Satisfação booleana: algoritmos, aplicações, implementações [*]

Iouliia Skliarova, António B. Ferrari

***Resumo* – Neste artigo considera-se em detalhe o problema de *satisfação booleana* (SAT) e descrevem-se os algoritmos discretos completos que são normalmente utilizados para a sua solução. É mostrado que SAT tem inúmeras aplicações práticas. Portanto, o desenvolvimento e a implementação de algoritmos eficientes assumem actualmente grande importância. Finalmente, são analizadas várias realizações de algoritmos baseadas em hardware reconfigurável e é comparado o seu desempenho.**

***Abstract* – This paper presents the detailed description of the Boolean satisfiability (SAT) problem and considers the complete discrete algorithms that are commonly employed in its solution. It is demonstrated that SAT has numerous practical applications. Thus the design and implementation of efficient algorithms is of great importance today. Finally, various realizations of SAT solvers based on reconfigurable hardware are analyzed and compared.**

I. INTRODUCÃO

O problema de *satisfação booleana* consiste em determinar se uma função booleana é satisfazível, i.e. se existe pelo menos uma atribuição de valores às variáveis que faz com que a função tome valor *1*. Normalmente a função é especificada em CNF (*Conjunctive Normal Form*), isto é como a conjunção de um número de cláusulas, sendo cada cláusula composta por uma disjunção de uns literais. Cada literal corresponde a uma variável ou à sua negação.

Por exemplo, a função (1) contém três variáveis e quatro cláusulas e é satisfeita quando $x_1=1$, $x_2=0$, $x_3=1$:

$$(x_1 \vee x_2)(\overline{x}_1 \vee x_3)(\overline{x}_2 \vee \overline{x}_3)(x_1 \vee \overline{x}_2) \qquad (1)$$

De outro lado a função seguinte é insatisfazível:

$$(x_1 \vee \overline{x}_2)(x_2)(\overline{x}_1)$$

O problema de SAT pertence à classe de problemas NP-completos [1] o que significa que até agora não foi descoberto nenhum algoritmo que tenha execução rápida nas piores condições, e é pouco provável que este exista. Contudo, alguns métodos podem resolver eficientemente muitas instâncias de SAT encontradas nas aplicações práticas. Portanto, dá-se bastante atenção ao desenvolvimento de tais algoritmos eficientes.

O SAT é um caso específico dos problemas de satisfação de restrições (*Constraint Satisfaction Problems*) [2] que consistem em determinar se um conjunto de restrições sobre variáveis discretas pode ser satisfeito (aqui cada restrição corresponde a uma cláusula).

Os algoritmos de solução de SAT podem ser divididos em várias classes segundo as três características seguintes [2]. Primeiro, o SAT é formulado sobre variáveis discretas, contudo alguns algoritmos (por exemplo, a programação linear) realizam os cálculos sobre variáveis contínuas. Como resultado são formadas duas classes de métodos: os discretos e os contínuos. Segundo, todos os algoritmos são divididos em restringidos (os que satisfazem sempre todas as restrições caso isto seja possível) e não restringidos (os que apenas minimizam o número de restrições não satisfeitas). E, terceiro, os métodos podem ser sequenciais ou paralelos. Sendo assim, obtemos 8 classes diferentes de algoritmos: sequenciais discretos restringidos, paralelos discretos restringidos, sequenciais contínuos restringidos, etc. É de notar que existem outras classificações. Por exemplo, os algoritmos podem ser divididos em completos e incompletos. Os completos são capazes de encontrar sempre a solução ou garantir que esta não existe.

O resto deste artigo está organizado da maneira seguinte. Na secção II são considerados os algoritmos discretos mais conhecidos de solução de SAT. Na secção III descrevem-se algumas aplicações práticas. A seguir, na secção IV, analizam-se várias implementações de algoritmos baseadas em hardware reconfigurável. Finalmente, as conclusões estão na secção V.

II. ALGORITMOS

Neste artigo consideramos apenas os algoritmos discretos restringidos. Um dos métodos eficientes de solução de SAT consiste na substituição recursiva da função original por uma ou duas sub-funções de tal maneira que a atribuição de valores às variáveis que satisfaz uma das sub-funções, satisfaça também a função original. Portanto logo que é encontrada a solução para uma das sub-funções, a pesquisa termina. A decomposição e a resolução são os dois métodos mais conhecidos para gerar sub-funções.

A. Resolução

A resolvente de duas cláusulas $c_1=(x_1 \vee ... \vee v_i \vee ... \vee x_{l1})$ e $c_2=(y_1 \vee ... \vee \overline{v}_i \vee ... \vee y_{l2})$ é a cláusula $(x_1 \vee ... \vee x_{l1} \vee y_1 \vee ... \vee$

[*] Trabalho financiado com a bolsa da FCT-PRAXIS XXI/BD/21353/99

y_{l2}). Na resolução é escolhida uma variável v_i, $1 \geq i \leq n$, e as resolventes que é possível criar com a ajuda da v_i são adicionadas à função original. Este processo continua até que seja criada uma claúsula vazia (isto significa que a função original é insatisfazível) ou até que seja impossível gerar mais resolventes (o que quer dizer que a solução existe). Por exemplo, ao resolver a função (1) com a ajuda da variável x_1 obtemos a função seguinte:

$$(x_1 \vee x_2)(\bar{x}_1 \vee x_3)(\bar{x}_2 \vee \bar{x}_3)(x_1 \vee \bar{x}_2)(x_2 \vee x_3)(\bar{x}_2 \vee x_3)$$

Na resolução, a escolha da variável v_i influencia significativamente a rapidez da obtenção da solução. Consideremos algumas das técnicas que permitem acelerar a resolução do problema.

A.1. Absorção

Quando os literais numa das cláusuas são um subconjunto dos literals de outra cláusula, então a cláusula menor absorve a maior. Consequentemente, a cláusula maior pode ser apagada da função não modificando deste modo o conjunto de soluções possíveis. Isto deve-se ao facto de que cada atribuição de valores às variáveis que satisfaz a cláusula menor, irá também satisfazer a cláusula maior. Esta técnica é de grande importância na resolução porque ajuda a eliminar cláusulas grandes facilitando deste modo a pesquisa.

