabela I:

- *bmc*.cnf: problemas BMC - *bounded model checking*.
- eq*.cnf: problemas de verificação de equivalência em hardware.
- phyhala*.cnf: problemas construídos. Forma de construção: *A permuted SAT Competition formula with seed*.
- *pipe*.cnf: fórmulas geradas na verificação formal de microprocessadores superescalares [30].
- ndhf-xits-*.cnf / rbcl-xits-*.cnf / rpoc-xits-*.cnf: Bio-Instâncias submetidas a competição do SAT de 2009.

Como esperado a solução baseada em *Stalmarck* não obteve bom desempenho para instâncias satisfazíveis. A Tabela I mostra os resultados da execução dos três solucionadores para problemas industriais UNSAT de diferentes naturezas. Como pode ser observada nesta tabela, a abordagem híbrida apresentou o seguinte resultado, em termos de tempo de execução:

– Para as instâncias *pyhala*.cnf* foi em média 45% superior ao *ZChaff* e 80% superior em relação ao *RSat*.

– Para as instâncias *eq*.cnf* foi significativamente mais rápido que os outros dois solucionadores, resolvendo os problemas centenas e em alguns casos milhares de vezes mais rápido.

– Para as instâncias **xits*.cnf* obteve desempenho superior, pois resolveu a maioria dos problemas dentro dos 3.600s enquanto as outras ferramentas não conseguiram resolver as mesmas instâncias no prazo. Destacando que para a instância *rbcl-xits-06-UNSAT.cnf* o solucionador *RSat* obteve desempenho superior e em alguns casos as três ferramentas abortaram por *time out*.

– Para as instâncias **pipe*.cnf* foi em média 5% inferior ao *ZChaff* e 28% superior ao *RSat*. Destacando que para instâncias com maior volume de variáveis da abordagem híbrida foi superior também ao *ZChaff*.

– Para as instâncias **bmc*.cnf* a abordagem híbrida não apresentou bom desempenho, só conseguindo resolver a instância *hoons-vbmc-lucky7.cnf* com desempenho próximo aos dos outros solucionadores.

Observando as instâncias relativas a verificação de equivalência em hardware (*eq*.cnf*), em que a abordagem híbrida obteve desempenho destacado, pode-se notar uma interessante característica: a maioria das cláusulas presentes nestas formulas são de tamanho igual ou inferior a 3 literais, incorrendo na geração de um número pequeno de triplas para se representar as cláusulas e consequentemente a instância.

Por este motivo intui-se que a velocidade da ferramenta está diretamente relacionada ao número de triplas necessário para se representar a instância, obtendo-se a solução mais rápida para problemas representados por um conjunto de cláusulas pequenas.

É importante destacar que os algoritmos utilizados são todos determinísticos e influenciados principalmente pelo número de variáveis e cláusulas de cada problema representado por certa instância.

TABELA I
ABORDAGEM HÍBRIDA X ZCHAFF X RSAT
(INSTÂNCIAS INDUSTRIAIS UNSAT)

