Lógicas de Raciocínio Aproximado e o Problema SAT

FABIO LUIZ IDE

*Dissertação apresentada
ao
Instituto de Matemática e Estatística da
Universidade de São Paulo
para
obtenção do grau
de
Mestre em Ciência da Computação*

Área de concentração: **Ciência da Computação**
Orientador: **Prof. Dr. Marcelo Finger**

São Paulo, setembro de 2005.

Lógicas de Raciocínio Aproximado e o Problema SAT

Este exemplar corresponde à redação
final da dissertação devidamente
corrigida e defendida por
Fabio Luiz Ide
e aprovada pela comissão julgadora.

São Paulo, setembro de 2005.

Banca examinadora:

Prof. Dr. Marcelo Finger (Orientador) - IME-USP

Profa. Dra. Renata Wassermann - IME-USP

Profa. Dr. Guilherme Bittencourt - UFSC

Aos meus pais
Luiz e Noriko
e aos meus irmãos
Danilo e Tatiana

Agradecimentos

Agradeço ao professor *Marcelo Finger* pela sua dedicação e paciência na orientação de meus trabalhos.

Agradeço aos meus pais, *Luiz* e *Noriko*, pela preocupação e carinho. Aos meus irmãos *Danilo* e *Tatiana*, pelo apoio nas horas mais difíceis.

Agradeço aos meus amigos *Adélia*, *Guto*, *Julian*, *Luiz Gustavo*, *Maurício*, *Patrícia*, *Paulo*, *Ricardo Bueno*, *Ricardo Massaro*, *Roberto*, *Shadow* e *Tiago* pelas palavras de incentivo ao longo de todo o curso e pelas dicas na elaboração da dissertação.

Por último, mas não menos importante, agradeço ao pessoal da *Comissão de Pós-Graduação* pela dedicação e atenção em todas as situações.

Abstract

The aim of this work is to use the knowledge of logics for approximate reasoning to improve complete and incomplete SAT solvers, through the creation of new heuristics which reduce the search space size of a given instance of the SAT problem. Two new algorithms, named *S1* and *s1*, were developed and tested on the N-Queens problem to analyze their efficiency.

Resumo

O objetivo deste trabalho é utilizar os conhecimentos de lógicas de raciocínio aproximado para incrementar os métodos de resolução do problema SAT (completos e incompletos), através da criação de novas heurísticas que reduzem o tamanho do espaço de busca de uma dada instância o problema. Foram desenvolvidos dois novos algoritmos, denominados $S1$ e $s1$, que foram testados sobre o problema das N-Rainhas para a análise de sua eficiência.

Sumário

Introdução	1
1 Lógica Proposicional Clássica	5
1.1 Sintaxe	5
1.2 Semântica	6
2 Problema SAT	8
2.1 Métodos de resolução completos	8
2.1.1 DP	9
2.1.2 DLL	10
2.1.3 POSIT	11
2.2 Métodos de resolução incompletos	14
2.2.1 GSAT	15
2.2.2 WalkSAT	17
2.3 Outros métodos de resolução	18
2.3.1 Chaff	18
2.3.2 GRASP	21
2.3.3 DLM	21
3 Lógicas Aproximadas	23
3.1 Lógicas $S1$ e $S3$	23
3.2 Lógica $s1$	27
3.2.1 $s1$ Clausal	29
4 Lógicas $S1$ e $s1$ e o Problema SAT	30
4.1 Método $S1$	30
4.2 Método $s1$	32
4.3 Método s^*1	32
5 Implementação dos Algoritmos	35
5.1 Considerações	35
5.2 Hierarquia de classes	35
5.3 Estruturas de dados e implementação	38
5.3.1 Estruturas de dados básicas	38
5.3.2 GSAT Steepest Descent	38
5.3.3 GSAT Hill Climbing	39
5.3.4 POSIT	41
5.3.5 Lógicas de raciocínio aproximado	41
5.3.6 $S1$	42
5.3.7 $s1$ Min	42
5.3.8 $s1$ Max	43
5.3.9 $s1$ Rnd	43

6	Análise comparativa entre os métodos	44
6.1	Tempo de execução	44
6.2	Eficiência das heurísticas	45
7	Conclusão	51
A	Resultados	52
	Referências Bibliográficas	56

Lista de Figuras

5.1	Hierarquia de classes	36
5.2	Estrutura de dados FormulaMultiMap	40
5.3	Simulação da execução do método GSAT Hill Climbing	41

Lista de Tabelas

6.1	Resultados dos melhores métodos sobre o problema das N-Rainhas	46
6.2	Resultados dos métodos de pior performance sobre o problema das N-Rainhas	47
6.3	Métodos s*1 MAX e s*1 RND: número médio de variáveis incluídas no conjunto S para obtenção da solução	48
6.4	Métodos s*1 MIN e S1: número médio de variáveis incluídas no conjunto S para obtenção da solução	48
6.5	Resultados do método GSAT Hill Climbing com e sem heurísticas	49
6.6	Resultados do método GSAT Steepest Descent com e sem heurísticas	49
6.7	Resultados do método POSIT com e sem heurísticas	50
A.8	Tempos de execução do método S1 com GSAT Steepest Descent, para cada fator.	52
A.9	Tempos de execução do método s*1 MAX com GSAT Steepest Descent, para cada fator.	53
A.10	Tempos de execução do método s*1 RND com GSAT Steepest Descent, para cada fator.	54
A.11	Tempos de execução do método s*1 MIN com GSAT Steepest Descent, para cada fator.	55

Lista de Algoritmos

2.1	Algoritmo DLL	12
2.2	Algoritmo GSAT	15
2.3	Algoritmo GSAT adaptado (Hill Climbing)	17
2.4	Algoritmo WalkSAT	18
4.5	Algoritmo S1	31
4.6	Algoritmo s1	33
4.7	Algoritmo s*1	34

Introdução

O problema SAT

A *Teoria da Complexidade Computacional* é uma das áreas da Teoria da Computação em que se estuda a complexidade de resolução de problemas em função dos recursos disponíveis, como tempo e espaço.

Nesta teoria, está definida a classe P que contém todos os problemas de decisão que podem ser resolvidos em tempo polinomial em relação ao tamanho da entrada. Por outro lado, a classe NP contém todos os problemas cuja resposta pode ser verificada em tempo polinomial.

O maior problema em aberto na Teoria da Computação consiste em determinar qual o relacionamento entre estas classes, mais especificamente se:

$$P = NP$$

Dentro da classe de problemas NP , existe o subconjunto chamado *NP-completo*, que compreende os problemas com a seguinte propriedade: se qualquer um dos problemas desta classe puder ser resolvido em tempo polinomial, então todo problema NP é polinomial - ou seja, $P = NP$. Porém, até hoje não se conhece nenhum problema NP -completo que possa ser resolvido em tempo polinomial, o que leva a maioria dos pesquisadores da área a crer que $P \neq NP$.

Em nosso dia a dia, pode-se verificar uma aplicação desta propriedade sobre métodos de encriptação de dados. Um destes métodos, o RSA [RSA78], utiliza como chave pública o produto de dois números primos grandes. O método é quebrado caso sejam identificados estes dois números a partir da chave. Porém, a decomposição de um número em seus fatores primos é um problema NP -completo e, como $P \neq NP$, não existe hoje um algoritmo capaz de fazê-lo em tempo polinomial (eficiente). Daí decorre a segurança do método RSA.

Outro problema NP-completo é o *Problema da Satisfatibilidade (SAT)*, considerado o primeiro problema NP-completo, conforme provado por Cook em 1971 [Cook71]. Até o momento, todos os algoritmos desenvolvidos para sua resolução são exponenciais em relação ao tamanho da entrada. Desta forma, para poder resolvê-lo em tempo aceitável, são utilizadas algumas técnicas como:

- Restrição no tamanho da entrada;
- Busca por uma resposta aproximada (que pode não corresponder à resposta ótima);
- Uso de um algoritmo otimizado somente para algumas instâncias do problema original;
- Uso de métodos probabilísticos;
- Uso de heurísticas.

Métodos de resolução para o problema SAT

Os métodos de resolução do problema SAT dividem-se em completos e incompletos.

Os métodos completos analisam de forma sistemática todo o espaço de busca para a dada instância do problema SAT, sendo capazes de encontrar todas as soluções possíveis ou indicar que tal instância não é satisfazível. No entanto, estes métodos levam a um maior esforço computacional.

São exemplos de métodos completos os algoritmos DP (Davis-Putnam) [DP60], DLL (Davis, Logemann e Loveland) [DLL62], POSIT [Free95] e Chaff [MMZ⁺01].

Os métodos incompletos, por outro lado, baseiam-se no conceito de busca local. Embora de forma geral estes métodos sejam capazes de fornecer soluções mais rapidamente que os métodos completos, eles não asseguram a cobertura de todo o espaço de busca. Desta forma, ao se executar um método incompleto sobre uma instância do problema SAT, têm-se as seguintes possibilidades:

- O método encontra uma valoração que satisfaz tal instância que, portanto, é satisfazível;
- O método não encontra tal valoração. Neste caso, não se pode afirmar nada a respeito da instância do problema SAT considerada.

Como exemplos de métodos incompletos, podem ser citados os algoritmos GSAT [SLM92], WalkSAT [SKC96] e MLD (Método Lagrangiano Discreto) [SW98].

Lógicas de Raciocínio Aproximado

Cadoli e Schaerf propuseram um método de raciocínio aproximado para resolução de problemas, definindo duas lógicas, denominadas $S3$ e $S1$ [SC95]. Enquanto a lógica $S3$ serviria de limitante inferior para a lógica clássica, a lógica $S1$ seria seu limitante superior. Porém, este método restringe-se a fórmulas na forma clausal.

Finger e Wassermann propuseram uma generalização sobre o método de raciocínio aproximado de Cadoli e Schaerf, eliminando a restrição a fórmulas na forma clausal [FW01]. Eles estenderam a semântica da lógica $S3$ para toda a lógica proposicional, além de definir uma heurística para aproximação incremental de $S3$ até a lógica clássica usando métodos de tableaux.

Em outro estudo de Finger e Wassermann [FW02], foi feita a definição de uma outra lógica aproximada, denominada $s1$, que procura contornar as limitações da lógica $S1$.

Objetivos

O objetivo deste trabalho é implementar e analisar a utilidade das heurísticas baseadas em lógicas de raciocínio aproximado, integradas a métodos completos e incompletos para resolução do problema SAT. Foram construídos dois métodos: o primeiro baseado na lógica $S1$, o segundo na lógica $s1$. Ambos têm uma função comum: a partir da entrada do problema ori-

ginal, devem gerar um subproblema equivalente de tamanho menor. Este subproblema será então resolvido pelos algoritmos GSAT e POSIT.

A partir dos resultados obtidos, foram feitos testes comparativos dos métodos, avaliando sua eficácia frente aos métodos GSAT e POSIT sem tais heurísticas.

Vale ressaltar que o foco deste trabalho é analisar as possíveis melhorias que podem ser trazidas ao agregar os métodos de raciocínio aproximado aos algoritmos já existentes para resolução do problema SAT. Desta forma, não faz parte do escopo deste trabalho a definição de um método otimizado de resolução.

No Capítulo 1, são descritos os conceitos básicos da Lógica Proposicional Clássica.

No Capítulo 2, é apresentado o problema SAT e alguns dos principais métodos para sua resolução, divididos em completos e incompletos.

No Capítulo 3, são discutidas as lógicas de raciocínio aproximado de Cadoli e Schaerf (S1 e S3) e de Finger e Wassermann (s1).

No Capítulo 4, é descrita a base teórica para as heurísticas S1 e s1, a partir das respectivas lógicas de raciocínio aproximado, e sua integração com os métodos de resolução tradicionais do problema SAT.

No Capítulo 5, é discutida a implementação dos métodos, detalhando-se aspectos como hierarquia de classes, estruturas de dados e algoritmos utilizados.

No Capítulo 6, são descritos os resultados e é feita a análise comparativa entre os métodos.

Por fim, o Capítulo 7 apresenta as conclusões e indica os trabalhos futuros.

1 Lógica Proposicional Clássica

A seguir, serão enunciadas as definições relacionadas à Lógica Proposicional Clássica e à notação que será utilizada ao longo de todo o texto.

1.1 Sintaxe

Os elementos básicos considerados na Lógica Proposicional Clássica são as variáveis proposicionais e os conectivos lógicos. As *variáveis proposicionais* (ou *átomos* ou simplesmente *variáveis*) consistem de símbolos pertencentes a um conjunto enumerável pré-definido, denominado P . Para os membros deste conjunto serão utilizadas letras minúsculas. Desta forma, $P = \{a, b, c, d, \dots\}$.

Já os *conectivos lógicos* consistem em 4 símbolos: $\wedge$ (conjunção), $\vee$ (disjunção), $\rightarrow$ (implicação) e $\neg$ (negação). Os conectivos lógicos ligam-se às variáveis proposicionais da seguinte maneira:

- O conectivo da negação $\neg$ é chamado de *conectivo unário*, pois liga-se a apenas uma variável proposicional. Ex.: $\neg a$
- Já os conectivos $\wedge$, $\vee$ e $\rightarrow$ são denominados *conectivos binários*, pois servem para unir duas variáveis proposicionais. Ex.: $a \wedge b$, $c \vee d$, $f \rightarrow g$

A partir destes dois elementos básicos, pode-se definir o que são literais e fórmulas. Um *literal* corresponde a uma variável proposicional, precedida ou não de um conectivo lógico de negação $\neg$. Assim, $\neg a$ e a são literais associados à variável proposicional a .

