O Problema SAT para N Rainhas

Francisco Felipe da Silva, Iury Queiros Soares, Matheus Fernandes do Nascimento Dantas
Matriculas: 374850, 374852, 378578
felipesilva543@alu.ufc.br, iuryqueiros@gmail.com, matheus_fernandes@alu.ufc.br

1 Introdução

O desenvolvimento desse trabalho tem por objetivo relatar a modelagem do problema das N-Rainhas em CNF e utilização de um resolvedor SAT para encontrar uma solução. Neste relatório apresentamos uma implementação em em linguagem C++ gerador de arquivos CNF com base em uma modelagem do problema por meio de lógica proposicional. Além de expor a experimentação de tais arquivos CNF em um resolvedor SAT.

2 Algoritmos SAT

O problema de satisfatibilidade booleana (SAT) foi o primeiro problema identificado como pertencente à classe de complexidade NP-completo. O problema de satisfatibilidade booleana é o problema de determinar se existe uma determinada valoração que satisfaça uma determinada fórmula em questão. Caso exista uma atribuição de valores de verdade para as variáveis da fórmula que a torne verdadeira, esta fórmula é considera satisfazível, caso não haja nenhuma atribuição ela é considerada insatisfazível.

3 Fórmula CNF

A fórmula CNF (Conjunctive Normal Form) é uma conjunção de cláusulas, constituindo a descrição generalizada de um problema SAT. Cláusulas são disjunção de literais. Um literal é a ocorrência de uma variável na sua forma positiva ou negativa. O CNF é útil em demonstrações automáticas de teoremas. Os únicos conectivos proposicionais que uma fórmula na CNF pode conter são os operadores e (^), ou (v) e negação (¬). O operador de negação pode ser usado apenas como parte de um literal.

Exemplo de um problema de SAT demonstrado na fórmula CNF:

$$\varphi = (p \vee q) \wedge (p \vee \neg r)$$

4 O Problema SAT das NRainhas

O problema das N rainhas é um problema combinatório exponencial que consiste em colocar N rainhas em um tabuleiro de xadrez de forma que não haja colisões, segundo Laguna (1994). Uma colisão ocorre quando duas ou mais rainhas

estão posicionadas na mesma linha, coluna ou diagonal. A solução ótima para este problema será o número de zero colisões, isto é, o problema das N rainhas no xadrez consiste dispor N rainhas num tabuleiro de dimensão NxN, sem que uma rainha ataque as demais.

4.1 Descrição da estratégia:

Para solucionar o problema das N-rainhas primeiramente analisamos quais condições no processo de alocação de rainhas no tabuleiro deveríamos evitar, condições de colisão, para obtermos o cenário ideal onde as N rainhas dispostas no tabuleiro NxN não se ataquem.

Diante disto, as condições de conflito seriam quando duas ou mais rainhas se encontrassem na mesma linha, coluna ou diagonal, isto é, deveria haver exclusivamente uma rainha por coluna, linha e diagonal, fazendo-se necessário uma disjunção exclusiva**.** Previamente definimos átomos proposicionais Pij, onde Pij representa a posição [i, j] na matriz NxN. Obtemos assim a matriz:

	1	2	3	⋯	N
1	P_{11}	P_{12}	P_{13}	⋯	P_{1N}
2	P_{21}	P_{22}	P_{23}	⋯	P_{2N}
3	P_{31}	P_{31}	P_{32}	⋯	P_{3N}
⋮	⋮	⋮	⋮	⋱	⋮
N	P_{N1}	P_{N2}	P_{N2}	⋯	P_{NN}

Imagem 01: Apresentação matriz NxN de átomos proposicionais Pij .

Sendo: Pij verdadeiro quando há uma rainha em [i, j] e falso quando não há. Dado o exposto, foram formuladas as seguintes fórmulas disjuntivas exclusivas para condicionar o cenário ideal.

I. Disjunção exclusiva por linha:

$(P_{i1} \vee P_{i2} \vee P_{i3} \vee ... \vee P_{iN}) \wedge (\neg P_{i1} \vee \neg P_{i2}) \wedge (\neg P_{i1} \vee \neg P_{i3}) \wedge ... \wedge (\neg P_{i1} \vee \neg P_{iN}) \wedge$
$(P_{i1} \vee P_{i2} \vee P_{i3} \vee ... \vee P_{iN}) \wedge (\neg P_{i2} \vee \neg P_{i3}) \wedge (\neg P_{i2} \vee \neg P_{i4}) \wedge ... \wedge (\neg P_{i2} \vee \neg P_{iN}) \wedge$
$(P_{i1} \vee P_{i2} \vee P_{i3} \vee ... \vee P_{iN}) \wedge (\neg P_{i3} \vee \neg P_{i4}) \wedge (\neg P_{i3} \vee \neg P_{i5}) \wedge ... \wedge (\neg P_{i3} \vee \neg P_{iN}) \wedge ...$
$(\neg P_{i(N-1)} \vee \neg P_{iN})$

