UNIVERSIDADE
FEDERAL DO CEARÁ


Discentes: Iago Oliveira Lima
Robert de Almeida Cabral
Docente: Profa. Dra. Maria Viviane de Menezes
Curso: Engenharia de Computação
6º Semestre


# Inteligência Artificial

## Laboratório 03 - Resolvedores SAT

7 de dezembro de 2017

# 1. Introdução

O problema das N rainhas é um problema clássico da computação e muito utilizado para o ensino de conteúdos da área de Inteligência Artificial. O problema consiste em colocar N rainhas em um tabuleiro N x N, de modo que nenhuma rainha fique na mesma linha, coluna ou diagonal que outra rainha, ou seja não possam se atacar. O número de rainhas deve ser sempre superior ou igual a 4, pois garantimos assim que exista pelo menos uma solução para o problema. Nas soluções sempre irá existir uma rainha em cada linha do tabuleiro.

Exitem vários métodos para resolver este problema, neste trabalho, apresentaremos a modelagem em lógica proposicional e os resultados da execução do problema no programa MiniSat, que é um dos inúmeros programas que solucionam problemas SAT.

# 2. Experimentos

## 2.1 Descrição dos experimentos

A tabela 1 descreve as reais condições de hardware e software onde os experimentos foram executados.

| | |
|---|---|
| Processador | Intel Core i7-3537U 2.00GHz |
| Núcleos | 4 |
| Memória RAM | 8GB |
| Disco Rigido | 1TB |
| Sistema Operacional | Ubuntu Gnome 17.04 |
| IDE/Versão | Qt Creator/5.9.1 |

Tabela 1: Plataformas de hardware e software

Os experimentos foram realizados executando 15 problemas, onde o número N de rainhas que era dado como entrada para o software desenvolvido era variante ($10 \leq N \leq 200$) como descrito na tabela 2, a primeira coluna nos mostra a entrada N que foi concedida ao programa, que indica o número de rainhas (N) e o tamanho do tabuleiro (N x N), já a segunda coluna nos mostra o tempo em segundos que o MiniSat levou para encontrar uma solução. A saída deste software é um arquivo contendo a modelagem no formato cnf (instance_NxN.cnf) que servirá como entrada para o MiniSat.

| Número de rainhas | Tempo (s) |
|---|---|
| 10 | 0,012s |
| 20 | 0,036s |
| 30 | 0,064s |
| 40 | 0,180s |
| 50 | 0,384s |
| 60 | 0,812s |
| 70 | 1,400s |
| 80 | 2,584s |
| 90 | 3,904s |
| 100 | 5,748s |
| 120 | 12,896s |
| 140 | 25,164s |
| 160 | 46,000s |
| 180 | 74,952s |
| 200 | 119,224s |

Tabela 2: Entrada vs Tempo de execução

## 3. Estratégia

Como dito na seção 1, o exercício proposto foi modelar o problema das N rainhas de acordo com as convenções do formato cnf, para que o programa usado solucione a fórmula descrita no arquivo gerado pelo software. É de conhecimento geral que o problema tem solução para $N \geq 4$ e que uma rainha pode atacar outra que estejam na mesma linha, coluna ou diagonais, tanto principal como secundária. Então, com isso, chegamos as seguintes regras:

1. Se $v(p_{ij}) = 1$, então não existe nenhuma outra rainha na linha i;
2. Se $v(p_{ij}) = 1$, então não existe nenhuma outra rainha na coluna j;
3. Se $v(p_{ij}) = 1$, então não existe nenhuma outra rainha nas suas diagonais principais e secundárias;
4. Existe uma rainha em cada coluna;
5. Existe uma rainha em cada linha.