Uma *fórmula* consiste em uma sucessão de literais ligados por conectivos lógicos. As fórmulas serão representadas por letras gregas minúsculas: $\alpha, \beta, \gamma, \dots$

Uma fórmula pode ser definida a partir das seguintes regras:

- Toda variável proposicional é uma fórmula;
- Se α é uma fórmula, então $\neg \alpha$ é uma fórmula;

- Se α e β são fórmulas, então $\alpha \wedge \beta$ é uma fórmula;
- Se α e β são fórmulas, então $\alpha \vee \beta$ é uma fórmula;
- Se α e β são fórmulas, então $\alpha \rightarrow \beta$ é uma fórmula.

Como notação, serão utilizadas letras gregas maiúsculas para indicar conjuntos de fórmulas: $\Gamma, \Delta, \Lambda, \dots$. O conjunto de todas as fórmulas será representado por Ω .

A ordem de precedência entre os conectivos lógicos é a seguinte (da maior para menor precedência): $\neg, \wedge, \vee, \rightarrow$. Podem ser utilizados *parênteses* $()$ para redefinição desta ordem de precedência.

Uma *cláusula* consiste em uma disjunção de literais. Diz-se que uma fórmula está no *formato clausal* ou na *Forma Normal Conjuntiva (FNC)* quando consiste em uma conjunção de cláusulas. A partir desta definição, pode-se concluir que:

- Não há ocorrência do conectivo $\rightarrow$;
- O conectivo $\wedge$ não tem ação sobre nenhum literal, somente sobre fórmulas;
- A negação só ocorre em variáveis proposicionais, nunca em fórmulas.

Ex.:

- As fórmulas $(a \wedge b) \vee (c \wedge \neg d)$ e $a \vee (\neg b \wedge c)$ não estão no formato clausal.
- Já a fórmula $(a \vee b \vee c) \wedge (d \vee \neg e \vee \neg f) \wedge (\neg a \vee \neg c \vee f)$ está no formato clausal.

1.2 Semântica

Define-se como *valor-verdade* um elemento do conjunto $V = \{0, 1\}$, onde 0 é considerado um valor *falso*, enquanto que 1, *verdadeiro*.

Uma *valorao* é uma função $\nu : P \rightarrow V$, que associa uma variável proposicional a um valor verdade.

Pode-se estender esta valoração para uma fórmula qualquer α , estabelecendo uma função $\nu : \Omega \longrightarrow V$. A partir destas definições, serão estabelecidas as regras para valoração de uma fórmula:

- $\nu(\neg\alpha) = 0 \Leftrightarrow \nu(\alpha) = 1$
- $\nu(\alpha \vee \beta) = 0 \Leftrightarrow \nu(\alpha) = 0 \text{ e } \nu(\beta) = 0$
- $\nu(\alpha \vee \beta) = 1 \Leftrightarrow \nu(\alpha) = 1 \text{ ou } \nu(\beta) = 1$
- $\nu(\alpha \wedge \beta) = 0 \Leftrightarrow \nu(\alpha) = 0 \text{ ou } \nu(\beta) = 0$
- $\nu(\alpha \wedge \beta) = 1 \Leftrightarrow \nu(\alpha) = 1 \text{ e } \nu(\beta) = 1$
- $\nu(\alpha \rightarrow \beta) = 0 \Leftrightarrow \nu(\alpha) = 1 \text{ e } \nu(\beta) = 0$
- $\nu(\alpha \rightarrow \beta) = 1 \Leftrightarrow \nu(\alpha) = 0 \text{ ou } \nu(\beta) = 1$

A partir destas regras, pode-se concluir que:

- $\nu(\neg\neg\alpha) = \nu(\alpha)$
- $\nu(\alpha \rightarrow \beta) = \nu(\neg\alpha \vee \beta)$

2 Problema SAT

O *Problema da Satisfatibilidade (SAT)* é um problema de decisão, em que cada instância consiste de um conjunto de fórmulas no formato clausal, e deseja-se verificar se existe alguma valoração para as variáveis que torne verdadeiras as fórmulas.

Por exemplo, considere o conjunto de fórmulas abaixo:

$$\begin{aligned}(c \vee \neg e) \\ (\neg c \vee d \vee \neg a) \\ (b \vee e)\end{aligned}$$

Todas as fórmulas serão satisfeitas se $\nu(a) = 0$, $\nu(b) = 1$, $\nu(c) = 1$, $\nu(d) = 1$ e $\nu(e) = 0$. Portanto, este conjunto de fórmulas é satisfazível.

Por outro lado, considere o seguinte conjunto de fórmulas:

$$\begin{aligned}(a \vee b) \\ (a \vee \neg b) \\ (\neg a \vee c) \\ (\neg a \vee \neg c)\end{aligned}$$

Não existe nenhuma valoração que torne verdadeiras todas as fórmulas. Portanto, esta instância não é satisfazível.

Visto que na lógica clássica qualquer conjunto de fórmulas pode ser traduzido para o formato clausal, todos os métodos de resolução do problema SAT trabalham com instâncias neste formato. Estes métodos estão agrupados em completos e incompletos, cujas características serão discutidas a seguir.

2.1 Métodos de resolução completos

Os *métodos completos* são aqueles capazes de analisar sistematicamente todo o espaço de busca. Como a verificação de todas as possíveis valorações

é computacionalmente inviável, estes métodos tratam de eliminar valores inconsistentes para as avaliações a partir do conjunto de fórmulas analisado, reduzindo, assim, o espaço de busca.

Por analisarem todo o espaço de busca, estes métodos permitem identificar uma série de possíveis soluções para o problema, caso existam, ou então indicar com exatidão que o problema em questão não é satisfazível. Em contrapartida, os métodos completos levam a um grande esforço computacional para sua execução, de forma geral.

Como exemplos de métodos completos, serão descritos os algoritmos DP, DLL e POSIT.

2.1.1 DP

Em 1960, Davis e Putnam publicaram um artigo descrevendo um método para resolução do problema SAT [DP60]. Este método foi denominado *Procedimento de Davis-Putnam* ou, simplesmente, *DP*. O DP é considerado o primeiro algoritmo desenvolvido para resolução do problema SAT, conforme indicado em [GPFW97].

Em sua essência, o DP elimina uma variável do conjunto de fórmulas a cada iteração do algoritmo, removendo as cláusulas nas quais a variável ocorre e simplificando aquelas nas quais ela ocorre negada.

Este algoritmo apresenta deficiências, conforme indicado em [DLL62]:

- Implementação complexa;
- Tendência a um grande consumo de memória, devido ao rápido incremento no número de fórmulas;
- Geração de cláusulas duplicadas devido ao processo de simplificação;
- Raramente gera novas cláusulas unitárias.

Visando superar estes problemas, Davis, Logemann e Loveland propuseram um novo algoritmo, chamado DLL, que será descrito na próxima seção.

2.1.2 DLL

O método *DLL*, publicado em 1962, tornou-se a base para o desenvolvimento dos métodos completos de resolução subseqüentes. Neste método, uma variável pode assumir 3 valores: *falso*, *verdadeiro* e *indeterminado*. Enquanto os dois primeiros correspondem aos valores usuais da lógica clássica, o terceiro é utilizado para aquelas variáveis sem valor atribuído até o momento. Inicialmente, o valor *indeterminado* é atribuído a todas variáveis, que gradativamente recebem os valores "clássicos", visando-se cobrir todas as possibilidades do espaço de busca. Para poder reduzi-lo, são adotadas as seguintes técnicas: *BCP* (*Boolean Constraint Propagation* ou Propagação de Restrições Booleanas) e eliminação de variáveis. O funcionamento destas técnicas está descrito a seguir, considerando-se um conjunto Γ de fórmulas em FNC e uma valoração ν sobre as variáveis de Γ .

BCP - Boolean Constraint Propagation

O algoritmo BCP consiste em uma busca por fórmulas que possuam somente um literal cujo valor é indeterminado. Neste caso, pode-se fixar o valor da sua variável correspondente a este literal em ν , de forma a satisfazer a respectiva fórmula. Fixado este valor, inicia-se o processo de simplificação:

- São eliminadas todas as fórmulas que passam a assumir o valor *verdadeiro*. Isto é, que contenham o literal a quando $\nu(a) = 1$ ou o literal $\neg a$ quando $\nu(a) = 0$;
- São eliminados das fórmulas todos os literais que passam a assumir o valor *falso*. Ou seja, o literal a quando $\nu(a) = 0$ ou o literal $\neg a$ quando $\nu(a) = 1$.

Caso todas as variáveis de uma fórmula sejam eliminadas ao final de uma iteração, tem-se que esta fórmula sempre assumirá o valor *falso* para a valoração parcial ν gerada até aquele momento. Neste caso, diz-se que foi atingido uma condição de *conflito*, devendo-se desconsiderar a ramificação atual e fazer o backtracking até uma ramificação anterior.

Para maiores detalhes, vide as linhas 1 a 23 do Algoritmo 2.1 na página seguinte.

Eliminação de variáveis

Consiste na seleção de uma variável a que será eliminada fixando-se o seu valor em ν . São geradas então duas ramificações:

- Uma que considera $\nu(a) = 1$. Para esta ramificação é adicionada a fórmula $\{a\}$ ao conjunto Γ ;
- Já a outra ramificação considera $\nu(a) = 0$, com a inclusão da fórmula $\{\neg a\}$ ao conjunto Γ

Com a adição de uma cláusula unitária, inicia-se o algoritmo BCP, conforme indicado no item anterior.

A escolha da próxima variável a ser eliminada é feita por meio de heurísticas, que tratam de:

- Estabelecer uma ordem de prioridade para seleção de variáveis;
- Selecionar qual o primeiro valor (*verdadeiro* ou *falso*) a ser atribuído para a variável escolhida.

A eliminação de variáveis está detalhada nas linhas 25 a 31 do Algoritmo 2.1 na próxima página.

De forma geral, outros métodos completos baseados no DLL se diferenciam pelas heurísticas utilizadas para seleção da próxima variável.

2.1.3 POSIT

Em 1995, Jon William Freeman publicou o método completo POSIT para resolução do problema SAT [Free95]. O POSIT baseia-se no método DLL e utiliza diversas técnicas para otimização, divididas em duas categorias:

- Algoritmos de busca adiante ("lookahead"): transformam o problema

Algoritmo 2.1 Algoritmo DLL

Entrada: Conjunto de fórmulas na forma clausal Γ , valoração parcial ν sobre as variáveis de Γ .

Saída: Valoração ν que satisfaz Γ , se Γ for satisfazível.

Método:

```
1: while  $\exists \alpha \in \Gamma \mid \alpha$  cláusula com um único literal indeterminado do
2:   if  $\alpha \equiv (\neg a)$  then
3:      $\nu' \leftarrow \nu \cup \{a = 0\}$ 
4:      $\Gamma' \leftarrow \Gamma \setminus \{\beta \in \Gamma \mid \beta \text{ contém o literal } \neg a\}$ 
5:     for all  $\beta \in \Gamma' \mid \beta$  contém o literal  $a$  do
6:       Remover de  $\beta$  as ocorrências do literal  $a$ 
7:     end for
8:   else
9:      $\nu' \leftarrow \nu \cup \{a = 1\}$ 
10:     $\Gamma' \leftarrow \Gamma \setminus \{\beta \in \Gamma \mid \beta \text{ contém o literal } a\}$ 
11:    for all  $\beta \in \Gamma' \mid \beta$  contém o literal  $\neg a$  do
12:      Remover de  $\beta$  as ocorrências do literal  $\neg a$ 
13:    end for
14:  end if
15:   $\Gamma \leftarrow \Gamma'$ 
16:   $\nu \leftarrow \nu'$ 
17: end while
18: if  $\Gamma = \emptyset$  then
19:   return  $\nu$ 
20: end if
21: if  $\exists \gamma \in \Gamma \mid \gamma$  é uma cláusula vazia then
22:   return  $\emptyset$ 
23: end if
24:
25:  $b \leftarrow$  Próxima variável a ser eliminada
26:  $\nu' \leftarrow DLL(\Gamma \cup \{b\}, \nu)$ 
27: if  $\nu' \neq \emptyset$  then
28:   return  $\nu'$ 
29: else
30:   return  $DLL(\Gamma \cup \{\neg b\}, \nu)$ 
31: end if
32: return  $\emptyset$ 
```

original em um subproblema equivalente com número menor de cláusulas e de variáveis proposicionais. São eles:

- BCP: semelhante ao do método DLL
- Regra do literal puro: exclusão de variáveis que só aparecem com a mesma polaridade em todas as fórmulas
- Exclusão inicial de variáveis a tais que ou a ou $\neg a$ é *simples*, isto é, um literal que só aparece em uma única fórmula. Esta exclusão é feita combinando-se a fórmula que contém o literal simples com as fórmulas que contém o seu oposto.
- Exclusão inicial de variáveis a tais que ou a ou $\neg a$ é *duplo*, isto é, um literal que só aparece em duas fórmulas. A exclusão é feita de forma semelhante a do literal simples.

Em particular, estes 3 últimos algoritmos só são executados na inicialização do POSIT.

- Heurísticas: definem um método para escolha da próxima variável para a ramificação do algoritmo. Pela sua maior relevância, serão descritas com detalhe a seguir:

- MOM - Máximo número de Ocorrências de Mínimo comprimento
Esta heurística, publicada por Pretolani em 1993 [Pret93], é simples, de fácil implementação, independente do tipo de problema e razoavelmente precisa. Seu objetivo é orientar a escolha da variável proposicional para a próxima ramificação, buscando aquela que apresente o maior número de ocorrências em cláusulas de tamanho mínimo.

As cláusulas de tamanho mínimo são as que contêm as variáveis com as maiores restrições para todo o problema. Ramificando-se sobre elas, aumenta-se a eficiência do BCP e a probabilidade de detecção de um ramo inválido antecipadamente.

