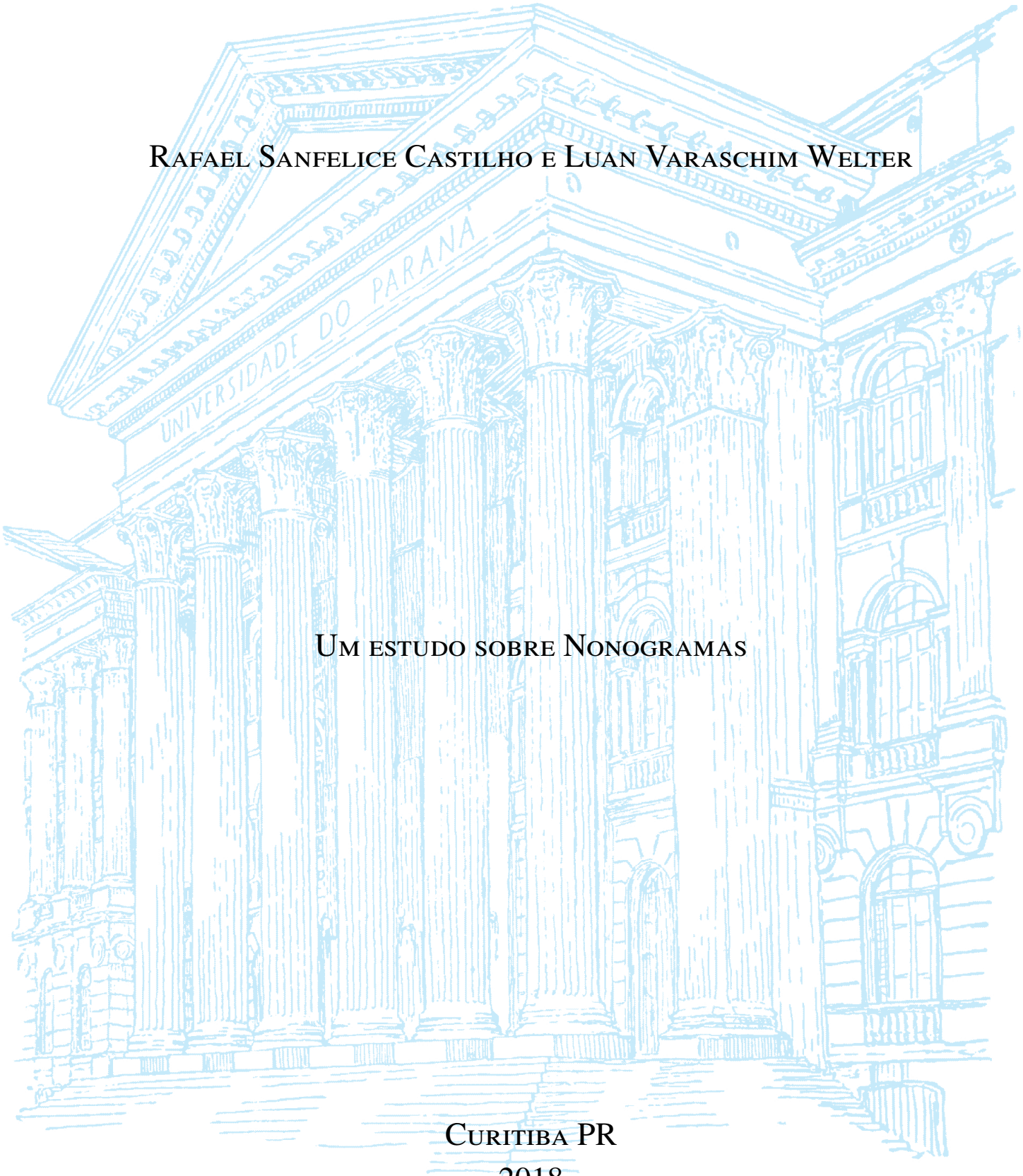
Universidade Federal do Paraná

Rafael Sanfelice Castilho e Luan Varaschim Welter



Um estudo sobre Nonogramas

Curitiba PR
2018

Rafael Sanfelice Castilho e Luan Varaschim Welter

Um estudo sobre Nonogramas

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Inteligência Artificial*.

Orientador: Fabiano Silva.

Curitiba PR
2018

Necessary?

Agradecimentos

Inserir os agradecimentos. Os agradecimentos devem ocupar no máximo uma página, devem ser justificados na largura da página e com um afastamento de parágrafo na primeira linha de 1,27 cm. O espaçamento entre linhas deve ser de 1,5 linhas. Não deve haver espaçamento adicional entre parágrafos.