O formato cnf ou *clausal normal form*, como diz o nome, é um formato em que uma cláusula é uma disjunção de literais, ou seja, todas as fórmulas devem ser escritas somente com negações, conjunções e disjunções. Cada fórmula deve ser escrita somente com disjunções e o conjunto de todas as fórmulas como conjunções. Para escrever um arquivo .cnf existem algumas regras, como descrito no exemplo abaixo:

```
p cnf 2 2
-1 2 0
-2 1 0
```

O arquivo .cnf deve ser iniciado com "p cnf", seguido da quantidade de átomos proposicionais e da quantidade de fórmulas. Desta forma, o exemplo acima mostra a conjunção das fórmulas $(\neg p \vee q) \wedge (\neg q \vee p)$, que seguindo a lógica proposicional pode ser escrita como $(p \Rightarrow q) \wedge (q \Rightarrow p)$. O valor 1 corresponde ao p e 2 ao q. O 0 no final da linha indica que a fórmula acaba ali.

Seguindo a regra 1 e modelando em lógica proposicional, temos as seguintes fórmulas:

$$
\begin{gathered}
(\neg p_{1,1} \vee \neg p_{1,2}) \wedge (\neg p_{1,1} \vee \neg p_{1,3}) \wedge \cdots \wedge (\neg p_{1,1} \vee \neg p_{1,N}) \\
\vdots \\
(\neg p_{N,1} \vee \neg p_{N,2}) \wedge (\neg p_{N,1} \vee \neg p_{N,3}) \wedge \cdots \wedge (\neg p_{N,1} \vee \neg p_{N,N-1})
\end{gathered}
$$

Então, como cada posição da matriz é tratada como um átomo proposicional, todas as outras regras seguem praticamente o mesmo padrão, porém com i e j variando para percorrer as colunas e as diagonais. Com isso, desenvolvemos um programa na linguagem de programação C++ que descreve essas regras no arquivo de saída. A classe CNF, que tem como parâmetro um N, que é a quantidade de rainhas a serem distribuídas no tabuleiro N x N. A classe também tem mais três métodos, o método getPos, generate e getLocateFile.

- getPos:
  Dado um i e j, retorna a posição exata na matriz
- generate:
  Gera o arquivo de saída com a fórmula em cnf e retorna *true* caso não ocorra nenhum erro e e *false* caso contrário.
- getLocateFile:
  Retorna a localização do arquivo com a fórmula.

Para a manipulação de arquivos, foi usado a classe *ofstream*. O trecho de código abaixo mostra como o programa trata a diagonal principal de uma rainha:

```
for (int i=k, j=z; j<N && i<N; i++, j++){
    if(getPos(i,j) != getPos(k,z)){
    outFile << getPos(k,z)*-1 << " " << getPos(i,j)*-1 << " 0" << endl;
    this->qtd++;
    }
}
```

Desta forma, o programa vai fazer para todos os valores de N suas diagonais principais e escrever no arquivo a posição (i,j) da matriz multiplicada por -1 e a posição (i+1,j+1) na matriz multiplicada por -1, pois $(p_{i,j} \Rightarrow \neg p_{i+1,j+1})$, e portanto $(\neg p_{i,j} \vee \neg p_{i+1,j+1})$, onde $v(P_{i,j}) = 1$. As demais diagonais são semelhantes. O zero indica o final da linha e a variavel "qtd"conta a quantidade de fórmulas escritas no arquivo, para no fim indicar no inicio do arquivo a quantidade de fórmulas.

E por fim as regras 4 e 5 podem ser descritas pelas seguintes fórmulas:

$$\begin{gathered}(p_{1,1} \vee p_{1,2} \vee p_{1,3} \vee \cdots \vee p_{1,N}) \\ \vdots \\ (p_{N,1} \vee p_{N,2} \vee p_{N,3} \vee \cdots \vee p_{N,N})\end{gathered}$$

Com as fórmulas descritas, o seguinte trecho de código consegue descrever no arquivo as regras.

```
for(int i = 1;i <= N*N;i = i+N){
    for(int j = i;j < i+N;j++)
        outFile << j << " ";

    this->qtd++;
    outFile << "0" << endl;
}
```

# 4. Análise dos resultados e Conclusão

O gráfico abaixo foi plotado utilizando os dados mostrados na segunda coluna da tabela 2, com isso poderemos analisar melhor o comportamento do tempo de execução dos problemas no MiniSat.