Esta heurística possui diversas formas de implementação ([ZM88], [CA96], [DABC96]). Especificamente no POSIT, foi implemen-

tada considerando-se tanto literais positivos quanto negativos, pertencentes às cláusulas de tamanho mínimo. Seja $f^*(a)$ o número de ocorrências de um literal indeterminado a nas cláusulas de tamanho mínimo. Os literais candidatos para a próxima ramificação serão aqueles que atenderem os seguintes critérios:

- * $f^*(a)$ é maximal
- * $f^*(\neg a)$ é maximal
- * O valor mínimo entre $f^*(a)$ e $f^*(\neg a)$ é maximal

Com base nestes critérios, pode-se concluir que $f^*(a)$ e $f^*(\neg a)$ são aproximadamente iguais.

– Heurística baseada no BCP

Esta heurística, associada diretamente ao algoritmo BCP, auxilia na definição da próxima variável para a ramificação de forma a obter o maior número possível de cláusulas binárias e minimizar o número de variáveis em aberto. Para cumprir com estas condições, deve-se simular a execução do algoritmo BCP sobre um conjunto de variáveis. Porém, como seria bastante lenta a execução desta heurística sobre todas as variáveis proposicionais, no POSIT ela foi executada logo após a heurística MOM somente sobre a lista de literais candidatos.

2.2 Métodos de resolução incompletos

Os *métodos incompletos* baseiam-se em processos probabilísticos sobre uma busca local. Dadas todas as possíveis valorações para uma dada instância do problema SAT, é estabelecida uma *relação de vizinhança* entre estas valorações. Partindo-se de uma valoração inicial, move-se a cada iteração a uma candidata vizinha, até que seja atingida uma das condições de término:

- Obtenção de valoração que satisfaz a instância do problema SAT;
- Atingido o número máximo de iterações. Neste caso, seleciona-se uma nova valoração candidata inicial e reinicia-se a busca local.

Os métodos incompletos utilizam-se de heurísticas para escolha da próxima valoração vizinha. Como exemplos de métodos incompleto, serão descritos os algoritmos GSAT e algumas de suas variações (como WalkSAT).

2.2.1 GSAT

Em 1992, Selman, Levesque e Mitchel publicaram um método para resolução do problema SAT [SLM92], chamado de GSAT. Ele é baseado em um algoritmo de busca local guloso e randômico que busca uma valoração que satisfaça o conjunto de fórmulas dado na forma clausal. Segue a descrição do algoritmo básico do GSAT (Algoritmo 2.2 nesta página).

Algoritmo 2.2 Algoritmo GSAT

Entrada: Conjunto de fórmulas na forma clausal Γ , inteiros **MAX-INVERSOES** e **MAX-TENTATIVAS**.

Saída: Valoração ν que satisfaz Γ , se encontrada.

Método:

```

1: for  $i \leftarrow 1$  to MAX-TENTATIVAS do
2:    $\nu \leftarrow$  uma valoração randômica inicial para as variáveis
3:   for  $j \leftarrow 1$  to MAX-INVERSOES do
4:     if  $\nu$  satisfaz  $\Gamma$  then
5:       return  $T$ 
6:     end if
7:      $a \leftarrow$  variável cuja alteração no seu valor forneça o maior aumento
       no número de fórmulas de  $\Gamma$  satisfeitas por  $T$ 
8:      $\nu \leftarrow \nu$  com o valor de  $a$  invertido
9:   end for
10: end for
11: return "Valoração não encontrada"

```

No GSAT, ao iniciar uma nova *tentativa*, gera-se uma valoração aleatória para a solução do problema (linha 2 do Algoritmo 2.2). A partir daí, são executadas as *inversões* sobre os valores das variáveis (linhas 3 a 9), visando-se atingir a valoração vizinha que maximiza o número de fórmulas satisfeitas. Esta versão do GSAT que busca a melhor valoração vizinha a cada inversão será denominada *Steepest Descent* neste texto.

No momento de escolher a variável para a inversão, existem 3 possíveis situações:

- Existem variáveis que incrementam o número de fórmulas satisfeitas. Neste caso, temos um *movimento para baixo*;
- Seja qual for a variável escolhida para a inversão, ocorrerá uma diminuição no número de cláusulas satisfeitas. Neste caso, temos um *movimento para cima*;
- A inversão do valor leva no máximo à manutenção no total de cláusulas satisfeitas. Esta situação é denominada de *movimento lateral*. Conforme indicado em [SLM92], a existência deste *movimento lateral* é essencial para uma execução bem sucedida do algoritmo.

Caso não seja encontrada uma solução após a execução do número máximo de tentativas pré-definido, o algoritmo termina com um resultado indefinido (linha 11), visto que não é possível determinar se a instância é satisfazível ou não.

Variações sobre o método GSAT original

Existem variações para o algoritmo GSAT para o cálculo da próxima variável x que sofrerá inversão do valor, visando uma redução no tempo de execução do algoritmo. Por exemplo, podem ser atribuídos pesos às fórmulas de Γ , atualizados durante o processo de busca local. A escolha da variável deve maximizar o peso das fórmulas satisfeitas a cada passo. Pode-se considerar que esta é uma generalização do algoritmo original, sendo que este considera todas as fórmulas com peso fixo igual a 1.

Alternativamente, pode-se selecionar a como a primeira variável encontrada que incremente o total de cláusulas satisfeitas. Este procedimento evitaria o cálculo do melhor incremento dentre todas as variáveis, que é relativamente custoso, principalmente quando o número de fórmulas e variáveis proposicionais é grande.

Esta variação, que será denominada *GSAT Hill Climbing* neste texto, está descrita no Algoritmo 2.3 na próxima página. A técnica de seleção da variável para inversão está detalhada nas linhas 7 a 10.

Algoritmo 2.3 Algoritmo GSAT adaptado (Hill Climbing)

Entrada: Conjunto de fórmulas na forma clausal Γ , inteiros **MAX-INVERSOES** e **MAX-TENTATIVAS**.

Saída: Valoração T que satisfaz Γ , se encontrada.

Método:

```
1: for  $i \leftarrow 1$  to MAX-TENTATIVAS do
2:    $\nu \leftarrow$  uma valoração randômica inicial para as variáveis
3:   for  $j \leftarrow 1$  to MAX-INVERSOES do
4:     if  $\nu$  satisfaz  $\Gamma$  then
5:       return  $T$ 
6:     end if
7:      $a \leftarrow$  primeira variável cuja inversão do valor aumente o número de
      fórmulas de  $\Gamma$  satisfeitas por  $T$ 
8:     if  $\nexists a$  then
9:        $a \leftarrow$  variável cuja alteração no seu valor forneça o menor decre-
        mento no número de fórmulas de  $\Gamma$  satisfeitas por  $T$ 
10:    end if
11:     $\nu \leftarrow \nu$  com o valor de  $a$  invertido
12:  end for
13: end for
14: return "Valoração não encontrada"
```

2.2.2 WalkSAT

Conforme indicado em [SKC96], mesmo com os movimentos laterais, o método de busca do GSAT pode ficar travado em um mínimo local, chamado de *platô*: conjunto de estados vizinhos com número igual de cláusulas não satisfeitas. Como alternativa para contornar este problema, Selman e Kautz introduziram em 1993 o método WalkSAT, que mescla duas estratégias para solução:

- *Passeio randômico*: escolhe-se aleatoriamente uma fórmula não satisfeita e inverte-se o valor de uma de suas variáveis, também escolhida de forma aleatória (vide linhas 8 a 9 do Algoritmo 2.4 na página seguinte);
- *Método guloso*: adota-se o esquema original do GSAT, alterando-se o valor da variável de forma a obter o melhor incremento no número de cláusulas satisfeitas (conforme indicado na linha 11 do Algoritmo 2.4).

A cada passo, executa-se ou o passeio randômico com uma probabilidade p ou o método guloso, com probabilidade $1 - p$. Esta mistura de estratégias evita que o algoritmo fique travado em um platô.

Algoritmo 2.4 Algoritmo WalkSAT

Entrada: Conjunto de fórmulas na forma clausal Γ , inteiros **MAX-INVERSOES** e **MAX-TENTATIVAS**, probabilidade p .

Saída: Valoração ν que satisfaz Γ , se encontrada.

Método:

```

1: for  $i \leftarrow 1$  to MAX-TENTATIVAS do
2:    $\nu \leftarrow$  uma valoração randômica inicial para as variáveis
3:   for  $j \leftarrow 1$  to MAX-INVERSOES do
4:     if  $\nu$  satisfaz  $\Gamma$  then
5:       return  $T$ 
6:     end if
7:     if probabilidade  $\leq p$  then
8:        $F \leftarrow$  fórmula selecionada aleatoriamente de  $\Gamma$ 
9:        $a \leftarrow$  variável selecionada aleatoriamente de  $F$ 
10:    else
11:       $a \leftarrow$  variável cuja alteração no seu valor forneça o maior aumento
        no número de fórmulas de  $\Gamma$  satisfeitas por  $T$ 
12:    end if
13:     $\nu \leftarrow \nu$  com o valor de  $a$  invertido
14:  end for
15: end for
16: return "Valoração não encontrada"

```

2.3 Outros métodos de resolução

Por fim, serão citados outros métodos de resolução do problema SAT.

2.3.1 Chaff

Foi publicado em 2001 o método completo *Chaff* para resolução do problema SAT [MMZ⁺01]. O *zChaff*, implementação deste método, foi considerado o melhor resolvidor completo em duas edições do SAT Challenge (2002 e 2004).

A boa performance do *zChaff* não está baseada em algoritmos sofisti-

cados para redução do espaço de busca, mas sim em um desenho bastante eficiente dos processos chaves dos métodos completos: BCP e seleção da próxima variável. Antes de descrever a implementação destes processos, vale indicar algumas definições do método Chaff:

- Uma cláusula é denominada *implicada* se e somente se todos seus literais exceto um possuírem valor falso. Para este outro literal, existem duas possibilidades: ou ele ainda não possui valor atribuído, ou então possui valor verdadeiro.
- Quando o algoritmo BCP entra em *conflito*, além de efetuar o backtracking, são incluídas novas cláusulas ao problema para evitar a repetição deste conflito. Estas cláusulas são denominadas *cláusulas de conflito*;
- Para evitar estouro de memória, estas *cláusulas de conflito* são periodicamente eliminadas, conforme sua relevância para a solução do problema;
- O processo de seleção da variável para a próxima ramificação é denominado *decisão*.

A seguir, será detalhada a implementação destes processos chaves:

BCP otimizado

Conforme indicado no artigo [MMZ⁺01], mais de 80% do tempo de execução de um método completo é destinado ao algoritmo de BCP. Desta forma, o desenvolvimento de um algoritmo eficiente é crítico para uma boa performance de todo o método.

Para otimizar a propagação de restrições booleanas, é necessário determinar de forma eficiente quais as cláusulas implicadas quando uma variável deixa de ser indeterminada. Para isso, o método Chaff *observa* duas variáveis quaisquer cujo valor é diferente de falso para cada cláusula. O procedimento adotado é o seguinte:

- No início, são selecionados arbitrariamente dois literais de cada cláusula para observação;
- A cada atribuição de valor, verificam-se as cláusulas que passam a

possuir um literal observado com valor falso. Existem 3 possíveis situações:

1. A cláusula já está satisfeita. Não é necessário tomar nenhuma ação;
2. A cláusula não está satisfeita e continua não implicada. Desta forma, existe um outro literal na cláusula, com valor verdadeiro ou indeterminado, que deve substituir o literal observado em questão;
3. A cláusula não está satisfeita e passou a ser implicada. Neste caso, é identificada uma nova variável para a propagação de restrições booleanas.

Principais vantagens desta técnica:

- Evita a avaliação de todas as cláusulas que contenham o literal que passou a ter valor falso;
- No caso de backtracking, não é necessário atualizar os literais observados das cláusulas envolvidas, o que torna constante o tempo para desassociar os valores das variáveis;
- Há uma concentração dos literais observados - à medida que o algoritmo é executado, reduz-se o número de cláusulas que observam um determinado literal, de forma que o tempo para se reatribuir um valor para uma variável tende a ser menor.

VSIDS - Variable State Independent Decaying Sum

Esta técnica consiste na estratégia para seleção da variável para a próxima ramificação (*decisão*). Os algoritmos tradicionais consideravam que a estratégia mais eficiente deveria minimizar o número de decisões durante a execução do algoritmo. No entanto, conforme analisado para o método Chaff, nem sempre o menor número de ramificações leva a um menor esforço computacional, visto que pode sobrecarregar o BCP.

Assim, o método Chaff definiu a estratégia de seleção VSIDS da seguinte forma:

- Para cada variável proposicional e para cada uma de suas polaridades é atribuído um contador, inicializado com zero.
- Ao se adicionar uma cláusula ao problema (seja na inicialização, seja uma nova cláusula de conflito), é incrementado o contador de cada um dos literais da cláusula;
- A variável indeterminada correspondente ao literal com o maior contador é escolhida a cada decisão. Deve-se utilizar uma estrutura de dados adequada para que as consultas e manutenções sejam eficientes;
- Caso haja mais de uma variável, a escolha é feita aleatoriamente;
- Periodicamente, todos os contadores são divididos por uma constante. Desta forma, os literais pertencentes às cláusulas incluídas mais recentemente acabam tendo maior prioridade.

Desta forma, este algoritmo privilegia a resolução das últimas cláusulas de conflito. Outro aspecto importante é que, por não depender dos valores das variáveis, ele gera uma sobrecarga menor, visto que os contadores só são atualizados em caso de conflito.

2.3.2 GRASP

O método *GRASP* (**G**eneric **seaR**ch **A**lgorithm for the **S**atisfiability **P**roblem), desenvolvido por Silva e Sakallah e publicado em 1996 [MSS96], consiste em um método completo, baseado em uma poderosa análise de conflitos e suas respectivas causas, que leva a uma considerável redução do espaço de busca.