A.2. Resolução de Davis-Putnam

Ao formar todas as resolventes de uma variável v_i é possível apagar todas as cláusulas que contêm v_i ou $\bar{v}_i$. Sendo assim, primeiro são geradas todas as resolventes para a primeira variável, depois para a segunda, etc. No final só fica uma variável, portanto torna-se claro qual o valor que esta deve tomar para satisfazer a função. Este valor incorpora-se na sub-função obtida no passo anterior. Como resultado obtemos novamente a função composta só por uma variável. Este processo continua até que todas as variáveis recebam um valor. Por exemplo, para a função (1) para a variável x_1 obtemos a sub-função seguinte:

$$(\bar{x}_2 \vee \bar{x}_3)(x_2 \vee x_3)(\bar{x}_2 \vee x_3) \qquad (2)$$

A seguir, para a variável x_2 é gerada a subfunção:

$$(x_3)$$

Portanto, é evidente que x_3 tem de ser igual a *1* para satisfazer a função. Ao incorporar este valor na sub-função (2) recebemos:

$$(\bar{x}_2)$$

Obviamente x_2 tem que ser igual a *0*. Ao incluir este valor na função (1) recebemos a sub-função:

$$(x_1)$$

Torna-se claro que x_1 tem de receber valor *1* para satisfazer a função.

A.3. Literais puros

Um literal diz-se puro quando todas as suas ocorrências são positivas ou, ao contrário, negativas. Obviamente, é impossível gerar resolventes para os literais puros. Contudo todas as cláusulas que contêm literais puros podem ser eliminadas da função não alterando deste modo o conjunto de soluções. À variável correspondente é atribuído o valor *0* (se o literal puro é negativo) ou o valor *1* (quando o literal puro é positivo).

B. Decomposição

Na decomposição, é escolhida uma variável x_i, $1 \geq i \leq n$, cujo valor ainda não está determinado, e são formadas duas sub-funções, na primeira das quais $x_i = 1$ e na segunda $x_i = 0$. Como resultado cada sub-função contém todas as cláusulas da função original para além daquelas que tomaram o valor *1*, e todos os literais da função original para além daqueles que tomaram o valor *0*. Portanto nenhuma das sub-funções contém a variável x_i e a função original só é solúvel quando uma das sub-funções o é. Consideremos um exemplo.

Dada a função (1) supomos que foi escolhida a váriavel x_1. Recebemos assim duas sub-funções a primeira das quais é satisfazível e a segunda não:

- para $x_1=1$: $(x_3)(\bar{x}_2 \vee \bar{x}_3)$;
- para $x_1=0$: $(x_2)(\bar{x}_2 \vee \bar{x}_3)(\bar{x}_2)$.

As sub-funções geradas na decomposição são analisadas uma por uma. Se a análise da primeira sub-função possibilitar o encontro da solução, o processo de pesquisa para. Caso contrário o algoritmo retrocede um passo atrás e tenta resolver a segunda sub-função. Portanto os algoritmos baseados na decomposição são designados algoritmos de retrocesso (*backtracking algorithms*). A ordem da análise de situações diferentes em algoritmos deste tipo é normalmente representada em forma de uma árvore de pesquisa. A raíz da árvore corresponde à função original, os vértices interiores coincidem com várias sub-funções geradas no processo da pesquisa, e as folhas representam aquelas sub-funções que podem ser resolvidas directamente, não recorrendo à decomposição. Todos os vértices da árvore de pesquisa são interligados por arcos etiquetados com as respectivas atribuições de valores às variáveis. Por exemplo, para a função (1), supondo que foi escolhida a ordem de variáveis de decomposição x_1-x_2-x_3, obteremos a árvore de pesquisa apresentada na fig. 1. Devido ao facto de só estarmos interessados numa solução, não é necessário percorrer a parte restante da árvore de pesquisa.

Os vários algoritmos de retrocesso distinguem-se pela maneira como é seleccionada a variável à qual são atribuídos os dois valores (esta variável designa-se *variável de decisão*). Consideremos alguns dos mais conhecidos destes algoritmos.

B.1. Retrocesso simples

Neste caso para seleccionar a variável de decisão escolhe-se a primeira variável x_i com o valor indeterminado. A seguir são geradas duas sub-funções numa das quais $x_i=1$ e na outra $x_i=0$. Cada uma das sub-funções é processada da maneira recursiva. Se uma sub-função tiver uma cláusula vazia, o processamento desta sub-função deve ser suspenso, porque será impossível encontrar uma solução nesta parte da árvore de pesquisa (situações 4 e 5 na fig. 1). Consequentemente, é necessário retroceder até ao último ponto de ramificação no qual o exame da variável de decisão não foi concluído, inverter o valor desta variável e continuar a percorrer a árvore de pesquisa. O facto de não existirem mais variáveis com valor indeterminado indica que a solução foi encontrada (situação 7 na fig. 1). A tentativa de retroceder para além da variável x_1 significa que todas as atribuições possíveis de valores às variáveis foram examinadas e que, consequentemente, a função original é insatisfazível.

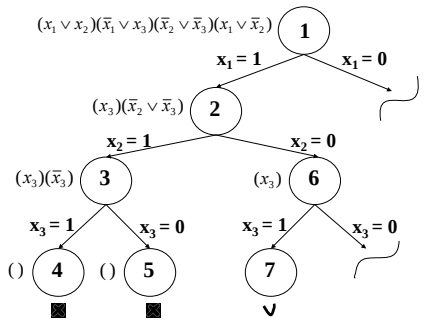

Fig. 1 – Árvore de pesquisa para a função (1)

B.2. Retrocesso de cláusula unitária

Caso uma cláusula c_j, $1 \geq j \leq m$, só contenha uma variável, então é esta variável que é seleccionada como a de decisão e é-lhe atribuído um valor *k* tal que satisfaça a cláusula c_j (diz-se que a variável é *implicada* pelo valor *k*). As cláusulas com estas características são chamadas *cláusulas unitárias*. Quando não existem cláusulas unitárias, é seleccionada a primeira variável com o valor indeterminado (como no retrocesso simples). Esta técnica permite acelerar significativamente a pesquisa porque tenta satisfazer primeiro aquelas cláusulas que são mais "dificeis".

