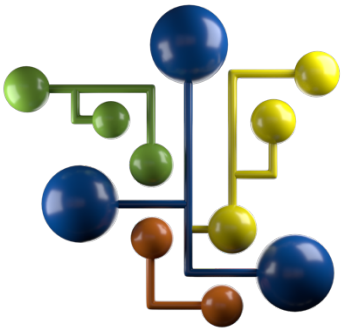

Data Science Academy

www.datascienceacademy.com.br

Introdução à Inteligência Artificial

Modelos Proposicionais Eficientes

Vamos descrever duas famílias de algoritmos eficientes para inferência proposicional geral, baseados na verificação de modelos: uma abordagem baseada na busca com retrocesso e outra na busca de subida de encosta local. Esses algoritmos fazem parte da "tecnologia" da lógica proposicional.

O primeiro algoritmo que examinaremos é frequentemente chamado algoritmo de Davis-Putnam, devido ao importante artigo de Martin Davis e Hilary Putnam (1960). De fato, o algoritmo é a versão descrita por Davis, Logemann e Loveland (1962), e, por essa razão, vamos chamá-lo DPLL, um acrônimo formado pelas iniciais dos quatro autores. O DPLL recebe como entrada uma sentença em forma normal conjuntiva — um conjunto de cláusulas. Como BUSCA-COM-RETROCESSO CONSEQUÊNCIA-LÓGICA-TV?, ele é em essência uma enumeração recursiva em profundidade de modelos possíveis. Ele incorpora três melhorias em relação ao esquema simples de CONSEQUÊNCIA-LÓGICA-TV?:

1. **Término prematuro**: O algoritmo detecta se a sentença tem de ser verdadeira ou falsa, mesmo no caso de um modelo parcialmente concluído. Uma cláusula é verdadeira se qualquer literal é verdadeiro, mesmo que os outros literais ainda não tenham valores-verdade; consequentemente, a sentença como um todo pode ser considerada verdadeira, mesmo antes de o modelo estar completo. Por exemplo, a sentença $(A \vee B) \wedge (A \vee C)$ é verdadeira se A é verdadeiro, independentemente dos valores de B e C. De modo semelhante, uma sentença é falsa se qualquer cláusula é falsa, o que ocorre quando cada um de seu literais é falso. Mais uma vez, isso pode ocorrer bem antes de o modelo estar completo. O término prematuro evita o exame de subárvores inteiras no espaço de busca.

2. **Heurística de símbolo puro**: Um símbolo puro é um símbolo que sempre aparece com o mesmo "sinal" em todas as cláusulas. Por exemplo, nas três cláusulas $(A \vee \neg B)$, $(\neg B \vee \neg C)$ e $(C \vee A)$, o símbolo A é puro porque só aparece o literal positivo, B é puro porque só aparece o literal negativo e C é impuro. É fácil ver que, se uma sentença tem um modelo, ela tem um modelo com os símbolos puros atribuídos de forma a tornar seus literais verdadeiros, porque isso nunca poderá tornar uma cláusula falsa. Observe que, na determinação da pureza de um símbolo, o algoritmo pode ignorar cláusulas que já são reconhecidas como verdadeiras no modelo construído até o momento. Por exemplo, se o modelo contém B = falso, a cláusula $(\neg B \vee \neg C)$ já é verdadeira, e nas cláusulas C restantes aparece apenas um literal positivo; por essa razão, C torna-se puro.

3. **Heurística de cláusula unitária**: Uma cláusula unitária foi definida anteriormente como uma cláusula com apenas um literal. No contexto de DPLL, essa expressão também significa cláusulas em que todos os literais com exceção de um já têm o valor falso atribuído pelo modelo. Por exemplo, se o modelo contém B = verdadeiro, então $(\neg B \vee \neg C)$ simplifica para $\neg C$, que é uma cláusula unitária. É óbvio que, para que essa cláusula seja verdadeira, C deve ser definido como falso. A heurística de cláusula unitária atribui todos esses símbolos antes de efetuar a ramificação sobre o restante. Uma consequência importante da heurística é que qualquer tentativa de provar (por refutação) um literal que já está na base de