Instância	Variáveis	Cláusulas	V-02	ZChaff	RSat
pyhala-braun-unsat-30-4-01.shuffled.cnf	5428	17845	47,834	74,771	86,174
pyhala-braun-unsat-30-4-02.shuffled.cnf	5428	17845	41,823	92,31	59,894
pyhala-braun-unsat-30-4-03.shuffled.cnf	5428	17845	1,236	68,912	61,102
pyhala-braun-unsat-30-4-04.shuffled.cnf	5428	17845	16,835	77,111	66,72
pyhala-braun-unsat-35-4-01.shuffled.cnf	7383	24320	324,363	260,008	818,648
pyhala-braun-unsat-35-4-02.shuffled.cnf	7383	24320	83,715	167,067	745,332
pyhala-braun-unsat-35-4-03.shuffled.cnf	7383	24320	198,187	244,009	790,554
pyhala-braun-unsat-35-4-04.shuffled.cnf	7383	24320	35,183	142,23	1125,864
pyhala-braun-unsat-40-4-01.shuffled.cnf	9638	31795	210,124	274,51	Time Out
pyhala-braun-unsat-40-4-02.shuffled.cnf	9638	31795	270,592	282,081	Time Out
pyhala-braun-unsat-40-4-03.shuffled.cnf	9638	31795	33,921	224,045	Time Out
pyhala-braun-unsat-40-4-04.shuffled.cnf	9638	31795	148,023	225,171	Time Out
eq.atree.braun.8.unsat.cnf	684	2300	0,117	9,27	133,215
eq.atree.braun.9.unsat.cnf	892	3006	0,162	72,94	505,156
eq.atree.braun.10.unsat.cnf	1111	3756	0,212	299,17	Time Out
eq.atree.braun.11.unsat.cnf	1400	4732	0,234	1918,11	Time Out
eq.atree.braun.12.unsat.cnf	1694	5726	0,304	Time Out	Time Out
eq.atree.braun.13.unsat.cnf	2010	6802	0,412	Time Out	Time Out
ndhf-xits-09-UNSAT.cnf	1910	167931	2579,763	Time Out	Time Out
ndhf-xits-10-UNSAT.cnf	2112	191788	2816,872	Time Out	Time Out
ndhf-xits-11-UNSAT.cnf	2316	216962	Time Out	Time Out	Time Out
ndhf-xits-12-UNSAT.cnf	2522	243459	Time Out	Time Out	Time Out
ndhf-xits-13-UNSAT.cnf	2730	271285	Time Out	Time Out	Time Out
ndhf-xits-14-UNSAT.cnf	2940	300446	Time Out	Time Out	Time Out
rbcl-xits-06-UNSAT.cnf	980	47620	1392,61	Time Out	80,515
rbcl-xits-07-UNSAT.cnf	1128	57446	1501,97	Time Out	2777,937
rbcl-xits-08-UNSAT.cnf	1278	68055	1623,239	Time Out	Time Out
rpoc-xits-07-UNSAT.cnf	1128	63345	1992,971	Time Out	Time Out
rpoc-xits-08-UNSAT.cnf	1278	74789	2099,875	Time Out	Time Out
rpoc-xits-09-UNSAT.cnf	1430	87044	2402,981	Time Out	Time Out
2pipe.cnf	892	6695	0,418	0,171	0,312
2pipe-1-ooo.cnf	834	7026	0,384	0,167	0,203
2pipe-2-ooo.cnf	925	8213	0,518	0,189	0,156
3pipe.cnf	2468	27533	2,771	2,221	3,14
3pipe-1-ooo.cnf	2223	26561	2,111	1,981	3,64
3pipe-2-ooo.cnf	2400	29981	2,579	2,911	5,828
3pipe-3-ooo.cnf	2577	33270	2,981	3,115	4,906
4pipe.cnf	5237	80213	13,696	16,297	33,653
4pipe-1-ooo.cnf	4647	74554	9,176	16,311	15,093
4pipe-2-ooo.cnf	4941	82207	10,872	35,299	24,261
4pipe-3-ooo.cnf	5233	89473	12,054	20,186	40,156
4pipe-4-ooo.cnf	5525	96480	13,132	22,11	41,203
5pipe.cnf	9471	195452	62,013	32,656	72,578
5pipe-1-ooo.cnf	8441	187545	48,694	60,008	57,812
5pipe-2-ooo.cnf	8851	201796	51,98	55,833	94,593
5pipe-3-ooo.cnf	9267	215440	53,785	57,255	89,765
5pipe-4-ooo.cnf	9764	221405	54,222	113,722	225,776
5pipe-5-ooo.cnf	10113	240892	62,605	66,733	83,406
6pipe.cnf	15800	394739	107,045	172,881	1194,969
6pipe-6-ooo	17064	545612	171,002	305,551	1560,226
7pipe	23910	751118	261,59	611,879	Time Out
hoons-vbmc-lucky7.cnf	8503	25116	31,216	63,54	44,36
cmu-bmc-longmult15.cnf	7807	24351	936,11	268,09	146,19
cmu-bmc-barrel6	2306	8931	Time Out	45,33	5,97

A execução de cada solucionador, para uma dada instância, sempre produz o mesmo resultado, pois os passos executados são sempre os mesmos. A execução repetida poderia ser influenciada pelo Sistema Operacional quando, por exemplo, sujeito a interrupções. Entretanto, testes realizados anteriormente para certas instâncias não resultou em diferenças significativas de tempo.

VI. CONCLUSÃO

O presente artigo descreve a construção de uma ferramenta de código aberto. Tal ferramenta tem por objetivo a resolução de instâncias do problema da satisfabilidade, utilizando a aplicação de conceitos dos algoritmos de *Stalmarck* e DPLL.

Conforme exposto neste artigo a abordagem híbrida apresentou bons resultados para instâncias industriais não satisfazíveis. Esta ferramenta possui uma etapa inicial baseada nos métodos de decisão e dedução presentes no algoritmo DPLL. E em uma segunda etapa o solucionador trabalha com métodos de transformação, decisão, dedução e saturação baseadas no algoritmo de *Stalmarck*. Alterações foram propostas em relação a alguns dos métodos originais apresentados por [15]. Tais alterações foram descritas neste artigo.