II. Disjunção exclusiva por coluna:

$(P_{1j} \vee P_{2j} \vee P_{3j} \vee ... \vee P_{Nj}) \wedge (\neg P_{1j} \vee \neg P_{2j}) \wedge (\neg P_{1j} \vee \neg P_{3j}) \wedge ... \wedge (\neg P_{1j} \vee \neg P_{Nj}) \wedge$
$(P_{1j} \vee P_{2j} \vee P_{3j} \vee ... \vee P_{Nj}) \wedge (\neg P_{2j} \vee \neg P_{3j}) \wedge (\neg P_{2j} \vee \neg P_{4j}) \wedge ... \wedge (\neg P_{2j} \vee \neg P_{Nj}) \wedge$
$(P_{1j} \vee P_{2j} \vee P_{3j} \vee ... \vee P_{Nj}) \wedge (\neg P_{3j} \vee \neg P_{4j}) \wedge (\neg P_{3j} \vee \neg P_{5j}) \wedge ... \wedge (\neg P_{3j} \vee \neg P_{Nj}) \wedge ...$
$(\neg P_{(N-1)\,j} \vee \neg P_{Nj})$

III. Disjunção exclusiva por diagonal:

A. Paralelas à diagonal principal:

Caso I: i < j

$(\neg P_{ij} \vee \neg P(i+1)(j+1)) \wedge (\neg P_{ij} \vee \neg P(i+2)(j+2)) \wedge \ldots \wedge (\neg P_{ij} \vee \neg P_{k(N-1)}) \wedge (\neg P_{ij} \vee \neg P_{kN})$

Caso II: i > j

$(\neg P_{ij} \vee \neg P(i+1)(j+1)) \wedge (\neg P_{ij} \vee \neg P(i+2)(j+2)) \wedge \ldots \wedge (\neg P_{ij} \vee \neg P_{(N-1)k}) \wedge (\neg P_{ij} \vee \neg P_{Nk})$

Caso III: i = j

$\neg P_{ij} \vee \neg P(i+1)(j+1)) \wedge (\neg P_{ij} \vee \neg P(i+2)(j+2)) \wedge \ldots \wedge (\neg P_{ij} \vee \neg P_{(N-1)(N-1)}) \wedge (\neg P_{ij} \vee \neg P_{NN})$

Excluso casos onde i = 1, j = N e i = N, j = 1. Sendo $K \in N$.

B. Perpendiculares à diagonal principal:

$\neg P_{ij} \vee \neg P(i+1)(j-1)) \wedge (\neg P_{ij} \vee \neg P(i+2)(j-2)) \wedge \ldots \wedge (\neg P_{ij} \vee \neg P_{xy})$

Sendo $x,y \in N$(Naturais) onde x = N ou y = 1, de modo não exclusivo a P_{N1}.

4.2 Descrição da implementação dos algoritmos:

Após expressarmos, por meio da lógica proposicional, as condições a serem seguidas para obtenção do cenário ideal, construímos alguns exemplos para estudo, um com N = 4 e outro com N = 5. Diante de tais exemplos, das fórmulas elaboradas e da sintaxe CNF, geramos manualmente as saídas CNF desejadas. Analisamos essas saídas de modo a traçar padrões que auxiliassem a criação de algoritmos que gerassem arquivos CNF coerentes a problemática, de modo satisfazível, mediante a entrada N. Sendo N o número de rainhas e NxN a dimensão do tabuleiro. Consitindo-se um processo de engenharia reversa.

Objetivando facilitar a implementação em linguagem C++, numeramos cada posição da matriz NxN com uma número natural em ordem crescente. Por exemplo, uma matriz uma matriz 4x4:

	1	2	3	4
1	**1**	**2**	**3**	**4**
2	**5**	**6**	**7**	**8**
3	**9**	**10**	**11**	**12**
4	**13**	**14**	**15**	**16**

Imagem 2: Matriz 4X4

O algoritmo percorre cada célula da matriz em questão criando as cláusulas em CNF que não permitirão os conflitos entre as rainhas no tabuleiro. Cada cláusula criada é transformada em uma string e armazenada em um vetor, que serve como um contador de cláusulas, e que também é utilizado para criar o arquivo CNF. Temos como

execução do aplicativo a entrada de um parâmetro indicando o número de rainhas e consequência a dimensão do tabuleiro. Assim, o comando ./NRainhas X gera um o arquivo rainhasX.cnf, X sendo o número de rainhas.