B.3. Retrocesso pela sequência de cláusulas

Neste caso para seleccionar a variável de decisão escolhe-se a primeira cláusula e dentro desta é eleita a primeira variável. Como resultado, o algoritmo às vezes evita a atribuição de valores a todas as variáveis, e a solução fica apresentada de uma forma compacta. Isto deve-se ao facto de serem atribuídos valores apenas àquelas variáveis que influenciam a satisfação de cláusulas, todas as outras variáveis ficam com o valor "-" (*don't care*).

B.4. Retrocesso de cláusula mais curta

Este algoritmo é idêntico ao anterior mas as cláusulas são escolhidas não pela sua ordem relativa na função, mas sim pela quantidade de literais que estas contêm (sempre se selecciona a cláusula mais curta). Como resultado, primeiro são satisfeitas as cláusulas mais "dificeis".

C. Algoritmo de Davis-Putnam

O algoritmo de Davis-Putnam [3, 4] considera-se clássico pois foi proposto nos anos sessenta e desde aí tornou-se muito conhecido e utilizado. Baseia-se em ambas as técnicas consideradas, i.e. na resolução e na decomposição. O algoritmo começa com os valores de todas as variáveis indeterminados. Inicialmente, selecciona-se a primeira variável x_i com o valor indeterminado e x_i torna-se igual a *0*. A seguir são descobertas todas as implicações (ver secção B.2) directas e transitivas em outras variáveis. Caso alguma variável seja implicada a dois valores opostos, estamos perante um conflito. Se não surgir conflito nenhum, selecciona-se novamente uma variável com o valor indeterminado e o processo continua (este chama-se *pesquisa para diante*). Caso apareça um conflito, o algoritmo retrocede até à variável de decisão mais recente e inverte o seu valor. Se novamente surgir um conflito, a variável de decisão recebe o valor indeterminado e o algoritmo retrocede mais uma vez. A tentativa de retroceder para além da primeira variável indica que todas as atribuições possíveis de valores às variáveis foram esgotadas e que, consequentemente, a função é insatisfazível. A solução é encontrada quando todas as variáveis recebem um valor sem conflitos.

III. APLICAÇÕES PRÁTICAS

O SAT tem muitas aplicações práticas em síntese lógica [5, 6], colocação e encaminhamento de circuitos electrónicos [7, 8], teste de circuitos [9, 10], planeamento [11, 12], telecomunicações [13], matemática [14], robótica [15], processamento de texto [16], arquitectura de computadores [17], etc. [2, 18]. Portanto, o desenvolvimento de métodos eficientes, bem como a sua implementação, tornou-se uma área vital de investigação. Consideremos em detalhe duas aplicações que podem ser formuladas como instâncias de SAT.

A. Planeamento

O problema de planeamento surge em muitas aplicações práticas. Suponhamos que existe um conjunto de investigadores que devem resolver uma série de tarefas. Cada cientista só pode lidar com um sub-conjunto de tarefas. Obviamente, o objectivo é maximizar a eficiência

do trabalho dos investigadores. Suponhamos que a execução de cada tarefa ocupa tempo p_i. A solução do problema é um plano que define o tempo de início s_i de cada tarefa. São normalmente aplicadas as seguintes restrições [11]:

- Restrições de sequência $i \rightarrow j$ definem que a tarefa *i* deve ser concluída antes do início da tarefa *j*. Esta restrição pode ser reformulada como $s_i + p_i \leq s_j$, i.e. o tempo de início da tarefa *i* mais o tempo da sua execução devem ser iguais ou menores que o tempo do início da tarefa *j*.
- Restrições de recursos $c_{i,j}$ indicam que as tarefas *i* e *j* estão em conflito, i.e. ambas requerem o mesmo recurso (o mesmo investigador). Portanto as tarefas *i* e *j* não podem ser executadas ao mesmo tempo. A outra forma de especificar esta restrição é $(s_i + p_i \leq s_j) \vee (s_j + p_j \leq s_i)$, i.e. a tarefa *i* deve ser concluída antes do início da tarefa *j*, ou, ao contrário, a tarefa *j* deve ser concluída antes do início da tarefa *i*.
- Restrições de início r_i definem o tempo depois do qual a tarefa *i* pode ser iniciada, i.e. $s_i > r_i$.
- Restrições de prazo d_i especificam o momento até ao qual a execução da tarefa *i* deve ser finalizada, i.e. $s_t + p_t \leq d_t$.

Existe uma transformação óbvia do problema de planeamento em SAT [11]. Para tal criam-se as variáveis s_{it} que representam os tempos de início *t* de cada tarefa *i* ($s_{it}=1$ significa que a tarefa *i* começa em tempo *t*). A seguir são criadas as cláusulas que representam todas as desigualdades necessárias. É de notar que o espaço de pesquisa resultante será neste caso demasiado grande, portanto são utilizadas só as transformações que permitem ordenar as tarefas que estão em conflito porque os tempos de início de cada tarefa podem ser determinados posteriormente [11]. Não vamos abordar estas transformações aqui pois são bastante complexas. Contudo, é importante realçar que a função resultante embora seja grande, é facilmente resolvida. Isto acontece porque esta é sub-restringida, i.e. existem poucas restrições, portanto há muitas soluções e é fácil encontrar uma.

B. Teste de circuitos combinatórios

O SAT pode ser também utilizado para gerar padrões de teste para descobrir falhas permanentes (*stuck-at faults*) em circuitos combinatórios [9]. Para detectar falhas num circuito são aplicados a este valores de entrada que produzem valores de saída diferentes dos do circuito sem falhas. Neste caso, para descobrir os valores de entrada, primeiro é construída uma função que define o conjunto de padrões de teste capazes de determinar uma falha e, a seguir, são encontrados os valores das variáveis que satisfazem a função.