Como trabalho futuro almeja-se a inserção de uma heurística de decisão na abordagem híbrida. Uma ideia inicial seria a escolha das triplas a serem levadas à decisão com base em variáveis mais presentes em toda a instância. Com esta estratégia intui-se que mais triplas sejam resolvidas a cada decisão, tornando o processo de resolução mais rápido.

Outro trabalho interessante a ser realizado seria o refinamento dos procedimentos de decisão, dedução e tratamento de conflitos, baseados em DPLL, conjugando-os com estratégias propostas recentemente para ferramentas desta natureza.

AGRADECIMENTOS

Os autores agradecem ao suporte financeiro da CAPES, FAPEMIG e CNPq. Além disso, agradecem também a todos os pesquisadores da PUC-MG que levantaram algum questionamento importante no desenvolvimento deste trabalho.

REFERÊNCIAS

[1] Alexander Nadel's Page. [Online]. Available: https://www.cs.tau.ac.il/research/alexander.nadel/

[2] The MiniSat Page. [Online]. Available: http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/

[3] SAT Competition, "The international SAT Competitions", 2009. [Online]. Available: http://www.satcompetition.org/.

[4] J. P. Marques-Silva and Thomas Glass, "Combinational equivalence checking using satisfiability and recursive learning", In Proceedings of IEEE/ACM Design, Automation and Test in Europe (DATE), no. 33, 1999.

[5] Joonyoung Kim, J. P. Marques-Silva, Hamid Savoj and K. A. Sakallah, "RID-GRASP: Redundancy identification and removal using GRASP", In IEEE/ACM International Workshop on Logic Synthesis, 1997.

[6] M. N. Velev and R. E. Bryant, "Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors", In Proceedings of the Design Automation Conference (DAC), 2001.

[7] S. A. Cook, “The complexity of theorem-proving procedures”, Proceedings of the Third IEEE Symposium on the Foundations of Computer Science, pp. 151–158, 1971.

[8] N. R. Potlapally. “Solving register allocation using boolean satisfiability”. [Online]. Available: URL: http://www.princeton.edu/~npotlapa/files/papers/regalloc.pdf

[9] F. Massacci and L. Marraro. “Logical cryptanalysis as a SAT problem: encoding and analysis of the U. S. Data Encryption Standard”. Journal of Automated Reasoning, no. 24, pp.165-203, 2000.

[10] Inês Lynce and J. P. Marques-Silva. “SAT in Bioinformatics: Making the Case with Haplotype Inference”. In 9th International Conference on Theory and Applications of Satisfiability Testing, pp. 136-141, 2006.

[11] F. Cifuentes, J. Bustos, J. Simmonds. “Formal Verification of Distributed System Using an Executable C Model”. IEEE Latin America Transactions, vol. 14, no. 6, pages 2874-2878, 2016.

[12] J. P. R. Crulhas, A. O. Artero, M. A. Piteri, F. A. Silva, D. R. Pereira, D. M. Eler, J. P. Papa e V. H. C. de Albuquerque. “Blank Spots Identification on Plantations”, IEEE Latin America Transactions, vol. 16, no. 8, pages 2115-2121, 2018.

[13] M. Davis, G. Logemann, and D. Loveland, “A machine program for theorem proving”, Communications of the ACM. vol. 5, no. 7, pp. 394–397, 1962.

[14] M. Davis and H. Putnam, “A computer procedure for quantification theory”, In Journal of the ACM, vol. 7, pp. 201–215, 1960.

[15] M. Sheeran and G. Stalmarck, “A tutorial on Stalmarck’s proof procedure for propositional logic”, Formal Methods in Computer-Aided Design: Second International Conference, FMCAD, pp. 23-58, 1998.

[16] G. Stalmarck, “A System for Determining Propositional Logic Theorems by Applying Values and Rules to Triplets that are Generated from a Formula”, Swedish Patent No. 467 076 (approved 1992), U. S. Patent No. 5 276 897 (1994), European Patent No. 0403 454, 1995.

[17] G. Stalmarck, “A note on the computational complexity of the pure classical implication calculus”, Information Processing Letters, vol. 31, no. 6, pp. 277–278, 1989.

[18] E. W. Beth, “Semantic entailment and formal derivability”, Mededelingen der Kon, Nederlandse Akademie van Wetenschappen. vol. 18, pp. 309–342, 1955.