5 Experimentos Realizados

Os experimentos consistem em medir o tempo que o programa zChaff, um resolvedor SAT, leva para resolver o problema das n-rainhas para os mais variados valores de N, retornando se o mesmo é ou não satisfazível. Para tal, foi utilizado nosso programa desenvolvido em C++ que gera para qualquer N uma fórmula do tipo CNF.

A plataforma de software utilizada foi a IDE Qt Creator, com base na linguagem de programação C++, no sistema operacional Ubuntu 16.04.3 LTS 64-bit. A plataforma de hardware utilizada foi um notebook casual com configurações: notebook E5-571G-57MJ Intel Core 5 i5 4GB, 2GB Memória Dedicada 1TB LED 15,6 Chumbo - Acer. Foram realizados 10 experimentos com os valores de N variados.

```
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas10.cnf > /dev/null

real    0m0.008s
user    0m0.000s
sys     0m0.000s
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas15.cnf > /dev/null

real    0m0.021s
user    0m0.008s
sys     0m0.000s
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas20.cnf > /dev/null

real    0m0.027s
user    0m0.012s
sys     0m0.000s
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas22.cnf > /dev/null

real    0m0.017s
user    0m0.012s
sys     0m0.008s
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas24.cnf > /dev/null

real    0m0.026s
user    0m0.020s
sys     0m0.004s
```

Imagem 03: Experimentos com N iguais a 10, 15, 20, 22, 24.

```
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas30.cnf > /dev/null

real    0m0.057s
user    0m0.044s
sys     0m0.004s
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas32.cnf > /dev/null

real    0m0.059s
user    0m0.056s
sys     0m0.000s
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas35.cnf > /dev/null

real    0m0.094s
user    0m0.088s
sys     0m0.004s
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas40.cnf > /dev/null

real    0m0.113s
user    0m0.100s
sys     0m0.012s
felipe@BibrownPc:build-NRainhas-Desktop-Debug$ time ./zchaff rainhas45.cnf > /dev/null

real    0m0.160s
user    0m0.140s
sys     0m0.016s
```

Imagem 04: Experimentos com N iguais a 30, 32, 35, 40, 45.

5.1 Gráfico de tempo plotado:

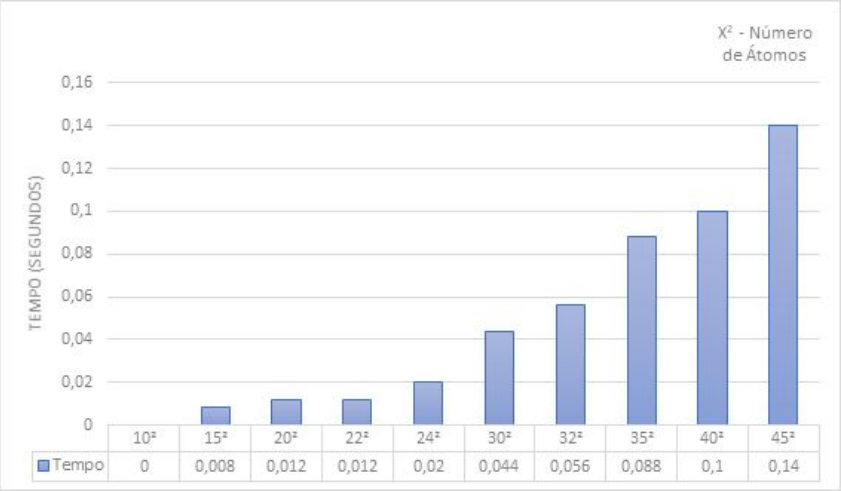

Gráfico 01: Relação Tempo X Num. Átomos

A partir do gráfico pode-se notar que o crescimento do tempo de execução é proporcional ao número de átomos na fórmula.

Conclusão:

O problema das N rainhas é um problema conhecido, de fácil entendimento mas de difícil resolução manualmente à medida que cresce o número de rainhas, e por isso é necessário um programa para resolver esse problema para determinados casos. A implementação da aplicação responsável por gerar o arquivo modelado em CNF para os testes no resolvedor SAT zChaff foi um pouco complicada, mas depois de um tempo de estudo conseguimos concluí-lo. Diante do desafio e do grande esforço de realizar tal trabalho foi interessante observar a propriedade exponencial do problema das N rainhas, além da importância de uma boa formulação para resolução de problemas por meio da implementação em alguma linguagem de programação.