conhecimento terá sucesso imediato. Note também que a atribuição de uma cláusula unitária pode criar outra cláusula unitária — por exemplo, quando C é definido como falso, (C ∨ A) se torna uma cláusula unitária, fazendo com que o valor verdadeiro seja atribuído a A. Essa "cascata" de atribuições forçadas é chamada propagação unitária. Ela lembra o processo de encadeamento para a frente com cláusulas definidas.

```
função SATISFATÍVEL-DPLL?(s) retorna verdadeiro ou falso
  entradas: s, uma sentença em lógica proposicional

  cláusulas ← o conjunto de cláusulas na representação em FNC de s
  símbolos ← uma lista dos símbolos proposicionais em s
  retornar DPLL(cláusulas, símbolos, { })

função DPLL(cláusulas, símbolos, modelo) retorna verdadeiro ou falso

  se toda cláusula em cláusulas é verdadeira em modelo então retornar verdadeiro
  se alguma cláusula em cláusulas é falsa em modelo então retornar falso
  P, valor ← ENCONTRAR-SÍMBOLO-PURO(símbolos, cláusulas, modelo)
  se P é não nulo então retornar DPLL(cláusulas, símbolos – P, modelo ∪ {P = valor})
  P, valor ← ENCONTRAR-CLÁUSULA-UNITÁRIA(cláusulas, modelo)
  se P é não nulo então retornar DPLL(cláusulas, símbolos – P, modelo ∪ {P = valor})
  P ← PRIMEIRO(símbolos); restantes ← RESTO(símbolos)
  retornar DPLL(cláusulas, restantes, modelo ∪ {P = verdadeiro}) ou
                DPLL(cláusulas, restantes, modelo ∪ {P = falso})
```

Algoritmo DPLL

No capítulo anterior, vimos vários algoritmos de busca local, incluindo SUBIDA-DE-ENCOSTA e TÊMPERA-SIMULADA. Esses algoritmos podem ser aplicados diretamente a problemas de satisfatibilidade, desde que seja escolhida a função de avaliação correta. Como o objetivo é encontrar uma atribuição que satisfaça a toda cláusula, uma função de avaliação que efetue a contagem do número de cláusulas não satisfeitas fará o trabalho. De fato, essa é exatamente a medida usada pelo algoritmo CONFLITOS-MÍNIMOS. Todos esses algoritmos executam etapas no espaço de atribuições completas, invertendo o valor-verdade de um símbolo de cada vez. Normalmente, o espaço contém muitos mínimos locais e são exigidas várias formas de aleatoriedade para escapar desses mínimos locais. Nos últimos anos, houve um grande volume de experimentos com a finalidade de descobrir um bom equilíbrio entre o caráter ambicioso e a aleatoriedade.

Um dos mais simples e mais eficientes algoritmos a emergir de todo esse trabalho é chamado WALKSAT (abaixo). Em toda iteração, o algoritmo seleciona uma cláusula não satisfeita e um símbolo na cláusula a ser invertido. Ele escolhe ao acaso entre dois modos de

selecionar o símbolo a ser invertido: (1) uma etapa de “conflitos mínimos”, que minimiza o número de cláusulas não satisfeitas no novo estado, e (2) uma etapa de “percurso aleatório”, que seleciona o símbolo aleatoriamente.

```
função WALKSAT(cláusulas, p, inversões_max) retorna um modelo satisfatório ou falha
  entradas: cláusulas, um conjunto de cláusulas em lógica proposicional
            p, a probabilidade de optar por realizar um movimento de “percurso aleatório”,
             normalmente em torno de 0,5
            inversões_max, número de inversões permitidas antes de desistir

  modelo ← uma atribuição aleatória de verdadeiro/falso aos símbolos de cláusulas
  para i = 1 até inversões_max faça
    se modelo satisfaz cláusulas então retornar modelo
    cláusula ← uma cláusula selecionada ao acaso de cláusulas que é falsa em modelo
    com probabilidade p, inverter o valor de modelo de um símbolo selecionado ao acaso de
cláusula
    senão inverter qualquer símbolo em cláusula que maximize o número de cláusulas satisfeitas
  retornar falha
```

Algoritmo WALKSAT

Por essa razão, o WALKSAT é mais útil quando esperamos que exista uma solução. Por outro lado, o WALKSAT nem sempre pode detectar a não satisfatibilidade, necessária para definir a consequência lógica. Por exemplo, um agente não pode utilizar o WALKSAT para provar que um quadrado é seguro em um determinado problema. Em vez disso, ele pode dizer: “Pensei durante uma hora e não consegui descobrir um mundo possível em que o quadrado não fosse seguro.” Esse pode ser um bom indicador empírico de que o quadrado é seguro, mas certamente não é uma prova.

Referências:

Livro: Inteligência Artificial
Autor: Peter Norvig