Para tal primeiro a estrutura do circuito combinatório é descrita com a ajuda de uma função booleana em CNF. Por exemplo, para a porta lógica *OR* com as entradas *x* e *y* e a saída *z* temos:

$$z = x \vee y$$

Tendo em conta que a função p=q é equivalente a $(p \rightarrow q)(q \rightarrow p)$, obtemos:

$$(z \rightarrow (x \vee y)) \, ((x \vee y) \rightarrow z)$$

Como a função $p \rightarrow q$ é equivalente a $\bar{p} \vee q$ pode-se escrever:

$$(\bar{z} \vee x \vee y)(\overline{(x \vee y)} \vee z)$$

Finalmente, applicando os leis de De Morgan obtemos:

$$(\bar{z} \vee x \vee y)(z \vee \bar{x})(z \vee \bar{y})$$

Da mesma maneira pode-se determinar as funções para todos os tipos de portas lógicas e, a seguir, construir a função do circuito inteiro. Por exemplo, para o circuito da fig. 2 obtemos a função seguinte:

$$(\bar{f} \vee \bar{c} \vee \bar{e})(f \vee c)(f \vee e)(\bar{c} \vee a \vee b)(c \vee \bar{a})(c \vee \bar{b})(d \vee e)(\bar{d} \vee \bar{e}) \quad (3)$$

Na fig. 3 está representada a versão defeituosa do mesmo circuito (a linha *e* está *stuck-at 1*). A função do circuito com falha constroi-se da mesma maneira, mas as linhas que ficam entre a falha e a saída devem ser renomeadas (ver fig. 3):

$$(\bar{f}' \vee \bar{c} \vee \bar{e}')(f' \vee c)(f' \vee e')(e')(\bar{c} \vee a \vee b)(c \vee \bar{a})(c \vee \bar{b}) \quad (4)$$

Para poder detectar esta falha será necessário encontrar os valores de entrada que produzem valores de saída diferentes para os dois circuitos considerados (um com falha e outro sem falha). Para tal construímos a conjunção de funções (3) e (4) e adicionamos a disjunção exclusiva das saídas de ambos os circuitos (ver fig. 4):

$$\begin{aligned}&(\bar{f} \vee \bar{c} \vee \bar{e})(f \vee c)(f \vee e)(\bar{c} \vee a \vee b)(c \vee \bar{a})(c \vee \bar{b}) \wedge \\ &\wedge (d \vee e)(\bar{d} \vee \bar{e})(\bar{f}' \vee \bar{c} \vee \bar{e}')(f' \vee c)(f' \vee e')(e') \wedge \\ &\wedge (\bar{f} \vee f' \vee g)(f \vee \bar{f}' \vee g)(\bar{f} \vee \bar{f}' \vee \bar{g})(f \vee f' \vee \bar{g})\end{aligned} \quad (5)$$

Para poder descobrir a falha, a saída *g* deve ser igual a *1*. Assim obtemos a função:

$$\begin{aligned}&(\bar{f} \vee \bar{c} \vee \bar{e})(f \vee c)(f \vee e)(\bar{c} \vee a \vee b)(c \vee \bar{a})(c \vee \bar{b}) \wedge \\ &\wedge (\bar{f}' \vee \bar{c} \vee \bar{e}')(f' \vee c)(f' \vee e')(e')(d \vee e)(\bar{d} \vee \bar{e}) \wedge \\ &\wedge (\bar{f} \vee \bar{f}')(f \vee f')\end{aligned} \quad (6)$$

A atribuição de valores às variáveis que satisfaz a função (6) define os padrões de teste necessários para detectar a falha considerada (por exemplo, *a=b=d=1*).

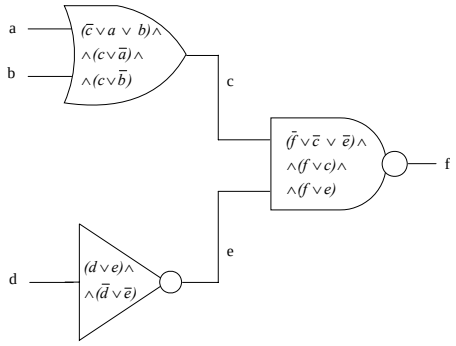

Fig. 2 – Circuito combinatório

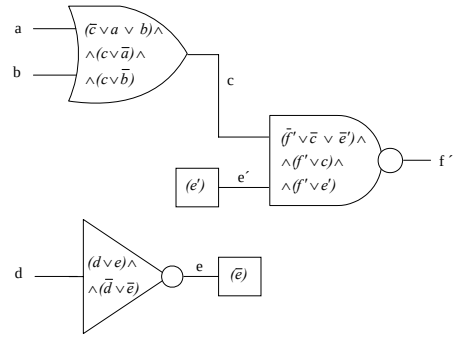

Fig. 3 – Circuito combinatório com falha

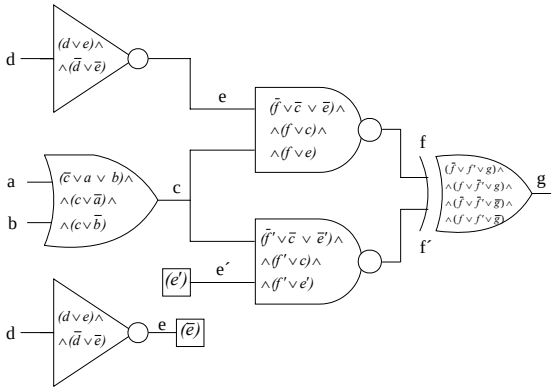

Fig. 4 – Combinação dos circuitos defeituoso e correcto

IV. IMPLEMENTAÇÕES BASEADAS EM HARDWARE RECONFIGURÁVEL

Durante os últimos 5 anos têm sido efectuadas várias tentativas de acelerar a solução de SAT com a ajuda de hardware reconfigurável [19-37]. Como já foi mencionado, SAT é um problema NP-completo; portanto, o tempo de solução de muitas instâncias depende expoencialmente do tamanho da função em causa. Em consequência, às vezes é praticamente impossível resolver algumas instâncias com a ajuda de computadores convencionais. Contudo, em dispositivos reconfiguráveis pode-se criar unidades funcionais optimizadas para certos tipos de operações, executar estas operações em paralelo, e personalizar a organização da memória para os tamanhos de dados necessários. Obviamente, tudo isto não elimina o efeito do tempo expoencial da solução do problema, mas permite minorá-lo, possibilitando deste modo o processamento de funções maiores [2].

Descrevemos algumas das implementações mais conhecidas (as secções subsequentes são nomeadas de acordo com o nome do primeiro autor da ideia respetiva). As características principais de todas as realizações consideradas são sumariadas na tabela 1.