2.3.3 DLM

O *DLM* (Discrete Lagrangian Method) é um método incompleto, descrito em um artigo de 1998 publicado por Shang e Wah [SW98]. Nele, a instância do problema SAT é traduzida para um problema de otimização discreta restrita e, a partir daí, aplica-se um método baseado em multiplicadores de Lagrange para contornar de forma consistente os mínimos locais atingidos durante a busca, sem implicar um novo início ou uso de movimento

aleatórios.

3 Lógicas Aproximadas

3.1 Lógicas $S1$ e $S3$

Cadoli e Schaerf propuseram um método de raciocínio aproximado para resolução de problemas, quando a obtenção da solução completa exigiria muitos recursos [SC95]. Este método define duas lógicas, $S1$ e $S3$, que serviriam de limitantes superior e inferior para a lógica clássica, trazendo como vantagem uma maior facilidade para os cálculos. O método considera apenas fórmulas na forma clausal.

Ambas lógicas $S1$ e $S3$ são definidas a partir de um conjunto de variáveis proposicionais $S \subseteq P$, que indica sua proximidade à lógica clássica.

Para uma valoração ν em $S3$, existem as possibilidades abaixo:

- $\nu(p) = 1, \nu(\neg p) = 0$;
- $\nu(p) = 0, \nu(\neg p) = 1$;
- $\nu(p) = \nu(\neg p) = 1$, somente se $p \in P \setminus S$.

Para um conjunto de cláusulas Γ e uma cláusula α , diz-se que Γ *implica logicamente* α em $S3$ (notação: $\Gamma \models_S^3 \alpha$) se e somente se toda valoração em $S3$ que satisfaz Γ também satisfaz α .

Para uma valoração ν em $S1$, existem as seguintes possibilidades:

- $\nu(p) = 1, \nu(\neg p) = 0$;
- $\nu(p) = 0, \nu(\neg p) = 1$;
- $\nu(p) = \nu(\neg p) = 0$, se e somente se $p \in P \setminus S$.

Para um conjunto de cláusulas Γ e uma cláusula α , diz-se que Γ *implica logicamente* α em $S1$ (notação: $\Gamma \models_S^1 \alpha$) se e somente se toda valoração em $S1$ que satisfaz Γ também satisfaz α .

A partir das definições, é fácil ver que se $S = P$, tanto $S1$ como $S3$

comportam-se como a lógica clássica.

Cadoli e Schaerf provam em seu trabalho que $\models_S^1$ é completa porém incorreta e, desta forma, pode ser utilizada para aproximar $\not\models$. Isto significa que, se $\Gamma \not\models_S^1 \alpha$ para algum S , então $\Gamma \not\models \alpha$. Por outro lado, $\models_S^3$ é incompleta porém correta, podendo, então, ser utilizada para aproximar $\models$. Assim, se $\Gamma \models_S^3 \alpha$ para algum S , então $\Gamma \models \alpha$. Os exemplos a seguir ilustram o uso destas lógicas:

EXEMPLO 1 [SC95] Considere:

$$\begin{aligned}\Gamma = & (\neg cow \vee grass-eater), \\ & (\neg dog \vee carnivore), \\ & (\neg grass-eater \vee \neg has-canine-teeth), \\ & (\neg grass-eater \vee mammal), \\ & (\neg carnivore \vee mammal), \\ & (\neg mammal \vee has-canine-teeth \vee has-molar-teeth), \\ & (\neg mammal \vee vertebrate), \\ & (\neg vertebrate \vee animal) \\ \alpha = & (\neg cow \vee has-molar-teeth)\end{aligned}$$

Deseja-se provar que $\Gamma \models \alpha$. Para isso, basta determinar um conjunto S tal que $\Gamma \models_S^3 \alpha$. Isto ocorre quando $S = \{cow, has-molar-teeth, grass-eater, mammal, has-canine-teeth\}$.

EXEMPLO 2 [SC95] Considere:

$$\begin{aligned}\Gamma = & (\neg person \vee child \vee youngster \vee adult \vee senior), \\ & (\neg youngster \vee student \vee worker), \\ & (\neg adult \vee student \vee worker \vee unemployed), \\ & (\neg senior \vee pensioner \vee worker), \\ & (\neg student \vee child \vee youngster \vee adult), \\ & (\neg pensioner \vee senior), \\ & (\neg pensioner \vee \neg student), \\ & (\neg pensioner \vee \neg worker) \\ \alpha = & (\neg child \vee pensioner)\end{aligned}$$

Deseja-se provar que $\Gamma \not\models \alpha$. Como $\Gamma \not\models_S^1 \alpha$ para $S = \{child, worker, pensioner\}$,

então $\Gamma \not\models \alpha$.

Cadoli e Schaerf descrevem um algoritmo para aproximação da lógica clássica a partir de $S1$ e $S3$, efetuando simplificações sobre o conjunto de cláusulas, conforme descrito a seguir. Considere um conjunto de cláusulas Γ , uma cláusula α e um conjunto S de variáveis proposicionais.

Lema 3.1 [SC95] *Seja $\text{simplifica-1}(\Gamma, S)$ uma função de simplificação que remove todos os literais de Γ correspondentes a variáveis que não pertençam a S . Γ é $S1$ -satisfazível se e somente se $\text{simplifica-1}(\Gamma, S)$ é classicamente satisfazível. Caso todos os literais de uma cláusula sejam removidos, a cláusula em questão assumirá sempre o valor falso.*

Teorema 3.2 [SC95] *Considere $\alpha = \alpha_S \vee \alpha'_S$, onde α_S contém somente átomos de α que pertençam a S , e nenhum dos átomos de α'_S pertence a S . Então, $\Gamma \models_S^1 \alpha$ se e somente se $\Gamma \cup \{\neg \alpha_S\}$ não é $S1$ -satisfazível.*

Lema 3.3 [SC95] *Seja $\text{simplifica-3}(\Gamma, S)$ o resultado de remover todas as cláusulas de Γ que contenham pelo menos uma variável que não pertença a S . Γ é $S3$ -satisfazível se e somente se $\text{simplifica-3}(\Gamma, S)$ é classicamente satisfazível.*

Teorema 3.4 [SC95] *Se todos os átomos de α pertencem a S , então $\Gamma \models_S^3 \alpha$ se e somente se $\Gamma \cup \{\neg \alpha\}$ não é $S3$ -satisfazível.*

EXEMPLO 3 Considere os dados do exemplo 2 na página precedente. O objetivo é construir um conjunto S tal que $\Gamma \not\models_S^1 \alpha$ ou $\Gamma \models_S^3 \alpha$. Com isso, será possível concluir que $\Gamma \not\models \alpha$ ou $\Gamma \models \alpha$, respectivamente.

Para poder aplicar o Teorema 3.4, inicializa-se o conjunto S com os átomos de α . Em seguida, são aplicados os métodos de simplificação definidos nos Lemas 3.1 e 3.3, como segue:

1. $S \leftarrow \{child, pensioner\}$. Usando o Teorema 3.2, faça

$$\begin{aligned}
\Gamma' &= \text{simplifica-1}(\Gamma \cup \{\neg\alpha_S\}, S) \\
&= \text{simplifica-1}(\Gamma \cup \{child \wedge \neg pensioner\}, S) \\
&= \{child, \text{falso}, pensioner, \neg pensioner\}
\end{aligned}$$

Note que devido à remoção de literais pelo método *simplifica-1*, há cláusulas que não possuem nenhum literal e assumem o valor FALSO. Portanto, o resultado do método *simplifica-1* não pode ser satisfeito classicamente e, assim, $\Gamma \cup \{\neg\alpha_S\}$ não é $S1$ -satisfazível. Pelo Teorema 3.2, $\Gamma \models_S^1 \alpha$.

Usando o Teorema 3.4, faça

$$\begin{aligned}
\Gamma' &= \text{simplifica-3}(\Gamma \cup \{\neg\alpha\}, S) \\
&= \text{simplifica-3}(\Gamma \cup \{child \wedge \neg pensioner\}, S) \\
&= \{child, pensioner\}
\end{aligned}$$

Desta forma, Γ' é satisfazível na lógica clássica e, portanto, a partir do Lema 3.3, deduz-se que $\Gamma \cup \{\neg\alpha\}$ é $S3$ -satisfazível e, pelo Teorema 3.4, conclui-se que $\Gamma \not\models_S^3 \alpha$.

2. $S \leftarrow S \cup \{worker\}$

$$\begin{aligned}
\Gamma' &= \text{simplifica-1}(\Gamma \cup \{\neg\alpha_S\}, S) \\
&= \text{simplifica-1}(\Gamma \cup \{child \wedge \neg pensioner\}, S) \\
&= \{child, worker, pensioner \vee worker, \neg pensioner, \\
&\quad \neg pensioner \vee \neg worker, \neg pensioner\}
\end{aligned}$$

Segue que Γ' é satisfazível na lógica clássica. Portanto, $\Gamma \cup \{\neg\alpha_S\}$ é $S1$ -satisfazível. Pelo Teorema 3.2, conclui-se que $\Gamma \not\models_S^1 \alpha$ e, assim, $\Gamma \not\models \alpha$.

Neste estudo, é proposto o seguinte teorema:

Teorema 3.5 *Seja S um conjunto de variáveis proposicionais e Γ um conjunto de cláusulas. Se $\text{simplifica-1}(\Gamma, S)$ é satisfazível, então Γ também é satisfazível na lógica clássica.*

Demonstração. Se $\text{simplifica-1}(\Gamma, S)$ é satisfazível, então, pelo Lema 3.1 temos que Γ é $S1$ -satisfazível. Considere $\alpha \leftarrow \text{falso}$. Pelo Teorema 3.2, como $\Gamma \cup \{\neg\alpha\}$ continua sendo $S1$ -satisfazível, temos que $\Gamma \not\models_S^1 \alpha$. Aplicando-se a definição de Cadoli e Schaerf [SC95], temos que $\Gamma \not\models \alpha$. Na lógica clássica, se $\Gamma \not\models \text{falso}$, então Γ é satisfazível. Portanto, Γ é satisfazível. $\square$

Uma boa característica do método proposto por Cadoli e Schaerf é estabelecer um algoritmo que se aproxima gradualmente da lógica clássica. Porém, o método apresenta algumas desvantagens:

- Ele é restrito a fórmulas na forma clausal, e não permite o uso de implicação ($\rightarrow$), pois sua tradução clássica para forma clausal (usando $\neg$ e $\vee$) não preserva os valores-verdade sob a semântica das lógicas $S3$ e $S1$, conforme visto em [FW01];
- O raciocínio dentro de $S1$ é não local [FW02]. Com isso, muitas cláusulas e literais irrelevantes para a solução devem ser examinadas desnecessariamente. Isto decorre da definição da lógica $S1$, em que uma cláusula só pode ter um valor diferente de falso se pelo menos uma de suas variáveis estiver no conjunto S .
- Não é descrito nenhum método para escolha da próxima variável a ser incluída no conjunto S .

3.2 Lógica $s1$

Inicialmente, é necessário definir o que são aproximações da lógica clássica por baixo e por cima. Para a propriedade indicada a seguir, considere a lógica $S3$, com a relação de implicação $\models_S^3$, parametrizada sobre um conjunto S pré-estabelecido. Conforme a definição de Cadoli e Schaerf, se

$$\emptyset \subseteq S' \subseteq S'' \subseteq \dots \subseteq S'^n \subseteq P$$

então

$$\models_{\emptyset}^3 \subseteq \models_{S'}^3 \subseteq \models_{S''}^3 \subseteq \dots \subseteq \models_{S'^n}^3 \subseteq \models_P^3 = \models$$

onde $\models$ é a implicação clássica. Por este motivo, a lógica $S3$ pode ser chamada de uma *aproximação da lógica clássica por baixo*.

De maneira análoga, diz-se que uma família de lógicas L é uma *aproximação da lógica clássica por cima* quando

$$\models_{\emptyset}^L \supseteq \models_{S'}^L \supseteq \models_{S''}^L \supseteq \dots \supseteq \models_{S^m}^L \supseteq \models_P^L = \models$$

Porém, a lógica $S1$ não aproxima a lógica clássica por cima. Se isto ocorresse, qualquer teorema clássico também seria um teorema de $S1$, para qualquer escolha de S . Porém, a fórmula $(a \vee \neg a)$ é falsa em $S1$ se $a \notin S$. Portanto, a lógica $S1$ não pode ser considerada uma aproximação da lógica clássica por cima.

Finger e Wassermann definiram em [FW02] uma nova família de lógicas, denominada $s1$, que aproxima a lógica clássica por cima, superando limitações encontradas na lógica $S1$. Ao invés de partir da valoração binária clássica $\nu : P \rightarrow \{0, 1\}$, foi definida uma nova valoração $\nu_S^1 : P \rightarrow 2^{\{0,1\}}$ para esta nova família de lógicas, que também adotam um conjunto S de variáveis para indicar sua proximidade da lógica clássica.

Segue a definição básica de ν_S^1 :

- $0 \in \nu_S^1(a) \Leftrightarrow \nu(a) = 0$
- $1 \in \nu_S^1 \Leftrightarrow \nu(a) = 1 \text{ ou } a \notin S$

Ou seja, ν_S^1 estende ν mas se $a \notin S$, então $1 \in \nu_S^1(a)$. Se $a \notin S$ e $\nu(a) = 0$, então $\nu_S^1(a) = \{0, 1\}$.

