MAC 5701 – Tópicos em Ciência da Computação
Relatório de Estudo

Lógicas de Raciocínio Aproximado e o Problema SAT

Aluno: Fabio Luiz Ide (fli@ime.usp.br)

Orientador: Marcelo Finger (mfinger@ime.usp.br)

25 de junho de 2002

1 – Introdução

O Problema da Satisfatibilidade (SAT) foi o primeiro problema considerado NP-completo, conforme provado por Cook em 1971. Trata-se de um problema de decisão, em que cada instância é definida como uma expressão booleana contendo apenas os conectores *AND*, *OR*, *NOT*, variáveis e parênteses. A questão é verificar se existe alguma valoração para as variáveis que torne toda a expressão verdadeira.

Uma outra abordagem para resolução do problema SAT é o uso de raciocínio aproximado. Cadoli e Schaerf propuseram um método de raciocínio aproximado para resolução de problemas, definindo duas lógicas, denominadas ***S3*** e ***S1*** [CS95]. Enquanto a lógica ***S3*** serviria de limitante inferior para a lógica clássica, a lógica ***S1*** seria seu limitante superior. Este método, porém, restringe-se somente 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.

O objetivo inicial deste trabalho seria aplicar raciocínio semelhante para a lógica ***S1***, tentando também estabelecer um método incremental de aproximação da lógica ***S1*** para a lógica clássica. Devido a algumas limitações na implementação da lógica ***S1***, Finger e Wassermann em [FW02] definem uma nova lógica, chamada de ***s1***, cuja implementação será o objetivo final do trabalho.

2 – O algoritmo GSAT

Em 1992 Selman, Levesque e Mitchel publicaram um método para resolução do problema SAT [SLM92]. O novo método recebeu o nome de GSAT. Ele se baseia num algoritmo guloso que busca uma valoração que satisfaça o conjunto de fórmulas dado. O algoritmo só considera fórmulas na forma clausal.

O algoritmo GSAT é correto, porém incompleto, pois não pode provar se uma fórmula não é satisfazível.

```
procedure GSAT
Entrada: conjunto de fórmulas na forma clausal Γ, inteiros MAX-FLIPS e
         MAX-TRIES.
Saída:   uma valoração T, que satisfaz o conjunto de fórmulas Γ, se
         encontrada.
begin
   for i := 1 to MAX-TRIES
      T := uma valoração randômica para as variáveis;
      for j := 1 to MAX-FLIPS
         if T satisfaz Γ then return T;
         p := variável cuja alteração no seu valor forneça o maior
              aumento no número de  fórmulas de Γ satisfeitas por T;
         T := T com o valor de p invertido;
      end for;
   end for;
   return "valoração não encontrada";
end;
```

O GSAT inicia a resolução do problema gerando uma valoração randômica para as variáveis das cláusulas. A cada passo, ele tenta escolher a variável cuja alteração no valor maximize o número de cláusulas satisfeitas. Este alteração recebe o nome de “flip”. Caso haja mais de uma variável, ela é escolhida aleatoriamente. O algoritmo prossegue até que uma resposta tenha sido encontrada, ou o número máximo de flips, definido pelo parâmetro MAX-FLIPS, tenha sido atingido. Neste caso, gera-se novamente uma valoração randômica inicial e repete-se o procedimento até no máximo o número de tentativas definido pelo parâmetro MAX-TRIES.

No momento de escolher a variável para o “flip”, existem situações em que qualquer variável escolhida diminui o número de cláusulas satisfeitas. Neste caso temos um “movimento para baixo”. Caso o número de cláusulas satisfeitas seja mantido, temos um “movimento lateral”. Conforme indicado em [SLM92], a existência destes movimentos é essencial para o sucesso do algoritmo. Uma implementação que não considerasse estes movimentos nem sempre é bem sucedida em achar uma solução, até mesmo para problemas fáceis.

3 – Lógicas aproximadas

Ao longo desta e das próximas seções, utilizaremos a seguinte notação:

Seja P o conjunto enumerável de todas as variáveis proposicionais. Usaremos letras minúsculas para denotar os elementos de P: p, q, r, s, t.

Usaremos ainda letras gregas minúsculas para denotar fórmulas (α, β) e letras gregas maiúsculas para denotar conjuntos de fórmulas (Γ, Λ).

Cadoli e Schaerf propuseram um método de raciocínio aproximado para resolução de problemas, quando a obtenção da solução completa exigiria muitos recursos [CS95]. Este método define duas lógicas, ***S1*** e ***S3***, que serviriam de limites superior e inferior para a lógica clássica e seriam mais fáceis para calcular. O método considera apenas fórmulas na forma clausal.