A. Suyama et al.

Suyama et al. [19-21] propuseram um algoritmo que é capaz de encontrar todas as soluções (ou um número fixo destas) de uma instância. Só foi abordado o problema de 3-SAT no qual cada cláusula contém no máximo 3 literais (é de notar que qualquer instância pode ser transformada em 3-SAT através da introdução de variáveis auxiliares).

Os autores começaram por um exame exaustivo de todos os valores possíveis de todas as variáveis. É fácil realizar esta ideia com a ajuda de um contador de *n* bits (*n* é o número de variáveis da função). Contudo, este algoritmo é pouco eficiente pois devem ser examinados 2^n vectores diferentes. Portanto, Suyama et al. implementaram algumas extensões que incluem a resolução unitária e o retrocesso pela sequência de cláusulas. Isto permitiu diminuir significativamente o número de vectores examinados.

Para escolher a variável de decisão seguinte os autores aplicaram a técnica de ordenação dinâmica, i.e. as variáveis são seleccionadas não pela sua ordem relativa na função (esta é a seleção estática), mas sim com a ajuda de métodos heurísticos especiais. Foram considerados dois métodos deste tipo. O primeiro chama-se *propagação unitária experimental* e é baseado na atribuição experimental de valores *0* e *1* a cada variável livre (i.e. com o valor indeterminado). A seguir é seleccionada a variável que gera o número máximo de cláusulas unitárias. O segundo método chama-se a *ocorrência máxima em cláusulas binárias* (i.e. em cláusulas que só contêm dois literais). Neste caso, como o próprio nome indica, é seleccionada a variável que ocorre num número máximo de cláusulas binárias. Em hardware só foi implementada a primeira destas heurísticas. O desempenho do algoritmo proposto é equivalente ao do de Davis-Putnam [3].

A implementação utiliza a abordagem optimizada para cada instância do problema (i.e. para cada instância é gerado um circuito específico e optimizado [22]) e é baseada numa máquina de estados finitos. Alguns circuitos foram realizados em FLEX10K250 FPGA da Altera e funcionaram à frequência de 10 MHz, e os outros só foram simulados. Sendo assim foi possível resolver algumas instâncias de SAT disponíveis da DIMACS [38] e conseguir alguma aceleração em comparação com o algoritmo POSIT [39] realizado em software e executado em UltraSPARC-II/296 MHz. Contudo, é de notar que:

- O tempo de geração do circuito não foi considerado.

- Não foi proposta nenhuma solução eficiente para o caso da função exceder significativamente os recursos de hardware disponíveis.

B. Zhong et al.

Zhong et al. [23, 24] implementaram a versão do algoritmo de Davis-Putnam [3]. Para cada variável na função foi construído um circuito de implicação e uma máquina de estados finitos (FSM). A seguir, todas as FSMs foram ligadas numa cadeia série. Em cada instante de tempo só uma FSM está activa. Logo que esta acabe o processamento, o controlo é transferido quer à FSM à sua direita (pesquisa para diante) quer à sua esquerda (retrocesso). Portanto, para atribuir valores às variáveis o circuito usa o controlo distribuído. Cada FSM guarda o valor corrente da sua variável (que pode ser *0, 1*, or indeterminado) e conhece se este valor foi atribuído ou implicado. O facto da última FSM tentar activar a FSM seguinte indica que a solução foi encontrada. A solução não existe quando a primeira FSM pretende passar o controlo à esquerda.

Inicialmente, todas as variáveis são ordenadas de acordo com o número das suas ocorrências em cláusulas. Esta ordem estática é utilizada para arranjar as variáveis na cadeia série. Para cada variável de decisão primeiro é examinado o valor *1* e depois o valor *0*. Foi também proposta a técnica de adição dinâmica de cláusulas [25] e a implementação do mecanismo do retrocesso não cronológico [23]. O retrocesso é chamado não cronológico quando o algoritmo é capaz de retroceder mais do que um nível na árvore de pesquisa. Isto permite reduzir significativamente o tempo de solução do problema.

A implementação utiliza também a abordagem optimizada para cada instância do problema e o tempo de geração do circuito é da ordem das dezenas de minutos mesmo que sejam aplicados os métodos especiais de aceleração [26]. A fim de ultrapassar esta restrição os autores propuseram só recorrer ao hardware reconfigurável quando a função é de grande dimensão, enquanto para as funções "pequenas" podem ser utilizadas aplicações de software apropriadas [23, 25].

Vários circuitos foram implementados com base no emulador IKOS e na placa Pamette [24] e funcionaram a frequências de 700 KHz a 2 MHz [23]. A aceleração média em comparação com o GRASP [40] (que é um dos mais eficientes algoritmos de SAT implementados em software) executado em Sun5/110 MHz/64 MB foi de 64x para um subconjunto de instâncias da DIMACS [38] (não incluindo o tempo de geração do circuito e o tempo de configuração). Quando o tempo de geração do circuito foi também considerado, os autores conseguiram uma aceleração de 8x para algumas instâncias difíceis [24].

C. Platzner et al.

A implementação proposta por Platzner et al. [27-30] é semelhante à de Zhong [23]. O circuito contém uma coluna de FSMs, a lógica de dedução e uma unidade de controlo central. A lógica de dedução calcula o resultado da função com base em valores correntes de variáveis. As variáveis são seleccionadas da maneira estática e cada variável de decisão primeiro recebe o valor *0* e depois *1*. Inicialmente, os valores de todas as variáveis são indeterminados e a unidade de controlo activa a primeira FSM. A FSM activa atribui à sua variável o valor *0* e, a seguir, a lógica de dedução calcula o resultado da função. Se a função se tornar igual a *1* o processo termina, pois a solução foi encontrada. Caso contrário, se a função receber o valor *0*, a FSM activa inverte o valor da sua variável. Se o valor da função ainda não pode ser determinado, é activada a FSM seguinte. Se uma FSM examinar os dois valores possíveis da sua variável e o resultado da função ficar sempre igual a *0*, então o controlo é transferido à FSM anterior.