Para as definições subsequentes da atuação de ν_S^1 sobre fórmulas, considere que uma fórmula $\alpha \in S$ caso todas suas variáveis pertençam a S :

- $0 \in \nu_S^1(\neg \alpha) \Leftrightarrow 1 \in \nu_S^1(\alpha)$
- $0 \in \nu_S^1(\alpha \wedge \beta) \Leftrightarrow 0 \in \nu_S^1(\alpha) \text{ ou } 0 \in \nu_S^1(\beta)$
- $0 \in \nu_S^1(\alpha \vee \beta) \Leftrightarrow 0 \in \nu_S^1(\alpha) \text{ e } 0 \in \nu_S^1(\beta)$
- $0 \in \nu_S^1(\alpha \rightarrow \beta) \Leftrightarrow 1 \in \nu_S^1(\alpha) \text{ e } 0 \in \nu_S^1(\beta)$
- $1 \in \nu_S^1(\neg \alpha) \Leftrightarrow 0 \in \nu_S^1(\alpha) \text{ ou } \neg \alpha \notin S$
- $1 \in \nu_S^1(\alpha \wedge \beta) \Leftrightarrow 1 \in \nu_S^1(\alpha) \text{ e } 1 \in \nu_S^1(\beta) \text{ ou } \alpha \wedge \beta \notin S$
- $1 \in \nu_S^1(\alpha \vee \beta) \Leftrightarrow 1 \in \nu_S^1(\alpha) \text{ ou } 1 \in \nu_S^1(\beta) \text{ ou } \alpha \vee \beta \notin S$

- $1 \in \nu_S^1(\alpha \rightarrow \beta) \Leftrightarrow 0 \in \nu_S^1(\alpha) \text{ ou } 1 \in \nu_S^1(\beta) \text{ ou } \alpha \rightarrow \beta \notin S$

Conforme indicado por Finger e Wassermann, esta semântica estende a lógica clássica. A seguir, serão enunciadas algumas propriedades básicas de ν_S^1 decorrentes das definições anteriores:

Lema 3.6 [FW02] *Para qualquer fórmula α , $\nu_S^1(\alpha) \neq \emptyset$. Assim, ν_S^1 é uma função que tem como imagem um conjunto de três elementos: $\{\{0\}, \{1\}, \{0, 1\}\}$.*

Lema 3.7 [FW02] *Para toda fórmula α , $\nu(\alpha) \in \nu_S^1(\alpha)$.*

Lema 3.8 [FW02] *Se $\alpha \in S$, então $\nu_S^1(\alpha) = \{\nu(\alpha)\}$.*

Lema 3.9 [FW02] *Se $\alpha \notin S$, então $1 \in \nu_S^1(\alpha)$.*

3.2.1 s1 Clausal

Considere Γ um conjunto de cláusulas e um conjunto S de variáveis proposicionais. Define-se a função *simplifica-s1*(Γ, S) como o resultado de remover de Γ todas as variáveis *negativas* que não pertençam a S e todas as fórmulas que contenham variáveis *positivas* que não pertençam a S . Por outro lado, a função *simplifica-s*1*(Γ, S) remove de Γ todas as variáveis *positivas* que não pertençam a S e todas as fórmulas que contenham variáveis *negativas* que não pertençam a S .

Lema 3.10 [FW02] *Se *simplifica-s1*(Γ, S) é satisfazível classicamente, então Γ também é satisfazível.*

Lema 3.11 [FW02] *Se *simplifica-s*1*(Γ, S) é satisfazível classicamente, então Γ também é satisfazível.*

4 Lógicas $S1$ e $s1$ e o Problema SAT

Para esta seção, considere Γ um conjunto de fórmulas na forma clausal e S o conjunto de variáveis proposicionais das lógicas aproximadas.

Com base nas definições das lógicas aproximadas $S1$ e $s1$, foram definidos dois métodos para resolução do problema SAT. Ambos métodos estão divididos em 3 partes básicas:

1. Inicialização do conjunto S
2. Redução do tamanho da entrada
3. Resolução do subproblema

A seguir, será descrito o funcionamento de cada um dos métodos.

4.1 Método $S1$

Vimos na seção anterior que o método $S1$ é não local. Desta forma, para a inicialização do conjunto S , é necessário incluir ao menos uma variável de cada fórmula para iniciar o procedimento de resolução (indicado nas linhas 4 a 7 do Algoritmo 4.5 na página seguinte). Então, com base no conteúdo do conjunto S , é aplicado o método *simplifica-1*, descrito no Lema 3.1 na página 25, obtendo-se um subproblema de tamanho menor e solução equivalente ao problema original (vide linha 10 do Algoritmo 4.5). Este subproblema obtido poderá ser resolvido ou por GSAT (incompleto) ou por POSIT (completo) (conforme consta na linha 11 do Algoritmo 4.5). Caso não se encontre uma solução, uma variável escolhida aleatoriamente é incluída em S e o processo se repete. Por fim, caso todas as variáveis do problema já estejam em S , executa-se o método SAT sobre o conjunto original de fórmulas, visto que neste momento a lógica $S1$ comporta-se como a lógica clássica (linha 19 do Algoritmo 4.5).

Algoritmo 4.5 Algoritmo S1

Entrada: Conjunto de fórmulas na forma clausal Γ .

Saída: *TRUE*, se existe uma valoração que satisfaça o conjunto Γ , *FALSE* se tal valoração não foi encontrada.

Método:

```
1:  $S \leftarrow \emptyset$ 
2:  $P \leftarrow$  conjunto de todas as variáveis proposicionais de  $\Gamma$ 
3:
4: repeat
5:    $a \leftarrow$  variável escolhida de  $P \setminus S$ ;
6:    $S \leftarrow S \cup \{a\}$ 
7: until  $S$  contém ao menos uma variável de cada fórmula de  $\Gamma$ 
8:
9: while  $S \neq P$  do
10:   $\Gamma' \leftarrow \text{simplifica-1}(\Gamma, S)$ 
11:  if  $\text{metodoSAT}(\Gamma')$  then
12:    return TRUE
13:  else
14:     $a \leftarrow$  variável escolhida de  $P \setminus S$ ;
15:     $S \leftarrow S \cup \{a\}$ 
16:  end if
17: end while
18:
19: return  $\text{metodoSAT}(\Gamma)$ 
```

4.2 Método s1

O conjunto S deve ser inicializado de forma que não haja fórmulas sempre falsas. Esta validação pode ser feita através do método *simplifica-s1* (descrito na seção anterior), como esquematizado nas linhas 4 a 8 do Algoritmo 4.6 na página seguinte. Feito isto, aplica-se o método *simplifica-s1* para redução do tamanho do problema. É obtido um novo conjunto de fórmulas Γ' cuja solução é equivalente ao do conjunto Γ original (vide linha 11 do Algoritmo 4.6). O subproblema obtido será resolvido por ou pelo GSAT ou pelo POSIT (indicado na linha 12 do Algoritmo 4.6). Caso não se encontre uma solução, é adicionada uma nova variável ao conjunto S (linhas 15 a 16 do Algoritmo 4.6 na próxima página). Os métodos para seleção desta variável estarão detalhados no próximo capítulo. Por fim, assim como no método S1, executa-se o método SAT sobre o conjunto original de fórmulas se não for encontrada uma resposta sobre os subproblemas simplificados (linha 20 do Algoritmo 4.6).

4.3 Método s*1

O método s*1 é semelhante ao método s1, com uma única diferença: é usada a função *simplifica-s*1* ao invés da *simplifica-s1*, conforme indicado em Algoritmo 4.7 na página 34.

Algoritmo 4.6 Algoritmo s1

Entrada: Conjunto de fórmulas na forma clausal Γ .

Saída: *TRUE*, se existe uma valoração que satisfaça o conjunto Γ , *FALSE* se tal valoração não foi encontrada.

Método:

```
1:  $S \leftarrow \emptyset$ 
2:  $P \leftarrow$  conjunto de todas as variáveis proposicionais de  $\Gamma$ 
3:
4: repeat
5:    $a \leftarrow$  variável escolhida de  $P \setminus S$ ;
6:    $S \leftarrow S \cup \{a\}$ 
7:    $\Gamma' \leftarrow \text{simplifica-s1}(\Gamma, S)^1$ 
8: until  $\Gamma'$  não contém fórmulas sempre falsas
9:
10: while  $S \neq P$  do
11:    $\Gamma' \leftarrow \text{simplifica-s1}(\Gamma, S)$ 
12:   if metodoSAT( $\Gamma'$ ) then
13:     return TRUE
14:   else
15:      $a \leftarrow$  variável escolhida de  $P \setminus S$ ;
16:      $S \leftarrow S \cup \{a\}$ 
17:   end if
18: end while
19:
20: return metodoSAT( $\Gamma$ )
```

Algoritmo 4.7 Algoritmo s*1

Entrada: Conjunto de fórmulas na forma clausal Γ .

Saída: *TRUE*, se existe uma valoração que satisfaça o conjunto Γ , *FALSE* se tal valoração não foi encontrada.

Método:

```
1:  $S \leftarrow \emptyset$ 
2:  $P \leftarrow$  conjunto de todas as variáveis proposicionais de  $\Gamma$ 
3:
4: repeat
5:    $a \leftarrow$  variável escolhida de  $P \setminus S$ ;
6:    $S \leftarrow S \cup \{a\}$ 
7:    $\Gamma' \leftarrow \text{simplifica-s*1}(\Gamma, S)^2$ 
8: until  $\Gamma'$  não contém fórmulas sempre falsas
9:
10: while  $S \neq P$  do
11:    $\Gamma' \leftarrow \text{simplifica-s*1}(\Gamma, S)$ 
12:   if metodoSAT( $\Gamma'$ ) then
13:     return TRUE
14:   else
15:      $a \leftarrow$  variável escolhida de  $P \setminus S$ ;
16:      $S \leftarrow S \cup \{a\}$ 
17:   end if
18: end while
19:
20: return metodoSAT( $\Gamma$ )
```

5 Implementação dos Algoritmos

5.1 Considerações

A implementação do algoritmo está feita em C++, orientada a objetos. Ao contrário do que foi feito por Freeman na implementação do POSIT [Free95] ou por Selman e Kautz no WalkSAT [SKC96], não houve uma preocupação em redução de fatores constantes através de recodificação de funções da biblioteca padrão, otimização do código assembly gerado pelo compilador, codificação sobre arrays ao invés de estruturas de dados mais elaboradas ou minimização de acessos a memória, visto que este estudo visa analisar as novas heurísticas, e não estabelecer um algoritmo mais otimizado.

Desta forma, foi feito um uso extensivo dos elementos da Standard Template Library (STL):

- Estruturas de dados: vector, list, map e multimap
- Uso de classes-função "functors"
- Algoritmos da biblioteca padrão: sort, for_each, remove_if

Neste capítulo, serão descritos os elementos construídos neste trabalho: classes, estruturas de dados e os algoritmos.

5.2 Hierarquia de classes

As classes do sistema estão indicadas a seguir (Figura 5.1 na próxima página):

Satisfiability Solver Consiste na superclasse de algoritmos para resolução do problema SAT.

General SAT Solver Consiste na superclasse dos algoritmos tradicionais de resolução do problema SAT, completos (como o POSIT) e incompletos (como o GSAT). Estes algoritmos serão aqueles utilizados pelos

Figura 5.1: Hierarquia de classes

métodos de raciocínio aproximado após a aplicação das heurísticas de simplificação do problema original.

GSAT O algoritmo GSAT está implementado concretamente de duas formas:

Steepest Descent consiste no método guloso que busca a cada iteração a variável que proporciona o maior incremento no número de fórmulas satisfeitas (Algoritmo 2.2 na página 15)

Hill Climbing consiste no método que busca a cada iteração a primeira variável que incrementar o número de fórmulas satisfeitas (Algoritmo 2.3 na página 17)

Cada uma destas implementações está refletida em uma subclasse da classe GSAT.

POSIT Foi utilizada a implementação do método POSIT de Freeman, com pequenos ajustes para integração com a estrutura de classes deste projeto.

Approximate Reasoning Consiste na superclasse para os algoritmos de resolução do problema SAT baseados nas lógicas aproximadas S1 e s1.

S1 Consiste na implementação do método S1 (Algoritmo 4.5 na página 31). Para resolução do problema SAT reduzido a partir da aplicação da função *simplifica-1*, este algoritmo necessita de uma instância da classe **General SAT Solver**.

s1 Consiste na implementação do método s1 (Algoritmo 4.6 na página 33). Este algoritmo também necessita de uma instância da classe **General SAT Solver**. O método s1 possui 3 variações:

Maximize procura maximizar o número de fórmulas satisfeitas automaticamente a cada escolha da variável que será incluída no conjunto S da lógica aproximada

Minimize de forma inversa, procura minimizar o número de fórmulas automaticamente satisfeitas

Random escolhe aleatoriamente a próxima variável a incluir no conjunto S

Baseado nesta hierarquia de classes, indicamos as 15 combinações de algoritmos avaliadas neste estudo:

1. GSAT Hill Climbing
2. GSAT Steepest Descent
3. POSIT
4. S1 com GSAT Hill Climbing
5. S1 com GSAT Steepest Descent
6. S1 com POSIT
7. s1 Max com GSAT Hill Climbing
8. s1 Max com GSAT Steepest Descent
9. s1 Max com POSIT
10. s1 Min com GSAT Hill Climbing
11. s1 Min com GSAT Steepest Descent
12. s1 Min com POSIT
13. s1 Random com GSAT Hill Climbing
14. s1 Random com GSAT Steepest Descent

15. s1 Random com POSIT

Com exceção do POSIT, todos os demais algoritmos foram desenvolvidos neste projeto. A seguir, serão descritos as estruturas de dados gerais, as estruturas de dados específicas para cada algoritmo e os detalhes da implementação de cada um deles.

5.3 Estruturas de dados e implementação

5.3.1 Estruturas de dados básicas

Formula Consiste na representação de uma fórmula da instância do problema SAT em questão, contendo a lista de variáveis e suas respectivas polaridades.