Resumo

O problema de encontrar uma solução para um nonograma qualquer é um problema NP-Completo. Neste trabalho exploramos as soluções já existentes para o problema, e a partir dela elaboramos uma proposta de solução própria, que foi passada por uma bateria de testes que analisamos.

Palavras-chave: Nonograma, SAT, CSP.

Abstract

The problem of finding a solution to a given nonogram is NP-Complete, in this work we go through the existent solutions for it, and based on them make our own attempt at solving nonograms, we pass our solution through a series of tests wich we analized.

Keywords: Nonogram, SAT, CSP.

Lista de Figuras

Sumário

1 Introdução

Nonogramas são um tipo de quebra-cabeça de lógica japonês criado por Non Ishida em 1988. Nos anos 1990 e 2000, James Dalgety negociou com Non Ishida a distribuição do quebra-cabeça fora do Japão, batizando-o de nonograma em sua homenagem (Non + "grama", de dia**grama**). Hoje esse tipo de jogo já esta presente em diversos jornais e revistas de quebra-cabeças ao redor do mundo.

A representação de um nonograma é simples: uma grade retangular quadriculada com um conjunto de restrições nas laterais. As restrições são dadas para cada linha e para cada coluna na forma de uma sequência de números representando a forma como deve ser preenchida a grade. É comum que em jornais e revistas a grade resolvida forme o desenho de uma uma pessoa, um objeto ou uma letra, como mostra a Figura 1.1.

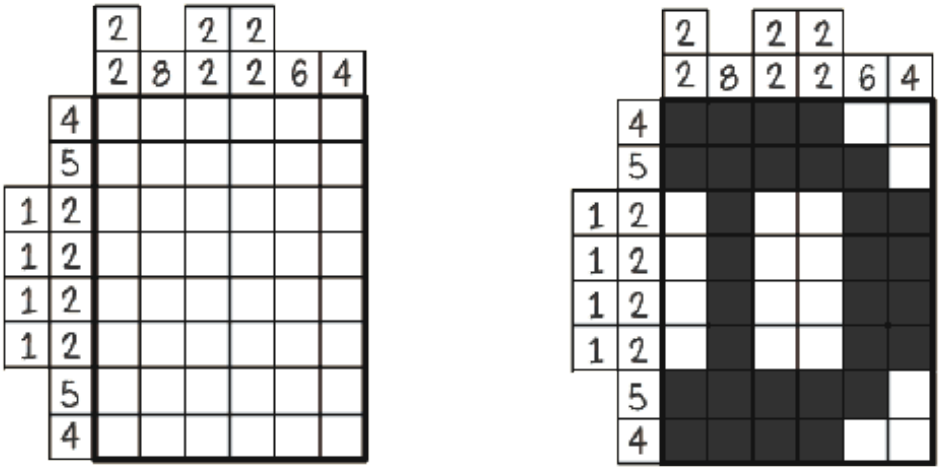


Figura 1.1: Nonograma vazio e sua versão completa

Tradicionalmente, nonogramas são monocromáticos, porém existe uma variação com múltiplas cores. Nesse caso, as restrições também indicam a cor com que preencher a grade do jogo. Há diferentes formas de orientação para o preenchimento; entre elas, colorir as restrições com a cor que deve ser utilizada ou ter um conjunto de restrições adicionais relacionando cores as restrições de preenchimento.

As regras do jogo são simples. Cada número em uma dica aponta uma sequência contínua de casas do tabuleiro que devem ser preenchidas. Por exemplo a dica "**3**" significa que existe na linha ou coluna uma sequência de três casas que devem ser preenchidas, enquanto a dica "**1 2 1**" mostra uma sequência de tamanho um, seguida por uma sequência de tamanho dois, seguida de outra de tamanho um que devem ser preenchidas. Entre cada sequência de casas preenchidas, pode haver um número arbitrário de casas vazias, com a única condição de que restrições adjacentes da mesma cor devem ter no mínimo uma casa não preenchida entre si

(consequentemente, na versão monocromática do jogo, todas as sequências de preenchimento tem, no mínimo, um espaço vazio entre si).