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

Para uma valoração υ em ***S3*** temos as possibilidades abaixo:

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

Para um conjunto de fórmulas Γ e uma fórmula α, dizemos que implica logicamente Γ 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 υ em ***S1*** temos as possibilidades abaixo:

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

Para um conjunto de fórmulas Γ e uma fórmula α, dizemos que implica logicamente Γ 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, podemos ver que se $S = P$ tanto ***S1*** como ***S3*** se comportam como a lógica clássica.

Cadoli e Schaerf provam em [CS95] que $\models_S^1$ é completa porém incorreta e desta forma pode ser utilizada para aproximar $\not\models$. Isto significa que se para algum S temos que $\Gamma \not\models_S^1 \alpha$, então $\Gamma \not\models \alpha$.

Por outro lado, temos que $\models_S^3$ é incompleta porém correta e pode ser utilizada para aproximar $\models$. Assim, se para algum S temos que $\Gamma \models_S^3 \alpha$, então $\Gamma \models \alpha$.

Os exemplos a seguir ilustram o uso destas lógicas:

Exemplo 3.1 ([CS95]) Considere

Γ = (¬*cow* ∨ *grass-eater*),

(¬*dog* ∨ *carnivore*),

(¬*grass-eater* ∨ ¬*has-canine-teeth*),

(¬*grass-eater* ∨ *mammal*),

(¬*carnivore* ∨ *mammal*),

(¬*mammal* ∨ *has-canine-teeth* ∨ *has-molar-teeth*),

(¬*mammal* ∨ *vertebrate*),

(¬*vertebrate* ∨ *animal*).

α = (¬*cow* ∨ *has-molar-teeth*)

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

Exemplo 3.2 ([CS95]) Considere

Γ = (¬*person* ∨ *child* ∨ *youngster* ∨ *adult* ∨ *senior*),

(¬*youngster* ∨ *student* ∨ *worker*),

(¬*adult* ∨ *student* ∨ *worker* ∨ *unemployed*),

(¬*senior* ∨ *pensioner* ∨ *worker*),

(¬*student* ∨ *child* ∨ *youngster* ∨ *adult*),

(¬*pensioner* ∨ *senior*),

(¬*pensioner* ∨ ¬*student*),

(¬*pensioner* ∨ ¬*worker*).

α = (¬*child* ∨ *pensioner*)

Queremos provar que $\Gamma \not\models \alpha$. Para S = {*child*, *worker*, *pensioner*}, temos que $\Gamma \not\models^1_S \alpha$ e, portanto, $\Gamma \not\models \alpha$.

Cadoli e Schaerf descrevem um algoritmo incremental para aproximação da lógica clássica a partir de ***S1*** e ***S3*** efetuando simplificações sobre os conjuntos de fórmulas, conforme descrito a seguir (para um conjunto de fórmulas Γ, uma conclusão α e um conjunto S de átomos):

Lema 3.1 [CS95] *Seja simplifica-1(Γ, S) o resultado de remover todos os literais de Γ que correspondam a variáveis que não estão em S. Γ é S1-satisfazível se e somente se simplifica-1(Γ, S) é classicamente satisfazível.*

Obs.: Para o caso em que todos os literais de uma fórmula sejam removidos, a fórmula em questão assumirá sempre o valor **falso**.

Teorema 3.2 [CS95] *Seja $\alpha = \alpha_S \vee \alpha_{S'}$, onde α_S contém somente átomos de α que pertençam a S, e nenhum dos átomos de $\alpha_{S'}$ pertence a S. Então, $\Gamma \models^1_S \alpha$ se e somente se $\Gamma \cup \{\neg\alpha_S\}$ não é S1-satisfazível.*

Lema 3.3 [CS95] *Seja simplifica-3(Γ, 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 simplifica-3(Γ, S) é classicamente satisfazível.*

Teorema 3.4 [CS95] *Se todos os átomos de α pertencem a S, então temos que $\Gamma \models^3_S \alpha$ se e somente se $\Gamma \cup \{\neg\alpha\}$ não é S3-satisfazível.*

Exemplo 3.3: Considere os dados do exemplo 3.2. Vamos tentar construir um conjunto *S* até que se possa decidir se $\Gamma \not\models^1_S \alpha$ ou $\Gamma \models^3_S \alpha$ (e, conseqüentemente, se $\Gamma \models \alpha$ ou $\Gamma \not\models \alpha$).