Variable Além da sua representação alfanumérica, cada variável possui uma lista com as fórmulas onde está contida. Esta lista visa facilitar o recálculo do número de fórmulas satisfeitas no caso de alterações no valor da variável.

FormulaList Esta estrutura de dados é utilizada para armazenar a lista de todas as fórmulas da instância do problema SAT considerado. Está construída sobre o template *list* da STL, pois todos acessos ao seu conteúdo são seqüenciais.

VariableSet Contém a lista de todas as variáveis proposicionais do problema. Ao contrário da lista de fórmulas, está construída sobre o template *set* da STL, para permitir uma verificação de existência da variável de forma eficiente ($O(\log n)$).

5.3.2 GSAT Steepest Descent

Para este algoritmo, a estrutura de dados foi projetada para que cada inversão no valor de uma variável não implicasse na reavaliação de todas as fórmulas.

A estrutura *FormulaMultiMap* (Figura 5.2 na página seguinte) é utilizada para indexar cada fórmula pelo seu número de literais verdadeiros. Esta estrutura é baseada no template STL *multimap*, implementado através de árvores rubro-negras³.

A cada iteração, para se determinar a melhor inversão, basta analisar:

- Fórmulas não satisfeitas: lista obtida a partir do índice 0 da *FormulaMultiMap*. Para todas estas fórmulas, uma inversão em qualquer uma de suas variáveis implica tornar a fórmula satisfeita.
- Fórmulas que poderiam deixar de ser satisfeitas: lista obtida a partir do índice 1 da *FormulaMultiMap*. Estas fórmulas deixam de ser satisfeitas caso o único literal verdadeiro tenha uma inversão no valor.

Assim, só há a necessidade de avaliação do número de literais verdadeiros de uma fórmula na inicialização do algoritmo, a partir da valoração randômica inicial.

A partir daí, a cada iteração, são percorridas as listas dos índices 0 e 1 da *FormulaMultiMap* para cálculo da alteração no número de fórmulas satisfeitas ao se inverter o valor de cada uma das variáveis das fórmulas das duas listas.

Selecionada a variável para inversão, é feita a atualização da estrutura *FormulaMultiMap*, deslocando-se para cima (índice superior) as fórmulas que passavam a conter mais um literal verdadeiro e para baixo (índice inferior) as fórmulas que passavam a conter um literal verdadeiro a menos.

5.3.3 GSAT Hill Climbing

Para encontrar a primeira variável que apresentasse algum incremento no número de fórmulas satisfeitas, este algoritmo utiliza uma estrutura de dados denominada *VariableList*, desenvolvida sobre o template STL *list*. Esta estrutura contém a lista de variáveis proposicionais do problema SAT em

³As árvores rubro-negras possuem a propriedade de sempre se manterem balanceadas, garantindo o acesso aos dados em $O(\log n)$.

Figura 5.2: Estrutura de dados FormulaMultiMap

questão, ordenadas aleatoriamente na inicialização. A cada iteração, a lista é percorrida de forma seqüencial até que se encontre a variável desejada. Após executar a inversão, esta variável é movida para o final da lista, de forma que a dar maior prioridade às variáveis que não sofreram inversões recentemente.

Ao percorrer a lista, o algoritmo armazena uma outra relação com os átomos não válidos, isto é, que mantêm ou reduzem o número de fórmulas satisfeitas, ordenados pelo seu decremento que geravam. Assim, ao detectar que a valoração atual não permite inversão com incremento no número de cláusulas satisfeitas (por exemplo, no caso de um mínimo local), escolhe-se arbitrariamente um membro com menor decremento desta outra relação para inversão.

A escolha do template STL *list* foi a mais adequada, pois sua implementação via lista duplamente ligada permite um eficiente acesso seqüencial e um remanejamento da ordenação das variáveis em tempo constante.

Figura 5.3: Simulação da execução do método GSAT Hill Climbing

5.3.4 POSIT

Para utilização do POSIT, foram desenvolvidas apenas pequenas adaptações para uma correta integração entre os programas: tradução das estruturas de dados, ajustes nas chamadas dos métodos e valores de retorno.

5.3.5 Lógicas de raciocínio aproximado

Os métodos S1 e s1 possuem estruturas de dados e implementações bastante semelhantes, que serão discutidas neste item. O conjunto S , que contém os átomos cujo comportamento está dentro da lógica clássica, está desenvolvido sobre o template STL *set*. Paralelamente, é mantido um outro conjunto ($notS$), também sobre o template *set*, com as variáveis proposicionais do problema que ainda não estão em S . Além disso, todos os métodos apresentam fórmulas inválidas quando o conjunto S está vazio, isto é, fórmulas que assumem sempre valor falso. Desta forma, é executada uma etapa inicial

de inclusão de átomos em S até eliminar todas as fórmulas inválidas. Para controle das fórmulas inválidas, é utilizado um conjunto baseado no template STL *set*, visando uma exclusão eficiente.

Após esta etapa inicial, eram executadas as iterações do algoritmo com a chamada do método SAT pré-definido (GSAT Hill Climbing, GSAT Steepest Descent, POSIT). No caso específico do método GSAT Hill Climbing, adotou-se uma abordagem distinta: o número de máximos de inversões era igual ao produto entre o tamanho do conjunto S e um fator constante. Foi feito desta forma para que se gastasse menos tempo nas tentativas iniciais, menos propensas ao êxito.

Caso não fosse encontrada uma solução para o problema, era incluído um novo átomo em S e iniciava-se uma nova iteração. O final do algoritmo podia ser atingido em duas situações: quando o método SAT retornasse uma solução ao problema (sucesso) ou quando não havia mais átomos a adicionar ao conjunto S (resultado indeterminado).

Nos próximos itens serão discutidos os pontos específicos de cada método.

5.3.6 S1

Para o S1, a inclusão de variáveis em S é feita de forma randômica, visto que Cadoli e Schaerf não definiram um método de inclusão de átomos. Para executar esta função de forma eficiente, o conjunto *notS* foi definido como uma lista ligada, contendo inicialmente uma ordenação randômica das variáveis do problema. Desta forma, a próxima variável a ser selecionada correspondia ao primeiro elemento da lista, que era então removido.

5.3.7 s1 Min

A estratégia de seleção da próxima variável a incluir no conjunto S visava minimizar o número de fórmulas satisfeitas automaticamente. Para efetuar esta seleção de forma eficiente, este algoritmo também utilizou a estrutura

FormulaMultiMap, indexando as fórmulas pelo número de literais verdadeiros que possuía. Porém, ao contrário do método GSAT Steepest Descent, não era necessário manter as fórmulas sob o índice zero.

Assim, partindo-se das fórmulas do índice 1, obtinha-se a lista das respectivas variáveis que as tornavam verdadeiras, sendo escolhida aquela que pertencia ao maior número de fórmulas (empates decididos aleatoriamente).

A partir do momento em que não houvesse mais fórmulas automaticamente verdadeiras, a escolha da próxima variável era feita de forma arbitrária.

5.3.8 s1 Max

Para seleção da variável que maximizasse o número de fórmulas automaticamente satisfeitas, utilizou-se um algoritmo semelhante ao do método s1 Min, com uma diferença: eram analisadas todas as variáveis do conjunto *notS*, não somente aquelas que tornavam verdadeiras as fórmulas do índice 1.

E, de forma análoga, a escolha da próxima variável passava a ser randômica caso não houvesse mais fórmulas automaticamente verdadeiras.

5.3.9 s1 Rnd

Visto que a escolha da próxima variável era aleatória, adotou-se uma estrutura de dados semelhante à do método S1 (conjunto *notS* implementado sobre uma lista ligada inicializada aleatoriamente).

6 Análise comparativa entre os métodos

Para comparar a performance dos métodos, foi utilizado o problema das N-rainhas, com N variando de 4 a 24 (todas instâncias satisfazíveis). Foi feita a tradução deste problema para fórmulas da seguinte maneira:

- Cada posição do tabuleiro foi associada a uma variável proposicional;
- Um valor verdadeiro para uma variável indica que existe uma rainha na posição correspondente;
- Para cada linha do tabuleiro, é incluída uma fórmula contendo uma disjunção de todas as variáveis desta linha. Com isso, garante-se que pelo menos uma rainha estará posicionada em cada linha;
- De forma análoga, cada coluna do tabuleiro é associada a uma fórmula contendo uma disjunção de suas respectivas variáveis;
- Por fim, são incluídas diversas disjunções de duas variáveis com sinal de negação para as posições na mesma linha, mesma coluna ou mesma diagonal, garantindo que não haverá rainhas em posições que se ataquem.

Para cada instância do problema, executou-se cada um dos algoritmos 100 vezes, obtendo-se o tempo mediano para comparação.

Conforme indicado na Seção 5.3.5 na página 41, para as execuções envolvendo lógicas aproximadas e o método GSAT Hill Climbing foi adotada uma parametrização específica para definição do número máximo de inversões do algoritmo GSAT. Devido à limitação de tempo, foram utilizados somente 8 fatores constantes distintos (0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 2.0 e 5.0). Nesta seção, só serão exibidos os resultados do melhor fator. Os dados completos podem ser visualizados no Apêndice A na página 52.

6.1 Tempo de execução

Nas Tabelas 6.1 e 6.2 estão indicados os tempos de execução de cada um dos 15 algoritmos, classificados da melhor para a pior performance.

Com base nos resultados obtidos, pôde-se concluir que:

- O método GSAT Hill Climbing apresentou um desempenho bastante superior aos demais métodos (conforme consta na Tabela 6.1 na próxima página);
- O método POSIT puro, que apresentava bom desempenho inicialmente, teve uma sensível queda de performance quando $N \geq 20$ (Tabela 6.2);
- Dos métodos ligados às lógicas de raciocínio aproximado, destacam-se S1 com GSAT Hill Climbing e o s*1 RND POSIT, que obtiveram um desempenho melhor que o método GSAT Steepest Descent;
- Por outro lado, o método s*1 MIN (aquele que *minimiza* o número de fórmulas automaticamente satisfeitas) gerou os piores tempos de execução.

6.2 Eficiência das heurísticas

Foram utilizados dois métodos para análise da eficiência das heurísticas:

- Tamanho médio do conjunto S no momento em que se encontrou uma solução, conforme indicado nas Tabelas 6.3 e 6.4. Quanto menor o tamanho do conjunto S, maior a *eficiência* da heurística.
- Comparação do tempo de execução dos métodos GSAT Steepest Descent, GSAT Hill Climbing e POSIT *com* e *sem* as heurísticas das lógicas de raciocínio aproximado, conforme indicado nas Tabelas 6.5, 6.6 e 6.7;

A partir dos resultados obtidos, concluiu-se que:

- O método S1 não provê nenhum ganho sobre os métodos POSIT e GSAT originais, pois, conforme visto na Tabela 6.4, sempre foi necessário completar o conjunto S com 100% das variáveis do problema. Esta conclusão também se reflete nos tempos de execução: o método S1 sempre foi mais lento que o método SAT original correspondente;
- Já o método s*1 MAX foi o mais *eficiente*, pois obteve soluções preenchendo-se o conjunto S com cerca de 1/3 do total de variáveis (Tabela 6.3 na página 48). Vale lembrar que este é o método que adiciona

N	GSAT HC	S1 GSAT HC (0.7)	s*1 RND POSIT	GSAT SD	s*1 MAX POSIT	s*1 MAX GSAT SD	s*1 RND GSAT HC (0.7)
4	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,01s	0,01s	0,02s	0,01s	0,01s	0,01s	0,01s
6	0,02s	0,02s	0,05s	0,03s	0,05s	0,06s	0,04s
7	0,03s	0,04s	0,06s	0,04s	0,06s	0,05s	0,05s
8	0,04s	0,06s	0,12s	0,07s	0,11s	0,11s	0,10s
9	0,06s	0,08s	0,17s	0,11s	0,20s	0,19s	0,17s
10	0,09s	0,14s	0,31s	0,20s	0,38s	0,42s	0,34s
11	0,12s	0,20s	0,46s	0,32s	0,56s	0,73s	0,52s
12	0,15s	0,29s	0,64s	0,48s	0,84s	0,95s	0,93s
13	0,22s	0,40s	0,95s	0,70s	1,17s	1,42s	1,22s
14	0,29s	0,61s	1,42s	1,01s	1,73s	2,15s	1,95s
15	0,36s	0,94s	1,83s	1,38s	2,31s	2,84s	3,01s
16	0,44s	1,17s	2,48s	1,93s	3,16s	4,31s	4,28s
17	0,57s	1,58s	3,34s	2,70s	4,13s	6,27s	6,52s
18	0,69s	2,06s	4,39s	3,64s	5,56s	8,55s	8,60s
19	0,86s	3,03s	5,47s	5,13s	7,42s	10,11s	12,12s
20	1,04s	3,94s	6,81s	6,56s	9,34s	15,37s	17,62s
21	1,19s	5,26s	9,56s	8,71s	12,67s	19,08s	19,91s
22	1,39s	5,85s	12,87s	11,32s	15,67s	26,16s	29,02s
23	1,56s	7,30s	13,87s	14,74s	19,79s	34,92s	35,26s
24	2,04s	9,37s	17,88s	18,53s	25,71s	43,02s	45,76s

Tabela 6.1: Resultados dos melhores métodos sobre o problema das N-Rainhas

Obs. 1: HC = Hill Climbing; SD = Steepest Descent

Obs. 2: Os números entre parênteses indicam o fator constante aplicado para o número máximo de inversões.