Os autores implementaram alguns circuitos com base na placa Pamette que contém 4 FPGAs XC4028 da Xilinx e a frequência de relógio foi de 20 MHz. A aceleração conseguida para instâncias *hole6-hole10* da DIMACS [38] em comparação com o GRASP [40] (executado em PII/300MHz/128MB) é de 0.003 a 7.408 vezes (aqui o tempo de geração e configuração do hardware está incluído).

Em [28] foram propostas duas extensões à arquitectura básica. Na primeira extensão são introduzidas cláusulas adicionais que identificam as variáveis que não influenciam a solução. Isto permite eliminar decisões sobre estas variáveis. Na segunda extensão é explorada a resolução unitária (da mesma maneira como no algoritmo de Davis-Putnam [3]). Os autores acreditam também que o hardware reconfigurável só deve ser utilizado para instâncias dificeis que requerem muito tempo enquanto resolvidas em software, enquanto para problemas simples uma aplicação de software pode ser mais eficiente. Se a aplicação de software não conseguir resolver uma função durante um período de tempo predeterminado, então a tarefa pode ser transferida para a FPGA.

D. Abramovici et al.

Abramovici et al. [31, 32] aplicaram a técnica de modelação da função por um circuito com dois níveis. A implementação é baseada no algoritmo PODEM que é normalmente utilizado para resolver problemas de geração de padrões de teste. Um conceito importante deste algoritmo é o objectivo que é a atribuição desejável de um valor a uma linha do circuito (inicialmente os valores de todas as linhas são indeterminados). Um objectivo só pode ser atingido através da atribuição de valores às entradas primárias. O procedimento de recuo (*backtrace*) é utilizado para propagar todos os objectivos em paralelo ao longo de todos os caminhos possíveis até chegar às

entradas primárias, e a seguir, determinar os valores das entradas que são capazes de ajudar mais a atingir os objectivos. Este procedimento permite atribuir às variáveis os valores certos.

Para implementar este método em FPGA os autores sugerem criar um biblioteca de módulos básicos que podem ser utilizados para qualquer função. Os módulos terão o encaminhamento e a colocação internos predefinidos. Sendo assim, o circuito continua a ser específico a cada instância mas este pode ser construído dos módulos, o que permite reduzir significativamente o tempo de compilação de hardware (até à ordem de alguns minutos). Os autores implementaram vários circuitos simples em FPGA XC6264 e simularam os mais complexos. Para um circuito que ocupa toda a área da XC6264 a frequência de relógio é de 3.5 MHz. A aceleração conseguida em comparação com o GRASP [40] executado em Sun UltraSparc/248MHz/1026MB foi de 0.01 a 7000 vezes para um subconjunto de instâncias da DIMACS [38]. Contudo, o tempo de compilação e configuração de hardware não foi considerado.

Em [32] foi proposto o sistema de lógica virtual que permite construir circuitos capazes de resolver funções que excedem os recursos de hardware disponíveis. Isto é atingido através da decomposição da função em sub-funções independentes que podem ser processadas em FPGAs separadas quer em paralelo, quer sequencialmente. Deve-se notar contudo, que a eficiência da decomposição depende muito da função em causa. Portanto a decomposição de algumas funções pode tornar-se pouco prática pois aumenta o tempo de compilação até níveis inaceitáveis.

Devido ao facto do tempo de compilação do hardware continuar a ser bastante elevado, na investigação mais recente nesta direcção tenta-se evitá-lo de alguma maneira [22, 33-36].

E. Sousa et al.

Sousa et al. [34, 35] implementaram um algoritmo de pesquisa derivado do de Davis-Putnam composto por três fases que são a decisão, a dedução e o diagnóstico. Na primeira fase, a variável de decisão e o seu valor são seleccionados da maneira dinâmica. Na dedução todas as implicações unitárias são calculadas. A fase de diagnóstico só é activada quando surge um conflito. Esta consegue indentificar as variáveis de decisão que o causaram, possibilitando deste modo o retrocesso não cronológico. A análise de conflitos permite também construir e adicionar à função novas cláusulas que ajudam a acelerar o processo da pesquisa.

Sousa et al. foram os primeiros a propor dividir o trabalho entre o software e o hardware reconfigurável sendo a parte mais intensiva em termos computacionais (esta inclui o cálculo de implicações e a selecção da variável de decisão seguinte) atribuída ao hardware, enquanto as tarefas de controlo (tais como a análise de conflitos, o controlo do retrocesso, etc) são executadas em software. Neste caso o computador hospedeiro recebe e envia à FPGA as variáveis que mudaram de valor, e adiciona ou retira algumas cláusulas. Contudo, parece-nos que esta comunicação é bastante intensiva, podendo influenciar significativamente o tempo de solução do problema.

Antes de abordar um problema, os autores sugerem convertê-lo em 3-SAT. Para lidar com as instâncias que excedem os recursos de hardwre disponíveis, foi proposto um esquema de hardware virtual com comutação de contexto. Isto é efectuado através da divisão do circuito original em páginas de hardware que são executadas sequencialmente, sendo os resultados intermédios guardados numa RAM externa.

Os resultados publicados são baseados numa simulação em software sob a frequência estimada de 80 MHz e assumindo que é possível substituir páginas de hardware num ciclo de relógio. Para instâncias *hole9-hole10* da DIMACS [38] a aceleração conseguida em comparação com o GRASP (a plataforma de software não foi publicada) foi de 3 vezes. Contudo, não fica claro como foi estimado o tempo gasto em comunicação entre software e hardware.

F. Skliarova et al.

Skliarova et al. [22, 36] propuseram a arquitectura para solução de SAT que é orientada ao problema em geral e não só a uma instância. Como resultado, a compilação de hardware é totalmente evitada. Inicialmente, o problema é formulado sobre uma matriz ternária estabelecendo para tal a correspondência entre as cláusulas e as variáveis da função e as linhas e as colunas da matriz [22]. Para encontrar a solução usou-se o facto de que a satisfação de uma função é equivalente ao encontro de um vector ternário que seja ortogonal a cada linha da matriz respectiva. O algoritmo implementado é uma derivativa do de Davis-Putnam, i.e. é o de retrocesso que usa a resolução unitária, a resolução dos literais puros e a selecção dinâmica da variável de decisão. Na qualidade da variável de decisão escolhe-se uma que ocorre num número máximo de cláusulas.