Com as regras explicadas é possível identificar nonogramas inválidos. Essa denominação, como o próprio nome sugere, é atribuída a nonogramas sem solução. Entre os mais fáceis de identificar estão aqueles em que a soma das sequências em uma dica com o número de espaços obrigatórios é maior do que o tamanho da linha ou da coluna a que a dica se refere, assim como os jogos em que há conflitos entre restrições diferentes, como uma linha tendo que estar vazia enquanto um coluna deve estar totalmente preenchida. As figuras 1.2 e 1.3 mostram um nonograma invalido por conflito entre restrições. O conflito ocorre na terceira linha, segunda coluna. A figura 1.2 mostra que caso seja atendida a regra da linha, a regra da coluna é violada, e a figura 1.3 mostra que caso seja atendida a regra da coluna, a regra da linha não poderá ser atendida.

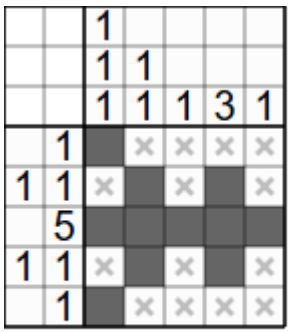


Figura 1.2: Nonograma inválido com conflito causado na segunda coluna

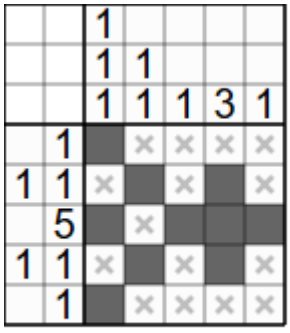


Figura 1.3: Nonograma inválido com conflito na terceira linha

Existem algumas formas de simplificar nonogramas para humanos resolvê-los mais facilmente. Em geral, nonogramas em revistas de quebra-cabeças são, como dito antes, criados a partir de uma imagem já existente, permitindo que os jogadores tenham palpites melhores sobre como preencher baseado na imagem que acham, ou sabem, que será formada. Normalmente os jogos de revistas possuem uma única solução possível, pois ambiguidade de soluções pode dificultar a tomada de decisões. Isto não significa que humanos não consigam resolver quebra-cabeças criados aleatoriamente, ou com múltiplas soluções, apenas que são mais trabalhosos.

O que torna o problema mais difícil de ser resolvido por humanos não é necessariamente uma dificuldade para máquinas. Nenhuma das estratégias de resolução, que serão mostradas

no Capítulo 3, leva em conta uma possível imagem a ser formada; são considerados apenas o tabuleiro e as restrições de preenchimento. Uma das estratégias encontra dificuldades em alguns casos particulares de nonogramas com múltiplas soluções, não conseguindo escolher qual das duas utilizar como solução, porém todas as outras formas estudadas conseguiram achar uma solução sem dificuldades. O problema de determinar se existe alguma solução para um determinado nonograma é classificado NP-completo [15].

No capítulo 2, serão explicados alguns conceitos necessários para o entendimento do trabalho; no capítulo 3 será feita uma breve introdução a outros trabalhos relacionados a resolução de nonogramas. No capítulo 4, apresentamos a proposta de solução para o problema. No capítulo 5 expomos um experimento para testar a solução proposta. E por fim, no capítulo 6, discutimos as conclusões a que o experimento nos permitiu chegar.

2 Fundamentação Teórica

Este trabalho principalmente em dois conceitos para discutir a resolução de nonogramas. O primeiro é a classe de problemas de satisfatibilidade (ou SAT). E o segundo é a classe de problemas de satisfação de restrições (do inglês *Constrain Satisfaction Problem* ou CSP). Ambos serão melhor detalhados nas seções 2.1 e 2.2, respectivamente.

2.1 SAT

Em ciência da computação, o problema da satisfatibilidade booleana (às vezes abreviado para **SATISFATIBILIDADE** ou **SAT**) é o problema de determinar se existe uma interpretação que satisfaça uma determinada fórmula booleana. Em outras palavras, SAT quer saber se as variáveis em uma fórmula booleana podem ser consistentemente substituídas por valores **verdadeiro** ou **falso** de modo que a fórmula em si seja avaliada como **verdadeira**. Neste caso, a fórmula é dita satisfatiível.

Por outro lado, se tal atribuição não existir, a função expressa pela fórmula é falsa para todos os possíveis valores que podem ser atribuídos às variáveis e a fórmula é insatisfatível. Por exemplo, a fórmula $a \wedge \neg b$ é satisfatível, pois a atribuição $a = verdadeiro, b = falso$ resulta em verdadeiro. Porém, a fórmula $a \wedge \neg a$ é insatisfatível.

