Resolvedores SAT
Inteligência Artificial

José Robertty de Freitas Costa[1], **Marianna de Pinho Severo**[1], **Marisa do Carmo Silva**[1]

[1]Campus Quixadá – Universidade Federal do Ceará (UFC)
Docente: Maria Viviane de Menezes

{robertty, mariannapinho, marisa296}@alu.ufc.br

1. Introdução

O problema das N rainhas é um famoso problema da computação onde, em um tabuleiro de xadrez *N x N*, tenta-se colocar *N* rainhas, de tal modo que uma rainha não fique na mesma linha, coluna ou diagonal onde outra rainha está posicionada. Este problema pode ser resolvido por vários métodos, e, neste trabalho, mostraremos como ele foi modelado em lógica proposicional, e os resultados de sua execução com o programa Minisat - um dos programas que resolvem problemas SAT.

O problema das *N* rainhas é uma extensão do problema das 8 rainhas e consiste em dispor *N* rainhas ($N \geq 4$, pois sabe-se que, a partir de $N = 4$, temos pelo menos uma solução para o problema) sobre um tabuleiro de dimensões *N x N*, de tal modo que elas não se ataquem, seguindo as regras do xadrez. Nas soluções do problema, há sempre uma rainha em cada linha do tabuleiro.

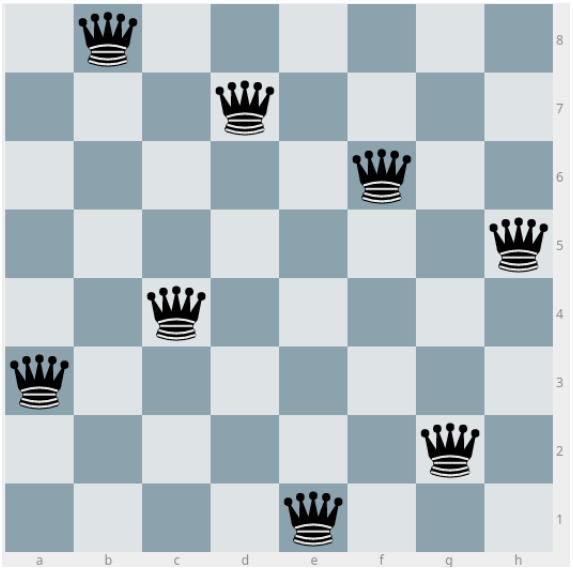

Figura 1. Exemplo de configuração

Para a realização dos testes e execução do Minisat, utilizamos uma máquina com as seguintes especificações:

- Sistema operacional: Ubuntu 16.04 LTS
- Processador: Intel core i5-4210U 1.70GHz
- Memória: 7,7 GiB

- Tipo de sistema: 64-bit

Já, para que se possa criar as instâncias para que o Minisat possa executar, criamos um programa utilizando a linguagem C++ e o ambiente de desenvolvimento Qt Creator.

Este trabalho está dividido em quatro seções, em que a presente seção teve como objetivo apresentar o problema. Na Seção 2, trazemos a estratégia de como resolvemos o problema. Na Seção 3, mostramos o gráfico de tempo de execução dos testes realizados e, por fim, na Seção 4 trazemos a conclusão de nosso trabalho.

2. Estratégia

Como já foi dito, utilizamos um resolvedor SAT para encontrarmos a solução para o problema. Para a modelagem deste, criamos uma matriz em que cada posição é considerada um símbolo atômico da Lógica Proposicional. Geramos fórmulas onde, para uma determinada matriz *N x N*, numeramos cada posição com valores de 1 até *N* * *N* (N vezes N), em que cada número representa um símbolo atômico. Cada fórmula passada deve ser vista como um conjunto de átomos proposicionais reunidos por disjunções, e a reunião entre as fórmulas deve ser vista como feita por conjunções. Dessa maneira, as fórmulas estão na Forma Normal Conjuntiva (CNF).

Exemplo:

p cnf 3 2

1 2 3 0

-1 2 -3 0

O código acima está formulando uma estrutura com 3 símbolos e 2 fórmulas. Estas estão na Forma Normal Conjuntiva, e descrevem, exatamente, a expressão [(p $\vee$ q $\vee$ r) $\wedge$ ($\neg$p $\vee$ q $\vee$ $\neg$r)], em que p, q e r são o primeiro, segundo e terceiro símbolo, respectivamente. O "0" é responsável por indicar o fim de uma fórmula.