N	s*1 MAX GSAT HC (1.1)	s*1 RND GSAT SD	S1 GSAT SD	s*1 MIN GSAT HC (0.7)	POSIT	S1 POSIT	s*1 MIN GSAT SD	s*1 MIN POSIT (*)
4	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,01s	0,02s	0,02s	0,02s	0,01s	0,02s	0,03s	0,03s
6	0,05s	0,07s	0,05s	0,05s	0,02s	0,02s	0,09s	0,06s
7	0,06s	0,08s	0,09s	0,10s	0,03s	0,04s	0,22s	0,12s
8	0,11s	0,20s	0,16s	0,18s	0,04s	0,06s	0,47s	0,25s
9	0,16s	0,32s	0,32s	0,36s	0,06s	0,08s	1,10s	0,49s
10	0,37s	0,73s	0,58s	0,66s	0,09s	0,11s	2,59s	0,89s
11	0,62s	1,14s	1,02s	1,26s	0,12s	0,15s	5,13s	1,88s
12	0,90s	1,55s	1,87s	2,21s	0,16s	0,20s	9,68s	3,43s
13	1,46s	2,22s	2,76s	3,63s	0,20s	0,26s	17,02s	7,03s
14	2,15s	3,40s	4,21s	5,88s	0,29s	0,37s	30,74s	19,45s
15	3,27s	5,38s	7,69s	9,36s	0,38s	0,49s	51,36s	57,88s
16	4,17s	6,49s	9,24s	15,00s	0,61s	0,76s	83,69s	-
17	6,46s	9,38s	12,70s	24,04s	1,17s	1,35s	137,81s	-
18	8,17s	12,07s	17,94s	34,28s	1,92s	2,11s	210,40s	-
19	11,79s	15,76s	33,35s	50,52s	1,30s	1,58s	343,61s	-
20	15,49s	21,28s	45,40s	76,44s	7,58s	7,89s	498,94s	-
21	21,61s	27,68s	60,96s	103,73s	22,77s	23,18s	804,61s	-
22	29,93s	35,47s	86,33s	144,46s	203,68s	202,60s	1126,76s	-
23	38,88s	40,88s	114,24s	200,85s	655,64s	652,15s	1686,92s	-
24	47,59s	51,90s	150,43s	277,44s	238,29s	433,73s	2532,72s	-

Tabela 6.2: Resultados dos métodos de pior performance sobre o problema das N-Rainhas

Obs. 1: HC = Hill Climbing; SD = Steepest Descent

Obs. 2: Os números entre parênteses indicam o fator constante aplicado para o número máximo de inversões.

(*) O método s*1 MIN POSIT apresentou um tempo de execução extremamente elevado a partir de 16 rainhas (cerca de 1 dia para executar uma única instância), de forma que seus tempos foram desconsiderados.

N	Total variáveis	s*1 MAX			s*1 RND		
		POSIT	GSAT SD	GSAT HC	POSIT	GSAT SD	GSAT HC
4	16	5 (32%)	5 (33%)	5 (34%)	12 (75%)	12 (76%)	14 (87%)
5	25	6 (26%)	7 (27%)	7 (27%)	15 (61%)	16 (64%)	18 (72%)
6	36	19 (53%)	21 (59%)	24 (66%)	26 (73%)	29 (80%)	32 (89%)
7	49	18 (36%)	18 (37%)	20 (41%)	29 (59%)	30 (61%)	34 (70%)
8	64	22 (35%)	26 (41%)	27 (43%)	37 (58%)	41 (64%)	46 (71%)
9	81	27 (34%)	30 (37%)	28 (35%)	43 (54%)	48 (60%)	54 (66%)
10	100	37 (37%)	42 (42%)	44 (44%)	53 (53%)	61 (61%)	71 (71%)
11	121	42 (34%)	51 (42%)	52 (43%)	61 (51%)	71 (58%)	81 (67%)
12	144	46 (32%)	54 (37%)	55 (38%)	67 (46%)	80 (55%)	95 (66%)
13	169	51 (30%)	61 (36%)	62 (37%)	74 (44%)	87 (51%)	104 (61%)
14	196	57 (29%)	69 (35%)	72 (37%)	84 (43%)	98 (50%)	120 (61%)
15	225	60 (27%)	74 (33%)	82 (36%)	90 (40%)	111 (49%)	133 (59%)
16	256	66 (26%)	86 (34%)	86 (34%)	97 (38%)	118 (46%)	147 (57%)
17	289	71 (25%)	95 (33%)	96 (33%)	104 (36%)	132 (46%)	161 (56%)
18	324	75 (23%)	106 (33%)	104 (32%)	114 (35%)	140 (43%)	174 (54%)
19	361	81 (22%)	106 (29%)	114 (32%)	120 (33%)	152 (42%)	190 (53%)
20	400	87 (22%)	122 (31%)	121 (30%)	128 (32%)	165 (41%)	208 (52%)
21	441	93 (21%)	126 (29%)	130 (30%)	134 (30%)	175 (40%)	215 (49%)
22	484	95 (20%)	137 (28%)	143 (29%)	142 (29%)	189 (39%)	235 (49%)
23	529	102 (19%)	152 (29%)	152 (29%)	148 (28%)	197 (37%)	251 (47%)
24	576	104 (18%)	158 (27%)	156 (27%)	156 (27%)	212 (37%)	264 (46%)
	Média	29%	35%	36%	45%	52%	62%

Tabela 6.3: Métodos s*1 MAX e s*1 RND: número médio de variáveis incluídas no conjunto S para obtenção da solução

N	Total variáveis	s*1 MIN			S1		
		POSIT	GSAT SD	GSAT HC	POSIT	GSAT SD	GSAT HC
4	16	13 (83%)	14 (87%)	14 (90%)	16 (100%)	16 (100%)	16 (100%)
5	25	20 (78%)	20 (81%)	22 (87%)	25 (100%)	25 (100%)	25 (100%)
6	36	29 (81%)	32 (89%)	34 (95%)	36 (100%)	36 (100%)	36 (100%)
7	49	38 (78%)	40 (82%)	43 (87%)	49 (100%)	49 (100%)	49 (100%)
8	64	52 (81%)	54 (84%)	57 (89%)	64 (100%)	64 (100%)	64 (100%)
9	81	63 (78%)	66 (82%)	70 (87%)	81 (100%)	81 (100%)	81 (100%)
10	100	78 (78%)	83 (83%)	88 (88%)	100 (100%)	100 (100%)	100 (100%)
11	121	92 (76%)	99 (82%)	106 (87%)	121 (100%)	121 (100%)	121 (100%)
12	144	109 (76%)	117 (81%)	124 (86%)	144 (100%)	144 (100%)	144 (100%)
13	169	126 (74%)	138 (81%)	144 (85%)	169 (100%)	169 (100%)	169 (100%)
14	196	150 (77%)	157 (80%)	166 (85%)	196 (100%)	196 (100%)	196 (100%)
15	225	167 (74%)	181 (80%)	188 (84%)	225 (100%)	225 (100%)	225 (100%)
16	256	-	207 (81%)	215 (84%)	256 (100%)	256 (100%)	256 (100%)
17	289	-	228 (79%)	240 (83%)	289 (100%)	289 (100%)	289 (100%)
18	324	-	258 (80%)	269 (83%)	324 (100%)	324 (100%)	324 (100%)
19	361	-	285 (79%)	297 (82%)	361 (100%)	361 (100%)	361 (100%)
20	400	-	313 (78%)	330 (83%)	400 (100%)	400 (100%)	400 (100%)
21	441	-	350 (79%)	360 (82%)	441 (100%)	441 (100%)	441 (100%)
22	484	-	379 (78%)	394 (81%)	484 (100%)	484 (100%)	484 (100%)
23	529	-	416 (79%)	430 (81%)	529 (100%)	529 (100%)	529 (100%)
24	576	-	451 (78%)	469 (81%)	576 (100%)	576 (100%)	576 (100%)
	Média	78%	81%	85%	100%	100%	100%

Tabela 6.4: Métodos s*1 MIN e S1: número médio de variáveis incluídas no conjunto S para obtenção da solução

GSAT Hill Climbing					
	Método original	S1	s*1 RND	s*1 MAX	s*1 MIN
4	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,01s	0,01s	0,01s	0,01s	0,02s
6	0,02s	0,02s	0,04s	0,05s	0,05s
7	0,03s	0,04s	0,05s	0,06s	0,10s
8	0,04s	0,06s	0,10s	0,11s	0,18s
9	0,06s	0,08s	0,17s	0,16s	0,36s
10	0,09s	0,14s	0,34s	0,37s	0,66s
11	0,12s	0,20s	0,52s	0,62s	1,26s
12	0,15s	0,29s	0,93s	0,90s	2,21s
13	0,22s	0,40s	1,22s	1,46s	3,63s
14	0,29s	0,61s	1,95s	2,15s	5,88s
15	0,36s	0,94s	3,01s	3,27s	9,36s
16	0,44s	1,17s	4,28s	4,17s	15,00s
17	0,57s	1,58s	6,52s	6,46s	24,04s
18	0,69s	2,06s	8,60s	8,17s	34,28s
19	0,86s	3,03s	12,12s	11,79s	50,52s
20	1,04s	3,94s	17,62s	15,49s	76,44s
21	1,19s	5,26s	19,91s	21,61s	103,73s
22	1,39s	5,85s	29,02s	29,93s	144,46s
23	1,56s	7,30s	35,26s	38,88s	200,85s
24	2,04s	9,37s	45,76s	47,59s	277,44s

Tabela 6.5: Resultados do método GSAT Hill Climbing com e sem heurísticas

GSAT Steepest Descent					
	Método original	s*1 MAX	s*1 RND	S1	s*1 MIN
4	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,01s	0,01s	0,02s	0,02s	0,03s
6	0,03s	0,06s	0,07s	0,05s	0,09s
7	0,04s	0,05s	0,08s	0,09s	0,22s
8	0,07s	0,11s	0,20s	0,16s	0,47s
9	0,11s	0,19s	0,32s	0,32s	1,10s
10	0,20s	0,42s	0,73s	0,58s	2,59s
11	0,32s	0,73s	1,14s	1,02s	5,13s
12	0,48s	0,95s	1,55s	1,87s	9,68s
13	0,70s	1,42s	2,22s	2,76s	17,02s
14	1,01s	2,15s	3,40s	4,21s	30,74s
15	1,38s	2,84s	5,38s	7,69s	51,36s
16	1,93s	4,31s	6,49s	9,24s	83,69s
17	2,70s	6,27s	9,38s	12,70s	137,81s
18	3,64s	8,55s	12,07s	17,94s	210,40s
19	5,13s	10,11s	15,76s	33,35s	343,61s
20	6,56s	15,37s	21,28s	45,40s	498,94s
21	8,71s	19,08s	27,68s	60,96s	804,61s
22	11,32s	26,16s	35,47s	86,33s	1126,76s
23	14,74s	34,92s	40,88s	114,24s	1686,92s
24	18,53s	43,02s	51,90s	150,43s	2532,72s

Tabela 6.6: Resultados do método GSAT Steepest Descent com e sem heurísticas

POSIT					
	s*1 RND	s*1 MAX	Método original	S1	s*1 MIN
4	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,02s	0,01s	0,01s	0,02s	0,03s
6	0,05s	0,05s	0,02s	0,02s	0,06s
7	0,06s	0,06s	0,03s	0,04s	0,12s
8	0,12s	0,11s	0,04s	0,06s	0,25s
9	0,17s	0,20s	0,06s	0,08s	0,49s
10	0,31s	0,38s	0,09s	0,11s	0,89s
11	0,46s	0,56s	0,12s	0,15s	1,88s
12	0,64s	0,84s	0,16s	0,20s	3,43s
13	0,95s	1,17s	0,20s	0,26s	7,03s
14	1,42s	1,73s	0,29s	0,37s	19,45s
15	1,83s	2,31s	0,38s	0,49s	57,88s
16	2,48s	3,16s	0,61s	0,76s	-
17	3,34s	4,13s	1,17s	1,35s	-
18	4,39s	5,56s	1,92s	2,11s	-
19	5,47s	7,42s	1,30s	1,58s	-
20	6,81s	9,34s	7,58s	7,89s	-
21	9,56s	12,67s	22,77s	23,18s	-
22	12,87s	15,67s	203,68s	202,60s	-
23	13,87s	19,79s	655,64s	652,15s	-
24	17,88s	25,71s	238,29s	433,73s	-

Tabela 6.7: Resultados do método POSIT com e sem heurísticas

variáveis ao conjunto S de forma a *maximizar* o número de fórmulas automaticamente satisfeitas;

- As lógicas de raciocínio aproximado $s1$ e $S1$ não proporcionaram nenhum ganho sobre os métodos GSAT originais (Steepest Descent e Hill Climbing), como pode-se ver nas Tabelas 6.6 e 6.5;
- Já o método POSIT teve um melhor desempenho quando associado aos métodos $s*1$ RND e $s*1$ MAX (vide Tabela 6.7 na página atual).

Vale dizer que estas conclusões são válidas para o problema das N-Rainhas, devendo ser estendidas para outras instâncias do problema SAT.

7 Conclusão

Neste trabalho, estudamos o problema SAT e alguns dos métodos existentes para sua resolução. A partir dos estudos relacionados com as lógicas de raciocínio aproximado S1 [SC95] e s1 [FW02], desenvolvemos novas heurísticas para redução do espaço de busca de um problema, para uma posterior verificação com um método SAT já desenvolvido. As heurísticas foram denominadas S1 e s1, correspondendo às lógicas de raciocínio aproximado.

Para avaliação da eficiência destas heurísticas, foram utilizados os métodos GSAT e POSIT, executados sobre o problema das N-Rainhas. Se por um lado não obtivemos nenhum ganho sobre o GSAT, para o POSIT foram atingidas melhoras significativas no tempo de processamento. Além disso, o método S1 provou-se ineficaz por não propiciar em nenhum caso uma redução do espaço de busca.