SAT foi o primeiro problema provado ser NP-completo[8]. O que significa que todos os problemas da classe de complexidade NP (que inclui uma grande variedade de problemas de decisão e otimização) são **no máximo** tão difíceis de resolver quanto o problema SAT. Não existe um algoritmo conhecido para resolver SAT de forma eficiente e, no geral, acredita-se que tal algoritmo não exista (ainda assim não existe prova matemática para nenhuma das afirmações) e a questão se SAT possui ou não um algoritmo de tempo polinomial é equivalente ao problema de P versus NP, um famoso problema em aberto no campo de computação.

Ainda assim, desde 2007, algoritmos SAT com heurísticas são capazes de resolver instâncias com dezenas de milhares de variáveis e fórmulas com milhões de símbolos, o que basta para diversos problemas práticos em SAT como design de circuitos e provas automáticas de teoremas.

Uma fórmula em lógica proposicional (também chamada de expressão booleana) é construída com variáveis, operadores **E** (conjunção, também representado por $\wedge$), **OU** (disjunção, $\vee$), **NOT** (negação, $\neg$) e parênteses. Uma fórmula é dita satisfatível se pode ser tornada verdadeira com a atribuição apropriada de valores lógicos (**Verdadeiro** e **Falso**) para suas variáveis.

Existem diversos casos especiais do problema SAT em que as fórmulas necessitam estar em determinadas estruturas. Um literal é uma variável (literal positivo) ou a negação de uma variável (literal negativo). Uma cláusula é uma disjunção de literais (ou um apenas um literal). Uma fórmula está na forma normal conjuntiva (CNF) se é uma conjunção de cláusulas (ou uma única cláusula). Neste trabalho tratamos de SAT apenas em CNF.

2.1.1 3-SAT

Da mesma forma que o problema de satisfatibilidade para fórmulas arbitrárias, determinar a satisfatibilidade de uma fórmula em CNF, em que todas as cláusulas estão limitadas a no máximo três literais, é NP-completo. Este problema é chamado de 3-SAT, 3CNFSAT ou 3-satisfatibilidade. Reduzir uma instância SAT irrestrita para 3-SAT resulta em uma fórmula que não é logicamente equivalente (ou seja a informação sendo representada não é mais a mesma), porém é equisatisfatível, ou seja, se a redução 3-SAT é satisfatível, sua contraparte irrestrita também o é.

3-SAT é um dos 21 problemas NP-Completos de Karp, e é usado como base provar que outros problemas também são NP-Difíceis. Isto é feito com uma redução em tempo polinomial de 3-SAT para outro problema.

3-SAT pode ser generalizada em k-SAT (ou k-CNF-SAT) quando as fórmulas em CNF são consideradas com cada cláusula contendo até k literais. Porém, como para qualquer $k \geq 3$ este problema não pode ser "mais fácil" do que 3-SAT nem "mais difícil" do que SAT, e como SAT é NP-Completo, k-SAT também deve ser NP-Completo.

Alguns autores restringem k-SAT a fórmulas em CNF com **exatamente** k literais. Isso não afeta a classe de complexidade do problema.

2.1.2 2-SAT

2-SAT é um caso especial de SAT, que pode envolver restrições em mais de duas variáveis, e de problemas de satisfação de restrições (clarificados na seção 2.2) que permitem mais de duas escolhas para o valor de cada variável. Mas diferentemente destes problemas mais gerais, que são NP-Completos, 2-SAT pode ser resolvido em tempo polinomial.

A satisfatibilidade de um problema 2-SAT pode ser descrita usando uma expressão booleana em uma forma especial mais restrita. É uma conjunção de cláusulas em que cada cláusula é uma disjunção de dois literais ou literais negados. Fórmulas nesse formato são conhecidas como fórmulas 2-CNF. O dois indica o número máximo de literais por cláusula na fórmula.

2.2 CSP

Problemas de satisfação de restrições (do inglês *Constraint satisfaction problems* ou **CSP**) são questões matemáticas definidas como um conjunto de variáveis cujas valorações devem satisfazer um número de restrições ou limitações. CSPs representam as entidades em um problema como uma coleção homogênea de restrições finitas sobre as variáveis, que são resolvidos com métodos de satisfação de restrições. CSPs são alvo de intensa pesquisa na área de inteligência artificial e pesquisa operacional, já que a regularidade em suas formulações provêm uma base comum para analisar e resolver problemas de famílias aparentemente sem relação. CSPs geralmente exibem alta complexidade, necessitando de uma combinação de heurísticas e busca combinatória para serem resolvidos em tempos razoáveis. O problema de satisfatibilidade booleana (SAT), pode ser visto como uma determinada forma de CSP.