Assim, dado um número *N*, podemos criar as fórmulas em CNF, utilizando o programa desenvolvido, que modelem o problema das *N* rainhas. As regras que utilizamos na geração foram as seguintes:

- Existe uma rainha em cada linha;
- Existe uma rainha em cada coluna;
- Se existe uma rainha na posição (i, j), então todas as posições da mesma linha i não podem possuir outra rainha;
- Se existe uma rainha na posição (i, j), então todas as posições da mesma coluna j não podem possuir outra rainha;
- Se existe uma rainha na posição (i, j), então todas as posições que pertencem a uma mesma diagonal (principal ou secundária) não podem possuir outra rainha.

Idealizamos o algoritmo gerador de instâncias da seguinte forma: dada uma posição (i,j), podemos pensar no posicionamento das rainhas como uma implicação: se a rainha está na linha i e na coluna j, então não pode haver outra rainha na mesma linha e coluna. Logo, a partir da adoção de uma determinada posição, percorremos linhas, colunas e diagonais e fomos criando as fórmulas, de acordo com a transformação da implicação para sua forma disjuntiva ($p \rightarrow q \Rightarrow \neg p \vee q$).

A tabela abaixo mostra os testes feitos neste trabalho, onde passamos um valor *N* para o programa desenvolvido e ele gerou as instâncias (fórmulas) que foram utilizadas pelo Minisat para resolver o problema. Na primeira coluna, temos os valores de *N*, que nos informam a dimensão do tabuleiro (*N x N*) e a quantidade de rainhas dispostas neste tabuleiro. A segunda coluna refere-se ao tempo, em segundos, que o Minisat levou para resolver o problema.

Tabela 1. Testes realizados

N	Tempo (s)
5	0.004
10	0.008
20	0.020
40	0.184
50	0.400
60	0.760
80	2.448
100	5.712
110	7.736
120	11.520
150	35.616
170	60.372
180	78.060
190	99.044
199	118.304

3. Gráficos de Tempo

A partir da tabela, construímos o gráfico da Figura 2 para, assim, analisarmos o crescimento do tempo de execução do Minisat de acordo com o aumento do tamanho do tabuleiro.

As linhas representam o valor de *N* e as colunas mostram o tempo, em segundos. Pode-se observar que, a partir de *N* igual a 200, ultrapassamos o tempo de dois minutos que foi estipulado, sendo o problema resolvido em 120.524s.

Observando-se o gráfico, podemos perceber que o tempo para a resolução do problema cresce muito para quantidades cada vez maiores de rainhas, apresentando grandes diferenças no tempo para pequenas diferenças entre os valores de *N*.

Também foram realizados testes para valores de *N* maiores que 200. Quando utilizamos 300, não conseguimos obter resultados, pois o tempo para geração do problema foi muito grande e o computador utilizado travou.

4. Conclusão

Com base no que foi apresentado, podemos notar que o crescimento do tempo de execução do Minisat apresenta um formato exponencial, fazendo com que os casos em que *N* é muito grande demorem muito para finalizarem a execução, tornando-se inviáveis.

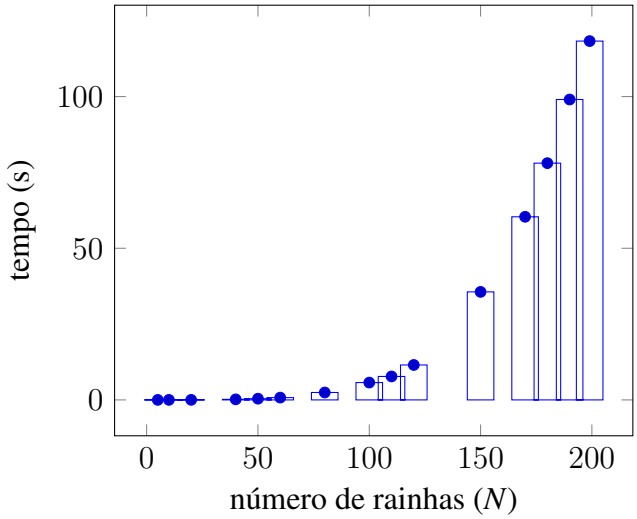

Figura 2. Tempos dos testes realizados

Para valores cada vez maiores de N, o número de cláusulas geradas cresce exponencialmente, de forma que a geração e resolução do problema exigem cada vez mais recursos computacionais, chegando ao instante em que eles não são suficientes e o problema não pode ser resolvido. Apesar de o problema ser de fácil entendimento, tais limitações tornam o problema difícil. O fato de possuirmos valores que tornam o tempo de execução exageradamente grande, leva-nos ao questionamento se o problema tem ou não solução, já que não chegamos a obter uma resposta, mas também não sabemos se ele pode gerar a resposta em algum momento. Desta forma, ele é considerado NP-Completo.