Para poder aplicar o teorema 3.4, inicializamos o conjunto S com os átomos de α. Em seguida, tentamos aplicar os métodos de simplificação definidos nos lemas 3.1 e 3.3. Desta forma, teremos:

1. $S := \{child, pensioner\}$.

   Usando o teorema 3.2, faça

   $\Gamma' :=$ simplifica-1($\Gamma \cup \{\neg\alpha_S\}, S$) = simplifica-1($\Gamma \cup \{child, \neg pensioner\}, S$)

   $\Rightarrow \Gamma' = \{child,$ **falso**$, pensioner, \neg pensioner\}$

   Note que devido à remoção de literais pelo método simplifica-1, temos 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, temos que $\Gamma \models^1_S \alpha$.

   Usando o teorema 3.4, faça

   $\Gamma' :=$ simplifica-3($\Gamma \cup \{\neg\alpha\}, S$) = simplifica-1($\Gamma \cup \{child, \neg pensioner\}, S$)

   $\Rightarrow \Gamma' = \{child, \neg pensioner\}$.

   Temos que Γ' é satisfazível na lógica clássica. Portanto, a partir do lema 3.3, temos que $\Gamma \cup \{\neg\alpha\}$ é ***S3***-satisfazível e pelo teorema 3.4 concluímos que $\Gamma \not\models^3_S \alpha$.

2. $S := S \cup \{worker\}$

   $\Gamma' :=$ simplifica-1($\Gamma \cup \{\neg\alpha_S\}, S$) = simplifica-1($\Gamma \cup \{child, \neg pensioner\}, S$)

   $\Rightarrow \Gamma' = \{child, worker, pensioner \vee worker, \neg pensioner, \neg pensioner \vee \neg worker, \neg pensioner\}$

   Temos que Γ' é satisfazível na lógica clássica. Portanto, temos que $\Gamma \cup \{\neg\alpha_S\}$ é ***S1***-satisfazível. Pelo teorema 3.2, concluímos que $\Gamma \not\models^1_S \alpha$ e, assim, $\Gamma \not\models \alpha$.

Uma boa característica deste sistema proposto por Cadoli e Schaerf é estabelecer um algoritmo incremental de aproximação para a lógica clássica. Porém, o método apresenta duas grandes desvantagens:

1. ele é restrito a fórmulas na forma clausal, e não permite o uso de implicação ($\rightarrow$), pois sua tradução para forma clausal não preserva os valores-verdade sob a semântica da lógica ***S3***, conforme visto em [FW01];
2. não é descrito nenhum método para escolha da próxima variável a ser incluída no conjunto *S*.

A fim de analisar melhor estas limitações, será feita a implementação da lógica ***S1***, a ser descrita na próxima seção.

5 – Implementando a lógica S1

Nesta seção iremos descrever uma primeira implementação da lógica ***S1***, associada ao algoritmo GSAT. Para isso, utilizamos o problema das N-rainhas como entrada para o programa. No problema das N-rainhas, deve-se encontrar um posicionamento para N rainhas em um tabuleiro N x N de forma que nenhuma rainha ataque outra.

A primeira tarefa, então, foi traduzir o problema das N-rainhas para o formato de um problema de satisfatibilidade. Cada casa do tabuleiro foi associada a uma variável proposicional. Caso uma rainha estivesse posicionada nesta casa, a variável assumiria valor verdadeiro. Para codificar o problema, usamos inicialmente N fórmulas para garantir a existência de pelo menos uma rainha para cada linha do tabuleiro. Depois, uma grande quantidade de fórmulas para garantir que não existissem duas rainhas em uma mesma linha, coluna ou diagonal.

A seguir, a implementação da lógica ***S1*** foi feita conforme o algoritmo abaixo:

```
procedure S1
Entrada: conjunto de fórmulas na forma clausal Γ, inteiros MAX-FLIPS e
         MAX-TRIES.
Saída:   0, se não existe uma valoração para as variáveis que satisfaça o
         conjunto de fórmulas, 1 caso contrário.
begin
   S := ∅;
   P := conjunto de todas as variáveis proposicionais em Γ;
   while S ≠ P
      p := variável escolhida aleatoriamente de P – S;
      S := S ∪ { p };
      if gsat(simplifica-1(Γ, S), MAX-FLIPS, MAX-TRIES) then return 1;
   end while;
   return 0;
end;
```

Foi feita uma pequena adaptação no algoritmo do GSAT para que ele somente retornasse **verdadeiro** ou **falso.** Além disso, foi utilizado o algoritmo **simplifica-1**, conforme descrito no lema 3.1.