O prosseguimento deste trabalho aponta para várias direções:

- Analisar o desempenho destas heurísticas para outras instâncias do problema SAT, como, por exemplo, para fórmulas aleatórias difíceis obtidas no SATLIB [HS00], de forma a obter conclusões mais gerais e precisas;
- Integrar estas heurísticas com outros métodos completos e incompletos de melhor performance (como o zChaff [MMZ⁺01] e o MLD [SW98]);
- Otimizar os métodos através de uma revisão das estruturas de dados e dos algoritmos. Como exemplo, pode-se citar o algoritmo de *observação de duas variáveis*, descrito no método Chaff [MMZ⁺01], que pode ser aplicado para o controle de fórmulas automaticamente satisfeitas da heurística s1;
- Estudar um novo algoritmo para inclusão de variáveis no conjunto S para as heurísticas S1 e s1, visando a obtenção do conjunto S minimal para a resolução do problema e, conseqüentemente, a maior redução do espaço de busca;
- Analisar o melhor fator para o número máximo de inversões para ambos os métodos GSAT (Hill Climbing e Steepest Descent).

A Resultados

N	Fator							
	0,7	0,8	0,9	1,0	1,1	1,2	2,0	5,0
4	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,02s
6	0,02s	0,03s	0,03s	0,03s	0,03s	0,03s	0,04s	0,06s
7	0,04s	0,04s	0,04s	0,04s	0,04s	0,04s	0,06s	0,11s
8	0,06s	0,06s	0,06s	0,07s	0,07s	0,07s	0,10s	0,22s
9	0,08s	0,09s	0,09s	0,10s	0,11s	0,11s	0,17s	0,38s
10	0,14s	0,15s	0,16s	0,17s	0,19s	0,21s	0,31s	0,65s
11	0,20s	0,23s	0,25s	0,28s	0,28s	0,34s	0,47s	1,07s
12	0,29s	0,32s	0,36s	0,38s	0,46s	0,47s	0,75s	1,74s
13	0,40s	0,45s	0,56s	0,63s	0,79s	0,70s	1,47s	3,56s
14	0,61s	0,70s	0,84s	1,00s	1,04s	1,24s	2,43s	6,63s
15	0,94s	1,10s	1,25s	1,32s	1,63s	1,90s	3,00s	7,17s
16	1,17s	1,51s	1,72s	2,18s	2,54s	2,63s	5,42s	10,87s
17	1,58s	2,01s	2,38s	2,90s	3,50s	3,95s	7,68s	15,40s
18	2,06s	2,88s	3,51s	4,07s	4,93s	5,56s	10,79s	30,90s
19	3,03s	3,81s	4,61s	4,56s	6,50s	7,24s	14,56s	41,44s
20	3,94s	4,80s	6,02s	6,97s	8,57s	9,58s	19,26s	55,46s
21	5,26s	6,17s	7,36s	9,30s	10,70s	12,62s	24,93s	71,83s
22	5,85s	8,01s	10,00s	12,13s	14,10s	16,31s	32,53s	93,73s
23	7,30s	10,34s	12,48s	15,11s	17,71s	25,06s	42,50s	118,23s
24	9,37s	13,17s	16,38s	19,07s	22,49s	25,36s	52,01s	151,11s

Tabela A.8: Tempos de execução do método S1 com GSAT Steepest Descent, para cada fator.

Neste caso, o melhor fator é o 0,7.

N	Fator							
	0,7	0,8	0,9	1,0	1,1	1,2	2,0	5,0
4	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s
6	0,04s	0,04s	0,05s	0,04s	0,05s	0,05s	0,06s	0,10s
7	0,06s	0,05s	0,05s	0,04s	0,06s	0,06s	0,06s	0,11s
8	0,12s	0,11s	0,10s	0,10s	0,11s	0,11s	0,11s	0,23s
9	0,19s	0,18s	0,16s	0,18s	0,16s	0,19s	0,23s	0,35s
10	0,42s	0,37s	0,38s	0,38s	0,37s	0,41s	0,51s	0,82s
11	0,68s	0,62s	0,62s	0,63s	0,62s	0,69s	0,83s	1,40s
12	1,01s	0,97s	1,00s	0,99s	0,90s	1,06s	1,32s	1,95s
13	1,56s	1,66s	1,53s	1,39s	1,46s	1,50s	1,73s	3,01s
14	2,46s	2,26s	2,11s	2,12s	2,15s	1,98s	2,67s	3,86s
15	3,60s	3,31s	3,25s	3,08s	3,27s	3,15s	3,83s	5,66s
16	5,14s	4,37s	4,77s	4,66s	4,17s	4,68s	5,06s	8,09s
17	7,47s	6,68s	6,14s	6,16s	6,46s	6,67s	6,66s	11,22s
18	10,97s	9,40s	9,28s	8,96s	8,17s	8,27s	9,39s	15,53s
19	13,55s	12,83s	12,86s	12,54s	11,79s	11,75s	13,97s	21,03s
20	18,93s	18,48s	16,82s	15,76s	15,49s	16,04s	17,44s	29,02s
21	27,35s	24,29s	23,49s	21,09s	21,61s	22,69s	24,56s	38,41s
22	35,73s	32,90s	33,31s	31,01s	29,93s	27,66s	31,58s	45,83s
23	45,46s	42,61s	39,11s	39,28s	38,88s	39,83s	42,41s	67,97s
24	58,14s	56,09s	54,97s	52,69s	47,59s	54,45s	62,14s	84,74s

Tabela A.9: Tempos de execução do método s*1 MAX com GSAT Steepest Descent, para cada fator.

Note que neste caso há uma convergência de tempos para o fator 1,1.

N	Fator							
	0,7	0,8	0,9	1,0	1,1	1,2	2,0	5,0
4	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,01s	0,01s	0,02s	0,02s	0,01s	0,02s	0,02s	0,02s
6	0,04s	0,04s	0,04s	0,04s	0,05s	0,05s	0,06s	0,12s
7	0,05s	0,05s	0,06s	0,06s	0,06s	0,06s	0,09s	0,18s
8	0,10s	0,10s	0,12s	0,12s	0,12s	0,13s	0,19s	0,40s
9	0,17s	0,18s	0,19s	0,20s	0,22s	0,25s	0,31s	0,72s
10	0,34s	0,38s	0,34s	0,38s	0,42s	0,48s	0,63s	1,56s
11	0,52s	0,63s	0,56s	0,65s	0,68s	0,72s	1,10s	2,48s
12	0,93s	0,90s	0,95s	0,92s	1,03s	1,08s	1,62s	3,65s
13	1,22s	1,39s	1,38s	1,39s	1,60s	1,69s	2,52s	5,62s
14	1,95s	2,28s	2,45s	2,38s	2,17s	2,54s	3,63s	8,82s
15	3,01s	2,92s	3,09s	3,59s	3,74s	4,13s	5,64s	11,75s
16	4,28s	4,42s	4,60s	4,92s	4,87s	5,40s	8,29s	17,59s
17	6,52s	6,18s	7,14s	7,17s	7,57s	7,16s	10,97s	22,82s
18	8,60s	9,78s	9,82s	9,39s	10,68s	10,76s	16,88s	31,66s
19	12,12s	12,19s	14,39s	12,80s	13,64s	15,63s	22,97s	49,73s
20	17,62s	17,07s	16,72s	18,57s	19,57s	20,43s	32,13s	62,93s
21	19,91s	22,77s	24,46s	23,98s	25,35s	27,17s	42,65s	81,19s
22	29,02s	29,77s	30,26s	35,03s	33,47s	39,23s	57,75s	113,45s
23	35,26s	38,82s	42,77s	41,35s	48,05s	47,61s	69,33s	133,91s
24	45,76s	46,19s	51,45s	55,36s	56,78s	64,16s	87,22s	165,41s

Tabela A.10: Tempos de execução do método s*1 RND com GSAT Steepest Descent, para cada fator.

Neste caso, o melhor fator é o 0,7.

N	Fator							
	0,7	0,8	0,9	1,0	1,1	1,2	2,0	5,0
4	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s	0,01s
5	0,02s	0,02s	0,02s	0,02s	0,02s	0,02s	0,03s	0,04s
6	0,05s	0,05s	0,05s	0,06s	0,06s	0,06s	0,08s	0,15s
7	0,10s	0,10s	0,10s	0,11s	0,12s	0,12s	0,18s	0,33s
8	0,18s	0,20s	0,22s	0,24s	0,25s	0,28s	0,45s	0,92s
9	0,36s	0,40s	0,44s	0,49s	0,51s	0,57s	0,87s	2,06s
10	0,66s	0,73s	0,80s	0,94s	1,01s	1,07s	1,78s	4,29s
11	1,26s	1,38s	1,64s	1,83s	2,04s	2,18s	3,50s	8,63s
12	2,21s	2,46s	2,87s	3,10s	3,57s	3,88s	6,52s	15,49s
13	3,63s	4,17s	4,94s	5,36s	5,99s	6,49s	11,20s	30,15s
14	5,88s	6,89s	8,28s	9,22s	10,62s	11,24s	19,49s	53,04s
15	9,36s	11,47s	13,76s	15,34s	17,36s	18,69s	35,50s	90,32s
16	15,00s	18,40s	22,03s	25,36s	28,74s	31,49s	59,08s	149,32s
17	24,04s	27,84s	33,73s	41,23s	45,03s	52,39s	96,08s	261,03s
18	34,28s	43,52s	52,42s	59,50s	70,19s	80,31s	154,26s	400,57s
19	50,52s	65,26s	81,89s	96,52s	107,43s	123,28s	229,30s	647,37s
20	76,44s	95,60s	112,96s	138,92s	161,11s	178,94s	326,30s	941,04s
21	103,73s	130,89s	167,13s	195,79s	228,15s	262,36s	502,80s	1379,05s
22	144,46s	195,99s	242,08s	279,43s	328,82s	371,16s	703,95s	2012,42s
23	200,85s	255,94s	323,69s	386,08s	450,71s	514,93s	949,22s	2765,15s
24	277,44s	347,24s	414,54s	522,63s	615,02s	690,77s	1364,18s	3526,60s

Tabela A.11: Tempos de execução do método s*1 MIN com GSAT Steepest Descent, para cada fator.

Neste caso, o melhor fator é o 0,7.

Referências

- [CA96] James M. Crawford and Larry D. Auton. *Experimental results on the crossover point in random 3-SAT*. Artif. Intell. **81** (1996). no. 1-2. 31–57.
- [Cook71] Stephen A. Cook. *The complexity of theorem-proving procedures*. Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (New York). 1971. pp. 151–158.
- [DABC96] O. Dubois, P. André, Y. Boufkhad and J. Carlier. *SAT versus UNSAT*. DIMACS **26** (1996). 415–436.
- [DLL62] Martin Davis, George Logemann and Donald Loveland. *A Machine Program for Theorem Proving*. Communications of the ACM **5** (1962). no. 7. 394–397.
- [DP60] Martin Davis and Hilary Putnam. *A computing procedure for quantification theory*. Journal of the Association for Computing Machinery **7** (1960). no. 3. 201–215.
- [Free95] Jon W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. Ph.D. thesis. University of Pennsylvania. Philadelphia. 1995.
- [FW01] Marcelo Finger and Renata Wassermann. *Tableaux for Approximate Reasoning*. IJCAI-2001 Workshop on Inconsistency in Data and Knowledge. 2001. pp. 71–79.
- [FW02] Marcelo Finger and Renata Wassermann. *Logics for Approximate Reasoning: Approximating Classical Logic 'From Above'*. Brazilian Symposium on Artificial Intelligence. 2002. pp. 21–30.
- [GPFW97] J. Gu, P. Purdom, J. Franco and B. Wah. *Algorithms for the Satisfiability (SAT) Problem: a Survey*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science **35** (1997). 19–152.
- [HS00] Holger H. Hoos and Thomas Stützle. *SATLIB: An Online Resource for Research on SAT*. Frontiers in Artificial Intelligence and Applications. vol. 63. IOS Press. 2000. pp. 283–292.

- [MMZ⁺01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang and Sharad Malik. *Chaff: engineering an efficient SAT solver*. Proceedings of the 38th conference on Design automation. ACM Press. 2001. pp. 530–535.
- [MSS96] Joao P. Marques-Silva and Karem A. Sakallah. *GRASP - A New Search Algorithm for Satisfiability*. Proceedings of IEEE/ACM International Conference on Computer-Aided Design. November 1996. pp. 220–227.
- [Pret93] D. Pretolani. *Efficiency and stability of hypergraph SAT algorithms*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. vol. 26. 1993. pp. 479–498.
- [RSA78] R. L. Rivest, A. Shamir and L. M. Adelman. *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM **21** (1978). no. 2. 120–126.
- [SC95] Marco Schaerf and Marco Cadoli. *Tractable Reasoning via Approximation*. Artificial Intelligence **74** (1995). no. 2. 249–310.
- [SKC96] Bart Selman, Henry A. Kautz and Bram Cohen. *Local Search Strategies for Satisfiability Testing*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. vol. 26. 1996. pp. 521–532.
- [SLM92] Bart Selman, Hector J. Levesque and D. Mitchell. *A New Method for Solving Hard Satisfiability Problems*. Proceedings of the Tenth National Conference on Artificial Intelligence (Menlo Park, California) (Paul Rosenbloom and Peter Szolovits, eds.). AAAI Press. 1992. pp. 440–446.
- [SW98] Yi Shang and Benjamin W. Wah. *A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems*. J. of Global Optimization **12** (1998). no. 1. 61–99.
- [ZM88] R. Zabih and D. McAllester. *A Rearrangement Search Strategy for Determining Propositional Satisfiability*. Proc. of AAAI-88 (St. Paul, MN). 1988. pp. 155–160.