[19] G. Gentzen, “Untersuchungen über das logische Schließen”, Mathematische Zeitschrift, vol. 39, pp. 176–210, 1969

[20] J. K. J. Hintikka, “Form and content in quantification theory”, Acta Philosophica Fenicca, vol. II, 1955.

[21] S. Kanger, “Provability in logic”, Acta Universitatis Stockholmiensis. Stockholm Studies in Philosophy, Stockholm: Almqvist & Wiksell, vol. 1, 1957.

[22] S. C. Kleene, “Mathematical Logic”, John Wiley and Sons Inc, 1967.

[23] K. Shatte, “Proof Theory”, Springer-Verlag Berlin Heidelberg, 1977.

[24] R. M. Smullyan, “First Order Logic”, Springer-Verlag. Berlin Heidelberg 1969.

[25] M. Mondadori, “An improvement if Jeffrey’s deductive trees”, Annali dell’Universitá di Ferrara, Nuova Serie, sessione III, Filosofia, discussion paper, no. 7, Universitá degli Studi di Ferrara, 1989.

[26] M. D’Agostino, “Investigation into the complexity of some propositional calculi”, D. Phil Dissertation, Programming Research Group, Oxford University, 1990.

[27] Martin Richards, “A Tautology Checker loosely related to Stalmarck’s Algorithm”, A Seminar given at Cambridge, 1998. [Online]. Available: http://www.cl.cam.ac.uk/users/mr/chkslds.pdf.

[28] Project SYRF, “Use of the Stalmarck Method 1”, 1998. [Online]. Available:http://www-verimag.imag.fr/PROJECTS/SYNCHRONE/SYRF/syrf.html.

[29] Jakob Nordstram, “Stalmarck’s Method versus Resolution: A Comparative Theoretical Study”, Master’s thesis, Stockholm University, 2001.

[30] SAT Benchmarks Instances. 2002. [Online]. Available: http://www.satcompetition.org/2002/submittedbenchs.html

Flávio Márcio de Morais e Silva possui graduação em Ciência da Computação e Mestrado em Informática pela Pontifícia Universidade Católica de Minas Gerais. Tem experiência profissional e atua como professor na área de Ciência da Computação com ênfase em sistemas de informação, banco de dados, engenharia de software e datawarehouse. Atualmente é professor na Universidade de Itaúna.

Julio Cesar Vale Neves possui graduação em Processamento de Dados pela UNA (2001), pós graduação em Engenharia de Software pela PUC-MG (2003), pós graduação em Gestão Estratégica da Informação pela UFMG (2004), mestrado em Ciência da Computação pela PUC-MG (2009), MBA em Finanças pela UNA (2012), e atualmente, aluno bolsista de doutorado em Ciência da Computação na PUC-MG. Atualmente é professor no Instituto de Gestão em Tecnologia da Informação (IGTI) e atua como Gerente Global de TI com foco em Infraestrutura, Service Desk e Sistemas, com atuação no Brasil e com reporte no exterior em uma multinacional americana.

Sérgio Mariano Dias é doutor em Ciência da Computação pela Universidade Federal de Minas Gerais (UFMG) (2016), Mestre em Ciência da Computação pela UFMG (2010) e Bacharel em Ciência da Computação pela Pontifícia Universidade Católica de Minas Gerais (2007). Atualmente, é analista Sênior no Serviço Federal de Processamento de Dados (SERPRO). Desenvolve pesquisas em análise formal de conceitos; ciência de dados; mineração de dados; extração e representação de conhecimento e aprendizado de máquinas. No SERPRO atua na Superintendência de Suporte e Dados (SUPSD) em ciência de dados.

Luis Enrique Zárate é doutor e mestre pela Universidade Federal de Minas Gerais (UFMG) (1992 e 1998 - respectivamente). Graduado em Engenharia Elétrica pela URP no Peru em 1981. Desde 1992 tem trabalhado com pesquisas e lecionando diversas disciplinas na Pontifícia Universidade Católica de Minas Gerais. Professor Zárate desenvolve pesquisas em Data Mining e Big Data, Análise Formal de Conceitos e Soft Computing. Atua como coordenador do Laboratório de Inteligência Computacinald da PUC-MG (LICAP) e é responsável por diversas pesquisas

que são suportadas por organizações governamnetais no Brasil.

Mark Alan Junho Song possui o bacharelado, mestrado e doutorado pela Universidade Federal de Minas Gerais, em 1991, 1996 e 2004, respectivamente. Desde 1993 é pesquisador e professor da Pontifícia Universidade Católica de Minas Gerais. Seu interesse de pesquisa inclui principalmente verificação formal, verificação de modelos, teste de software e análise de conceito formal.