Alguns problemas que podem ser modelados como CSPs incluem o problema das oito rainhas, o problema de colorir mapas, sudoku, palavras cruzadas, nonogramas e diversos outros quebra cabeças lógicos. Em casos mais gerais os problemas podem ser muito mais difíceis e até mesmo inexpressáveis em alguns dos sistemas mais simples. Exemplos de casos na

vida real incluem planejamento automático, desambiguação lexical, compreensão de sentenças, musicologia e alocação de recursos.

A existência da solução para um CSP pode ser visto como um problema de decisão. Isto pode ser decidido achando solução ou em caso de falha após testes exaustivos. Em alguns casos pode-se descobrir que o CSP possui uma solução de ante mão com inferência matemática.

De maneira formal, um CSP é definido como um trio $\langle X, D, C \rangle$ em que:

$X = \{X_1, ..., X_n\}$ é um conjunto de variáveis

$D = \{D_1, ..., D_n\}$ é um conjunto dos respectivos domínios de valor, e

$C = \{C_1, ..., C_m\}$ é um conjunto de restrições

Cada variável X_i pode receber um dos valores no domínio não-vazio D_i. Cada restrição $C_j \in C$ é em contrapartida um par $\langle t_j, R_j \rangle$ em que $t_j \subset X$ é um subconjunto de k variáveis e R_j é uma relação k-ária no subconjunto correspondente de domínios D_j. Uma avaliação das variáveis é uma função de um subconjunto de variáveis de um conjunto em particular de valores no subconjunto de domínios correspondente. Uma avaliação v satisfaz uma restrição $\langle tj, Rj \rangle$ se os valores atribuídos às variáveis em t_j satisfazem a relação R_j.

Uma avaliação é consistente se não viola nenhuma das restrições e é completa se inclui todas as variáveis. Uma avaliação é a solução se é consistente e completa. Esta avaliação resolve o problema de satisfação de restrições.

CSPs com domínios finitos são tipicamente resolvidos com alguma forma de busca. As técnicas mais comuns são retrocessos, propagação de restrições e buscas locais.

3 Revisão de literatura

O problema de resolver nonogramas não é um problema novo, e já foram feitas diversas propostas de resolvedores desses jogos. Essas propostas encaram o problema de formas diferentes, buscam ideias em áreas diferentes e possuem eficácias diversas para encontrar soluções.

Batenburg *et al*. em [4] faz uma análise da dificuldade de nonogramas, e após uma série de testes empíricos, concluem que tabuleiros com preenchimentos mais próximos a 50% são mais difíceis de resolver do que tabuleiros com preenchimentos mais próximos a 10% ou 90%.

Em [3] Batenburg *et al* mostram como construir nonogramas de diferentes níveis de dificuldade a partir de uma imagem em preto e branco, sendo a medida de dificuldade dada pelo algoritmo apresentado em [4].

Em [5] Batenburg *et al* propõem uma estratégia de resolução de nonogramas usando a lógica simples do jogo para fixar certas casas do tabuleiro em conjunto com algoritmos de tomografia discreta para achar uma representação em 2-SAT que pode ser facilmente resolvido com um grafo de dependências para achar uma solução parcial, e após algumas iterações encontrar a solução. Porém, a representação em 2-SAT pode falhar em descrever certas combinações que só podem ser representadas em 3-SAT e, portanto esta estratégia pode, por vezes, falhar em encontrar uma solução.

As soluções em [17] se baseiam na proposta de Batenburg. Propõe-se um algoritmo que resolve mais casas no tabuleiro antes do procedimento de busca começar a fazer retrocessos. Este caminho traz resultados melhores que Batenburg, mas sofre do mesmo problema de falhar em encontrar a solução de alguns nonogramas.

Yen *et al* [18] buscam todas as possíveis soluções considerando apenas linhas e colunas e se utilizam de funções para descobrir intersecções entre as soluções válidas para as linhas e soluções válidas para as colunas.

Outras soluções feitas com base nas regras do jogo são propostas em [12]. O artigo propõe duas heurísticas para resolver o problema baseadas na ideia de fixar casas que logicamente tem que estar preenchidas; baseando-se no fato de a casa manter o mesmo valor de preenchimento em possíveis soluções diferentes. Uma heurística utiliza-se de todos os possíveis preenchimentos da linha ou coluna, enquanto a outra observa apenas os preenchimentos mais à esquerda e mais à direita da linha ou coluna; O segundo método obteve maior sucesso.