Como já foi mencionado, a arquitectura é universal, portanto para resolver qualquer instância do problema só é necessário transferir a matriz para a FPGA e indicar as dimensões actuais da mesma (para que a unidade de controlo processe apenas a área adequada). O circuito foi implementado com base na placa com interface PCI ADM-XRC que contém uma FPGA XCV812E (a frequência de funcionamente é de 40 MHz).

A fim de poder resolver problemas com dimensões maiores do que os recursos da FPGA disponíveis, foi proposta uma técnica baseada na colaboração de software e hardware reconfigurável [41, 42]. Neste caso a FPGA só é responsável pelo processamento das sub-funções que aparecem em vários níveis da árvore de pesquisa e satisfazem as restrições impostas pelo circuito (tais como

o número máximo de linhas e colunas da matriz). Aquelas sub-funções que não satisfazem as restrições são processadas inicialmente em software. Os resultados das experiências com as instâncias *hole6-hole10* da DIMACS [38] demonstraram que é possível conseguir uma aceleração bastante significativa (até 111x, incluíndo o tempo de configuração da FPGA) comparando com o GRASP [40] executado num AMD Athlon/1GHz/256MB.

V. CONCLUSÕES

Neste artigo foram descritos os principais algoritmos completos discretos beseados na resolução e na decomposição. A seguir foram consideradas em detalhe duas aplicações práticas que requerem a solução de SAT. Finalmente, a parte principal do artigo é dedicada à descrição, análise e comparação de várias implementações

	O circuito é específico à instân-cia?	**È efectuada a selecção dinâmica de variáveis de decisão?**	**O valor à variável de decisão é atribuído de uma maneira dinâmica?**	**Foi proposta alguma solução eficiente para o caso da função exceder os recursos de hardware disponíveis?**	**Como lidar com o tempo significativo de geração do circuito para cada instância?**	**O problema é partilhado entre o software o hardware?**
(A) Suyama et al.	Sim	Sim	Sim	Não	Só usar hardware para funções dificeis	Não
(B) Zhong et al.	Sim	Não	Não	Não	Só usar hardware para funções dificeis	Não
(C) Platzner et al.	Sim	Não	Não	Não	Só usar hardware para funções dificeis	Não
(D) Abramovici et al.	Sim	Sim	Sim	Sim	Criar uma biblioteca de módulos pre-compilados	Não
(E) Sousa et al.	Sim	Sim	Sim	Sim	A implementação não requer o encaminhamento e o roteamento específicos à instância, portanto o tempo de compilação é bastante reduzido	Sim
(F) Skliarova et al.	Não	Sim	Não	Sim	Não aplicável pois a implementação não é orientada à instância	Sim

Tabela 1 – Comparação de várias implementações baseadas em hardware reconfigurável

de algoritmos baseadas em hardware reconfigurável. A complexidade do problema, a grande quantidade de aplicações práticas e o desenvolvimento intensivo que se tem observado ultimamente na área de dispositivos reconfiguráveis, tornaram tais implementações possíveis e razoáveis. A sua análise cuidadosa permite chegar às conclusões seguintes:

- A maioria de autores desenvolve algoritmos de pesquisa derivados do de Davis-Putnam.

- Quase todas as implementações usam a abordagem optimizada para cada instância do problema. Contudo, o tempo de compilação de hardware limita significativamente a gama de problemas para os quais a utilização da FPGA pode ser considerada razoável em comparação com a solução em software. Durante os últimos dois anos procura-se evitar o encaminhamento e o roteamento específicos à instância. Até agora só foi proposta uma arquitectura que é orientada ao problema em geral e não à instância.

- É bastante difícil comparar os resultados conseguidos. Primeiro, a maioria dos autores não publica os tempos de compilação e da configuração de hardware. Segundo, os investigadores comparam os seus próprios resultados com a aplicação de software GRASP, mas executam o GRASP em plataformas diferentes e usando opções diferentes. Como resultado, os tempos de solução

da mesma instância em software podem diferir em vários casos até ao nível de ordens de grandeza.

- Muitos autores evitam enfrentar a situação quando os recursos de hardware não são suficientes para realizar uma função. Contudo, as funções encontradas em aplicações práticas são normalmente de grandes dimensões. Ultimamente tem sido proposta e realizada a ideia de dividir de algum modo o problema entre software e hardware, o que parece ser a solução mais promissora e eficiente.

REFERÊNCIAS

[1] M.R.Garey, D.S.Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H.Freeman and Company, San Francisco, 1979.

[2] J.Gu et al., "Algorithms for the Satisfiability (SAT) Problem: A Survey", DIMACS Volume Series on Discrete Mathematics and Theoretical Computer Science: The Satisfiability (SAT) Problem, American Mathematical Society, 1996.

[3] M.Davis, H.Putnam, "A Computing Procedure for Quantification Theory", Journal of ACM, vol. 7, pp. 201-215, 1960.

[4] M.Davis, et al., "A Machine Program for Theorem Proving", Communications of the ACM, vol. 5, pp. 394-397, 1962.

[5] R.Brayton, et al., "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

[6] J.Gu, R.Puri, "Asynchronous circuit synthesis with boolean satisfiability", IEEE Trans. on CAD, vol. 14, no. 8, pp. 961--973, August 1995.

[7] S.Devadas, "Optimal Layout Via Boolean Satisfiability", IEEE Int. Conf. on CAD, 1989, pp.294-297.

[8] R.G.Wood, R.Rutenbar, "FPGA Routing and Routability Estimation via Boolean Satisfiability", IEEE Trans. on VLSI Systems, vol. 6, no. 2, pp. 222-231, 1998.

[9] T. Larabee, "Test Pattern Generation Using Boolean Satisfiability", IEEE Trans. on CAD, Vol. 11, No. 1, pp. 4 - 15, 1992.

[10] P.R.Stephan, et al., "Combinational Test Generation Using Satisfiability", IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 15, no. 9, pp. 1167-1176, Sept. 1996.

[11] J.M.Crawford, A.B.Baker, "Experimental result on the Application of Satisfiability Algorithms to Scheduling Problems", Proc. of the 12th AAAI, 1994, pp. 1092-1097.