Seja Γ o conjunto de fórmulas na forma clausal. A idéia do algoritmo consiste em provar que $\Gamma \not\models^1_S$ **falso**. Isto só acontece se existir uma valoração que satisfaça Γ. Daí, conforme vimos na seção anterior, teremos que $\Gamma \not\models$ **falso** e, portanto também existirá na lógica clássica uma valoração que satisfaz Γ.

Após a implementação, notamos algumas limitações da lógica ***S1***.

1. O raciocínio dentro de ***S1*** é não local. Isto significa que pelo menos uma variável de cada fórmula em Γ deve estar contida em S para que ela possa ter um valor verdadeiro, decorrente da definição de ***S1***. Desta forma, mesmo cláusulas que são

completamente irrelevantes devem ser examinadas para que se possa obter uma solução. Em particular para o nosso problema das N-rainhas, na maioria das vezes todas as variáveis precisavam ser incluídas em *S*. Desta forma, esta implementação da lógica ***S1*** não imprimiu um ganho considerável sobre o algoritmo GSAT já existente.

2. Como não existe um método para a escolha dos elementos a serem adicionados ao conjunto *S*, a adição de novos elementos em *S* teve de ser feita de maneira aleatória;
3. Este método está limitado a fórmulas na forma clausal, conforme indicado anteriormente.

Apesar destas limitações, o próximo passo seria implementar a lógica ***S1*** utilizando métodos de tableaux, de maneira semelhante à que foi feita com a lógica ***S3*** em [FW01]. Através deste método seria possível superar as duas últimas limitações, pois ele é capaz de indicar um próximo elemento a ser incluído em *S*. Porém, ainda continuaremos com a limitação de ***S1*** ser não local. Em [FW02], é definida uma nova lógica, chamada de ***s1***, que supera esta limitação. Ela será vista na próxima seção.

6 – A lógica s1

Antes de mostrar a lógica ***s1***, vamos definir o que são aproximações da lógica clássica *por baixo* e *por cima*.

Considere a lógica ***S3***(S) como a relação de implicação $\models^3_S$, conforme a definição de Cadoli e Schaerf, parametrizada sobre um conjunto S escolhido. Sendo P o conjunto enumerável de todas as variáveis proposicionais. Temos a seguinte propriedade:
Para

$$\varnothing \subseteq S' \subseteq S'' \subseteq \ldots \subseteq S'^{\,n} \subseteq P$$

temos que

$$\models^3_\varnothing \subseteq \models^3_{S'} \subseteq \models^3_{S''} \subseteq \ldots \subseteq \models^3_{S'^{\,n}} \subseteq \models^3_P = \models_{CL}$$

onde $\models_{CL}$ é a implicação clássica. Por este motivo a lógica ***S3***(S) pode ser chamada de uma *aproximação da lógica clássica por baixo*.

De maneira análoga, dizemos que uma família de lógicas parametrizadas ***L***(S) é uma *aproximação da lógica clássica por cima* se tivermos

$$\models^L_\varnothing \supseteq \models^L_{S'} \supseteq \models^L_{S''} \supseteq \ldots \supseteq \models^L_{S'^{\,n}} \supseteq \models^L_P = \models_{CL}$$

Infelizmente, a lógica ***S1*** não aproxima a lógica clássica por cima. Se isso ocorresse, deveríamos ter que qualquer teorema clássico também seria um teorema de ***S1***(S), para qualquer escolha de S. Porém, temos que fórmula $p \vee \neg p$ é falsa em ***S1*** caso $p \notin S$. Portanto, a lógica ***S1*** não pode ser considerada uma aproximação da lógica clássica por cima.

Finger e Wassermann propuseram então uma nova família de lógicas que aproximam a lógica clássica por cima [FW02], que ainda supera as limitações encontradas na lógica ***S1***. Ao invés de partir de uma valoração binária $\upsilon : \zeta \rightarrow \{0,1\}$ estendendo a lógica clássica, será definida uma nova valoração $\upsilon^1_s : \zeta \rightarrow 2^{\{0,1\}}$ para esta nova lógica. Mais tarde mostraremos que esta valoração é ternária, pois $\upsilon^1_s(\alpha) \neq \varnothing$.

Considere a linguagem completa da lógica clássica baseada no conjunto P de variáveis proposicionais. Definimos a família de lógicas ***s1***(s), cujo parâmetro é um conjunto $s \subseteq P$. Abusando da notação, dizemos que uma fórmula $\alpha \in s$ se todas suas variáveis proposicionais estão contidas em s.