Algumas das soluções propostas envolvem o uso de algoritmos genéticos. Bobko *et al*[7] propõem um algoritmo genético simples para resolver nonogramas utilizando-se das restrições providas como genes para o algoritmo, isso porém cria um espaço de busca muito grande. Percebendo este problema, Tsai *et al* [14] propõem uma codificação melhor para os genes e uma função de decisão que escolhe descendentes de forma melhor e obtém resultados melhores do que os apresentados em [7]. Alkhraisat *et al* [1] propõem o uso de um algoritmo de otimização de enxame de partículas com inércia dinâmica no peso e obtêm resultados bons na resolução de nonogramas.

Wang *et al* [16] cita os problemas existentes com soluções heurísticas e soluções baseadas em algoritmos genéticos e propõem um algoritmo de têmpera simulada para resolver

nonogramas. Na solução proposta, o tabuleiro é inicialmente preenchido de forma aleatória e o algoritmo escolhe um par de casas de cores diferentes para trocar de posição baseado em uma métrica de contradição na solução. Conforme o tempo passa as trocas ficam menos frequentes e mais difíceis de achar. É necessário tomar um certo cuidado com o algoritmo que decide em qual par de casas trocar para que ele não escolha sempre a melhor troca e arrisque ficar preso em um ótimo local, assim como não ter escolhas aleatórias demais para evitar perder uma solução parcial.

Mingote *et al* [10] propõem uma solução que utiliza programação linear inteira para resolver nonogramas coloridos, precisando de poucas modificações nas funções para ser aplicado em nonogramas monocromáticos. Separa o problema em linhas e colunas e define que deve haver um acordo com a cor de determinada casa em ambas as soluções. A solução proposta se baseia em um cálculo do limite mínimo de onde cada sequência pode começar e do limite máximo em que cada sequência pode ser encerrada para resolver o nonograma.

Tamura *et al* [13] propõem utilizar codificação ordenada para melhor transformar problemas de satisfação de restrições (CSP) em instâncias SAT e obtêm bons resultados.

Inspirados pela solução proposta por Batenburg [5], optamos por representar um nonograma como uma instância de um CSP, que pode ser traduzido para uma instância 3-SAT e solucionado por um resolvedor de problemas SAT, ao invés de utilizar heurísticas para gerar instâncias 2-SAT que podem não resolver o problema (a busca por uma forma de fazer isso foi o que nos levou a encontrar o artigo de Tamura *et al* [13]). A forma como fizemos esta transformação está descrita de forma mais detalhada a seguir, no Capítulo 4.

4 Proposta

A ideia por trás de descartar as heurísticas que transformam o problema em instâncias 2-SAT e utilizar uma conversão mais simples para 3-SAT é que considerando a eficiência de resolvedores SAT modernos como Minisat [9], lingueling [6] e Glucose [2] é possível que as heurísticas estejam gerando informações desnecessárias para a instância SAT e, portanto, uma transformação direta, não só cobriria o problema de instâncias não representáveis em 2-SAT encarado por Batenburg [5], como também poderia ser uma forma mais rápida de alcançar a solução.

Pesquisando por formas de codificar e resolver nonogramas como instâncias de um CSP, encontramos um programa chamado *Sugar* [13] baseado na proposta de Tamura *et al* [13]. O programa recebe uma instância de um CSP, e retorna a solução da instância. Isto significa que dado uma instância de nonograma codificada em CSP é possível achar a solução do nonograma utilizando-se do *Sugar*. De forma alternativa o programa pode ser utilizado para gerar a codificação SAT do nonograma na forma normal conjuntiva (CNF) caso seja desejado manter uma instância de um nonograma e não apenas a solução final.

Para representar um nonograma como uma instância CSP formamos algumas regras (ou restrições) que são usadas na resolução. As duas primeiras regras são obrigatórias e devem sempre ser seguidas, a primeira determina quais os valores válidos que cada casa no tabuleiro pode possuir (no caso com branco sendo 0, preto sendo 1, outras cores representadas por outros números inteiros quando presentes). A segunda regra instancia todas as sequências de cada linha e de cada coluna, para que as posições delas no tabuleiro possam ser decididas, a única restrição neste caso é que a casa em que uma sequência começa é no máximo o fim da linha (ou coluna) subtraído do seu tamanho, para que a sequência caiba na linha (ou coluna).