[12] R.Feldman, M.C.Golumbic, "Optimization Algorithms for Student Scheduling via Constraint Satisfiability", Proc. of the Int. Joint Conf. on Artificial Intelligence, pp. 1010-1015.

[13] W.Wang, C.K.Rushforth, "An Adaptive Local Search Algorithm for Channel Assignment Problem", IEEE Trans. on Vehicular Technology, vol. 45, no. 3, pp. 459-466, Aug. 1996.

[14] D.W.Matula, et al., "Graph coloring algorithms", In Graph Theory and Computing. R.C.Read, editor, pp. 109-122. Academic Press, New York, 1972.

[15] R.T.Chin, C.R.Dyer, "Model-based Recognition in Robot Vision", ACM Computing Surveys, vol. 18, no. 1, pp. 67-108, March, 1986.

[16] X.Huang, et al., "A Constrained Approach to Multi Font Character Recognition", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 15, no.8, pp. 838-843, Aug.1993.

[17] C.D.V.P.Rao, N.N.Biswas, "On the Minimization of Wordwidth in the Control Memory of a Microprogrammed Digital Computer", IEEE Trans. on Computers, vol. 32, no. 9, pp. 863-868, September 1983.

[18] J.Gu, "Satisability Problems in VLSI Engineering", DIMACS Workshop on Satisability Problem, March 1996.

[19] T.Suyama et al., "Solving Satisfiability Problems Using Reconfigurable Computing", IEEE Trans. on VLSI Systems, vol. 9, no. 1, pp. 109-116, 2001.

[20] M.Yokoo et al., "Solving Satisfiability Problems Using Field Programmable Gate Arrays: First Results", Proc. of 2nd Int. Conf. on Principles and Practice of Constraint Programming, pp. 497-509, 1996.

[21] T.Suyama et al., "Solving Satisfiability Problems Using Logic Synthesis and Reconfigurable Hardware", Proc. of the 31st Annual Hawaii Int. Conf. on System Sciences, vol. VII, pp. 179-186, 1998.

[22] I.Skliarova, A.B.Ferrari, "Utilização de hardware reconfigurável para acelerar a satisfação booleana", Electrónica e Telecomunicações, Set., 2001, pp. 304-310.

[23] P.Zhong, et al., "Using reconfigurable computing techniques to accelerate problems in the CAD domain: a case study with Boolean satisfiability", Proc. of the Design Automation Conf., pp. 194-199, 1998.

[24] P.Zhong, et al., "Accelerating Boolean satisfiability with configurable hardware", Proc. 6th IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), pp. 186-195, 1998.

[25] P.Zhong et al., "Solving Boolean satisfiability with dynamic hardware configurations", in R. W. Hartenstein and A. Keevallik, editors, Field-Programmable Logic: From FPGAs to Computing Paradigm, pp. 326-235. Springer-Verlag, Berlin, Aug./Sept. 1998.

[26] P.K.Chan, et al., "Reducing compilation time of Zhong's FPGA-based SAT solver", in Proc. 6th IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), pp. 308-309, 1998.

[27] M.Platzner, "Reconfigurable accelerators for combinatorial problems", IEEE Computer, pp. 58-60, April 2000.

[28] M.Platzner, G. De Micheli, "Acceleration of satisfiability algorithms by reconfigurable hardware", in Reiner W. Hartenstein and Andres Keevallik, editors, Field-Programmable Logic: From FPGAs to Computing Paradigm, pp. 69-78. Springer-Verlag, Berlin, Aug./Sept. 1998.

[29] O.Mencer, M.Platzner, "Dynamic Circuit Generation for Boolean Satisfiability in an Object-Oriented Design Environment", Proc. of the 32nd Hawaii Int. Conf. on System Sciences, 1999.

[30] O.Mencer, et al., "Object-Oriented Domain Specific Compilers for Programming FPGAs", IEEE Trans. on VLSI Systems, vol. 9, no. 1, February 2001.

[31] M.Abramovici, et al., "A massively-parallel easily-scalable satisfiability solver using reconfigurable hardware", in Proc. Design Automation Conf., pp. 684-690, June 1999.

[32] M.Abramovici, J.T.de Sousa, "A SAT solver using reconfigurable hardware and virtual logic", Journal of Automated Reasoning, vol. 24, nos. 1-2, pp. 5-36, Feb. 2000.

[33] M.Boyd, T.Larrabee, "ELVIS – a scalable, loadable custom programmable logic device for solving Boolean satisfiability problems", in Proc. 8th IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), 2000.

[34] J. de Sousa, et al., "A configware/software approach to SAT solving", in 9th IEEE Proc. Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), May 2001.

[35] R.C.Ripado, J.T.de Sousa, "A Simulation Tools for a Pipelined SAT Solver", Proc. of the XVII Conf. on Design of Circuits and Integrated Systems – DCIS'2001, Portugal, Nov. 2001, pp. 498-503,.

[36] I.Skliarova, A.B.Ferrari, “A SAT Solver Using Software and Reconfigurable Hardware”, aceite para publicação nas actas de DATE’2002.

[37] P.H.W.Leong, et al., “A Bitstream Reconfigurable FPGA Implementation of the WSAT Algorithm”, IEEE Trans. on VLSI Systems, vol. 9, no. 1, February 2001, pp. 197-201.

[38] http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html

[39] J.W.Freeman, “Improvements to Propositional Satisfiability Search Algorithms”, Ph.D. dissertation, Univ. Pennsylvania, 1995.

[40] J.M.Silva, K.A.Sakallah, “GRASP: a search algorithm for prepositional satisfiability”, IEEE Trans. Computers, pp. 506-521, vol. 48, nº. 5, May 1999.

[41] I.Skliarova, A.B.Ferrari, "Design and Implementation of Reconfigurable Processor for Problems of Combinatorial Computations”, Proc. of the Euromicro Symposium on Digital System Design – DSD’2001, Poland, Sept. 2001, pp.112-119.

[42] V.Sklyarov, et al., “Hierarchical Specification and Implementation of Combinatorial Algorithms Based on RHS Model”, Proc. of the XVI Conf. on Design of Circuits and Integrated Systems – DCIS’2001, Portugal, Nov. 2001, pp. 486-491.