Seja υ_p a valoração clássica. A definição de υ^1_s será feita a partir da valoração clássica, da seguinte maneira:

Seja p uma variável proposicional. Temos:

$0 \in \upsilon^1_s(p) \Leftrightarrow \upsilon_p(p) = 0$

$1 \in \upsilon^1_s(p) \Leftrightarrow \upsilon_p(p) = 1$ ou $p \notin s$

Desta forma, υ_s^1 estende υ_p, mas sempre que temos um átomo $p \notin s$, $1 \in \upsilon_s^1(p)$. Se $p \notin s$ e $\upsilon_p(p) = 0$, então temos que $\upsilon_s^1(p) = \{0,1\}$. O restante da definição de υ_s^1 segue o mesmo raciocínio, conforme indicado abaixo:

$0 \in \upsilon_s^1(\neg\alpha) \Leftrightarrow 1 \in \upsilon_s^1(\alpha)$

$0 \in \upsilon_s^1(\alpha \wedge \beta) \Leftrightarrow 0 \in \upsilon_s^1(\alpha)$ ou $0 \in \upsilon_s^1(\beta)$

$0 \in \upsilon_s^1(\alpha \vee \beta) \Leftrightarrow 0 \in \upsilon_s^1(\alpha)$ e $0 \in \upsilon_s^1(\beta)$

$0 \in \upsilon_s^1(\alpha \rightarrow \beta) \Leftrightarrow 1 \in \upsilon_s^1(\alpha)$ e $0 \in \upsilon_s^1(\beta)$

$1 \in \upsilon_s^1(\neg\alpha) \Leftrightarrow 0 \in \upsilon_s^1(\alpha)$ ou $\neg\alpha \notin s$

$1 \in \upsilon_s^1(\alpha \wedge \beta) \Leftrightarrow 1 \in \upsilon_s^1(\alpha)$ e $1 \in \upsilon_s^1(\beta)$ ou $\alpha \wedge \beta \notin s$

$1 \in \upsilon_s^1(\alpha \vee \beta) \Leftrightarrow 1 \in \upsilon_s^1(\alpha)$ ou $1 \in \upsilon_s^1(\beta)$ ou $\alpha \vee \beta \notin s$

$1 \in \upsilon_s^1(\alpha \rightarrow \beta) \Leftrightarrow 0 \in \upsilon_s^1(\alpha)$ ou $1 \in \upsilon_s^1(\beta)$ ou $\alpha \rightarrow \beta \notin s$

A seguir vamos enunciar duas propriedades básicas de υ_s^1.

Lema 6.1 ([FW02]) Seja α uma fórmula qualquer. Temos que $\upsilon_s^1(\alpha) \neq \emptyset$.

Prova: Note que para cada variável proposicional temos que $\upsilon_p(p) \in \upsilon_s^1(p)$ e, portanto, $\upsilon_s^1(p) \neq \emptyset$. Fazendo uma simples indução estrutural sobre a fórmula α mostra que $\upsilon_s^1(\alpha) \neq \emptyset$.

Lema 6.2 ([FW02]) Seja α uma fórmula qualquer. Se $\alpha \notin s$, então $1 \in \upsilon_s^1(\alpha)$.

Prova: Pode-se deduzir diretamente a partir da definição de υ_s^1.

7 – Considerações finais e trabalhos futuros

O estudo sobre a lógica **s1** ainda não foi concluído. Portanto, as próximas atividades deste projeto consistirão em um aprofundamento no estudo desta lógica e sua implementação. Inicialmente, tentaremos adaptar o sistema já desenvolvido para a lógica ***S1*** que utiliza o algoritmo GSAT.

A seguir, será feita a implementação da lógica ***s1*** utilizando métodos de tableaux, seguindo a mesma linha de [FW01]. Uma análise comparativa entre os diferentes métodos será feita, utilizando-se de benchmarks contidos no SATLIB [SL].

8 – Referências

[CS95] Marco Cadoli e Marco Schaerf. Tractable Reasoning via Approximation. *Artificial Intelligence*, 74(20):249-310, 1995

[FW01] Marcelo Finger e Renata Wassermann. Tableaux for Approximate Reasoning.*Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI2001)*, Seattle, Washington, USA, 2001

[FW02] Marcelo Finger e Renata Wassermann. Logics for Approximate Reasoning: Approximating Classical Logic "From Above", 2002

[SL] Holger H. Hoos e Thomas Stüzle. SATLIB - The Satisfiability Library *http://www.satlib.org*

[SLM92] Bart Selman, Hector Levesque, David Mitchell. A New Method for Solving Hard Satisfiability Problems. *Proceedings of the Tenth National Conference on Artificial Intelligence*, 440-446, 1992