Criamos uma versão alternativa para a regra dois que tenta posicionar melhor os limites de onde cada sequência pode ficar, considerando as outras restrições presentes. Ao contrário da regra dois original que considera apenas que cada sequência tem que pertencer a linha (ou coluna), nossa versão alternativa leva em consideração que o limite máximo para a casa em que uma sequência comece não é simplesmente o tamanho da linha (ou coluna) menos o tamanho da sequência, é necessário também considerar as outras sequências que existem, e os espaços em branco obrigatórios que existem entre sequências na mesma linha (ou coluna). Estar regra não remove a necessidade das regras de ordenação de sequências pois ainda pode haver sobreposição entre sequencias adjacentes.

O resto das regras cuida de tentar valorar as casas do tabuleiro individualmente, elas podem ser usadas em conjunto umas com as outras, ou separadamente, contanto que no mínimo uma delas esteja sempre presente na resolução do problema.

A terceira regra utilizada serve para ordenar as sequências em cada linha e cada coluna, definindo que para uma determinada linha ou coluna, a sequência **um** inicia na casa antes da casa de início da sequência **dois**, mais **um** (para contar o espaço em branco obrigatório).

Estas 3 primeiras regras foram as propostas por Tamura *et al*[13] em seu website quando demonstravam como utilizar o programa *Sugar* para resolver nonogramas; nós criamos outras

quatro regras adicionais que podem ser utilizadas. A quarta regra é basicamente o inverso da regra três, limitando que o final da sequência **n** acontece após o final da sequência **n - 1**. A quinta e a sexta regras são ambas baseadas na proposta de Oliveira [11], que conclui que problemas SAT que conseguem manter a propriedade de arco consistência podem ser resolvidos com menos tomadas de decisão ao custo de um aumento no tempo de resolução; A quinta é regra o fecho da terceira (dizendo que a sequência **um** acontece antes das sequência **dois**, **três** e assim por diante, e a sexta regra faz de forma similar o fecho da quarta regra).

Para a geração de nonogramas aleatórios foi utilizado um programa em python que recebia como parâmetros de entrada o tamanho do nonograma, a porcentagem do nonograma que deveria estar preenchida e a semente de aleatorização, e retornava um arquivo com as restrições em formato mk. O programa gerava uma matriz quadrada com o tamanho recebido da entrada, e para cada casa da matriz gerava um número aleatório a partir de uma semente que também veio da entrada. Se o número fosse menor ou igual a porcentagem de preenchimento a casa seria pintada de preto, caso contrário seria branca. Assumimos que com matrizes grandes o suficiente isso obteria matrizes com o preenchimento desejado. Não garantimos quantas soluções diferentes podem existir para cada nonograma gerado, temos apenas a garantia da existência de ao menos uma solução Por fim o programa percorria a matriz e gerava um arquivo do tipo “.mk” com a matriz obtida.

Um “.mk” é um arquivo em um formato específico para armazenamento para nonogramas, a primeira linha contém o número de linhas e colunas do nonograma. Abaixo disso estão as restrições propriamente ditas, cada linha equivale a uma linha de restrições no nonograma e portanto pode ter vários números. Restrições vazias são representadas por um 0. Na figura 4.1 está um exemplo de um nonograma simples, e sua codificação em “mk” logo abaixo.

	1	1
1		
1		

Figura 4.1: Nonograma simples vazio

```
2 2
1
1
#
1
1
```

O símbolo de "#" serve para separar as restrições da horizontal e da vertical.

Este arquivo é então passado para um outro programa que interpreta o nonograma e o descreve em uma linguagem própria do programa *Sugar* para representar o nonograma como um conjunto de restrições. Seguindo ainda o nonograma da figura 4.1, ele seria representado da seguinte forma:

- Regra 1:

$$(int\,x_{00}\ \ 0\ \ 1)$$

$$(int\,x_{01}\ \ 0\ \ 1)$$

$$(int\,x_{10}\ \ 0\ \ 1)$$

$$(int\,x_{11}\ \ 0\ \ 1)$$

A primeira regra irá marcar que todas as casas ([0,0],[0,1][1,0],[1,1]) podem receber valores entre 0 (branco) e 1 (preto). Note pela contagem das casas que consideramos que a primeira linha é a linha 0, e de forma similar consideramos a primeira coluna a coluna 0. A representação acima é a forma como estas informações são passadas para o programa *Sugar*