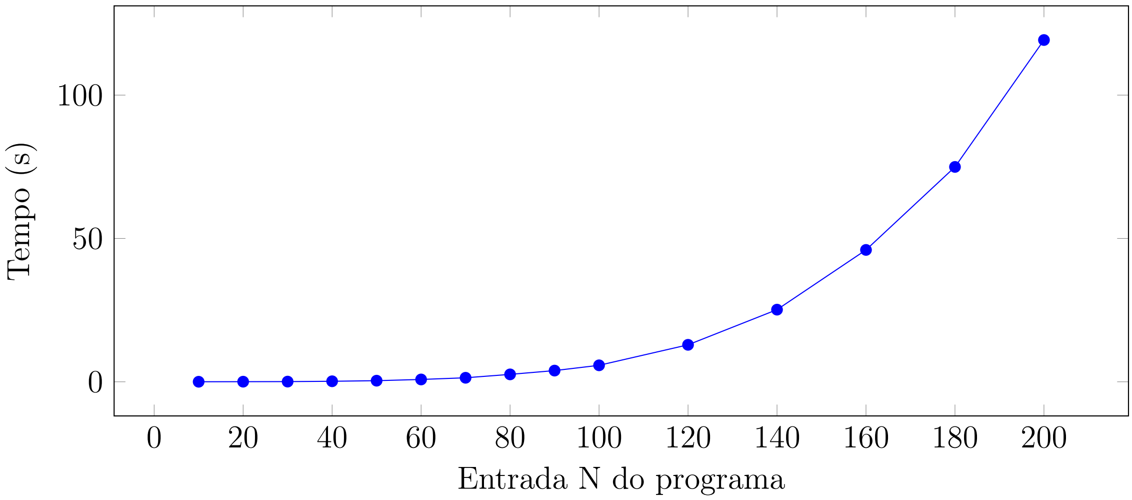


Figura 1: Tempo de execução dos experimentos no MiniSat

Observando o gráfico é perceptível que o tempo para que o MiniSat solucionasse o problema, assemelha-se com o gráfico de um função exponencial ($e^x$). Vemos assim que o tempo para a solução do problema aumenta consideravelmente com entradas (N) variando em pequenas quantidades. O problema pode se tornar inviável para ocorrências onde a entrada N seja muito grande.

Notamos que com o N = 200, obtivemos um valor de tempo extremamente próximo ao valor limite estipulado, que era de 120 segundos. Obtivemos nesse teste o valor de tempo igual à 119.224 segundos.

Com entradas cada vez maiores e com o problema gerando cláusulas de forma exponencial, a resolução de tais problemas requerem cada vez mais recursos, podendo atingir um ponto onde os recursos não permitam que os problemas sejam resolvidos.

O fato de existirem casos onde o tempo de resolução é exorbitantemente grande, e dada as limitações de recursos computacionais, o problem das N rainhas enquadra-se como um provlema NP-Completo.

A definição dada por Paulo Feofiloff de problemas NP-Completos é: "Os problemas NP-completos estão em NP, o conjunto de todos os problemas cujas **soluções podem ser verificadas em tempo polinomial**. Um problema é NP-difícil se for tão difícil quanto qualquer problema em NP. **Um problema é NP-completo se for NP-difícil e estiver em NP**."

Ou seja, um problema é considerado NP-Completo se não for conhecido um algoritmo que encontre soluções em tempo polinomial, como é o caso do problema das N rainhas.

> A maior deficiência da raça humana é a nossa incapacidade de entender a função exponencial.
>
> Albert A. Bartlett