- Regra 2:

$$(int\,h_{00}\ \ 0\ \ 1)$$

$$(int\,h_{10}\ \ 0\ \ 1)$$

$$(int\,v_{00}\ \ 0\ \ 1)$$

$$(int\,v_{10}\ \ 0\ \ 1)$$

A segunda regra irá marcar que a sequência da primeira restrição horizontal pode começar na coluna 0 até a coluna 1 (uma regra de tamanho 1 pode estar em qualquer lugar na linha), de forma similar a restrição na segunda linha irá ser marcada como podendo começar na coluna 0 até a coluna 1. O mesmo ocorre com as restrições nas colunas em que ambas as colunas são marcadas com a sequência podendo começar entre a linha 0 e a linha 1. Neste nonograma em particular não há diferença entre a aplicação da regra 2 padrão ou alternativa, pois a apenas uma restrição por linha ou coluna. As duas primeiras linhas são referentes as restrições da primeira e segunda linha respectivamente, as duas ultimas fazem o mesmo para a primeira e segunda coluna.

$(predicate\ \ (r_0\ \ j)\ \ (and\ \ (<= h_{00}\ \ j)(<\ \ j\ \ (+\ \ h_{00}\ \ 1))))$

$(iff\ \ (=\ \ x_{00}\ \ 1)\ \ (r_0\ \ 0))$

$(iff\ \ (=\ \ x_{01}\ \ 1)(r_0\ \ 1))$
$(predicate(r_1\ \ j)(and\ \ (<= h_{10}\ \ j)(<\ j\ \ (+h_{10}\ \ 1))))$

$(iff\ \ (=\ \ x_{10}\ \ 1)\ \ (r_1\ \ 0))$

$(iff\ \ (=\ \ x_{11}\ \ 1)\ \ (r_1\ \ 1))$

$(predicate\ \ (r_2\ \ j)\ \ (and\ \ (<= v_{00}\ \ j)\ \ (<\ \ j\ \ (+\ \ v_{00}\ \ 1))))$

$(iff\ \ (=\ \ x_{00}\ \ 1)\ \ (r_2\ \ 0))$

$(iff\ \ (=\ \ x_{10}\ \ 1)\ \ (r_2 1))$

$(predicate\ \ (r_3 j)\ \ (and\ \ (<=\ \ v_{10}\ \ j)\ \ (<\ \ j\ \ (+\ \ v_{10}\ \ 1))))$

$(iff\ \ (=\ \ x_{01}\ \ 1)\ \ (r_3\ \ 0))$

$(iff\ \ (=\ \ x_{11}\ \ 1)\ \ (r_3\ \ 1))$

Ainda na regra 2, esta é a forma de marcar os posicionamentos das sequências. O *predicate* funciona de forma similar a uma função em outras linguagens de programação, ele é utilizado para evitar ter que constantemente escrever regras que diferem por apenas um valor.

Esta notação é prefixada, ou seja, o operador aparece antes dos operandos, neste caso o "iff" significa uma dupla implicação entre os operandos, ou seja a casa x_{ab} será valorada como 1 (preto) se e somente se as condições do predicado forem atendidas, caso contrário receberá outro valor (no caso de nonogramas monocromáticos isto significa branco, uma implementação com nonogramas coloridos precisaria de outro predicado para negar todas as outras cores quando a casa fosse ser branca).

A figura 4.2 abaixo mostra em amarelo as casas do nonograma em que é possível que as sequências comecem como ditado pela regra 2:

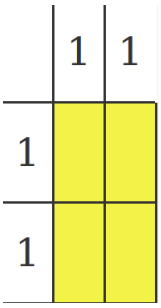


Figura 4.2: Nonograma simples marcado

- outras regras:
  Como este nonograma possui apenas 1 restrição por linha ou coluna não foram geradas

as regras 3, 4, 5 e 6. Caso houvesse múltiplas restrições na mesma linha ou coluna, a regra 3 gerada seria similar a:

$$(< \quad (+ \quad h_{10} \quad T) \quad h_{11})$$

A regra quatro seria similar a:

$$(> \quad h_{10})(+ \quad h_{11} \quad T)$$

Em que T é o tamanho da restrição h_{10}, indicando que a restrição h_{10} começa no máximo T casas antes de h_{11}, a conta é estritamente menor por levar em consideração o espaço obrigatório entre sequências de mesma cor.

Geradas as regras elas são escritas em um arquivo ".csp" que o resolvedor utiliza para descobrir que o nonograma é o que está na figura 4.3 (ou o da figura 4.4, já que este é um nonograma ambíguo, e pode ser usado como exemplo de um que a solução de Batenburg *et al* [5] teria problemas de resolver). No capítulo 5 mostramos como este arquivo ".csp" é transformado em uma soluçao para o nonograma.

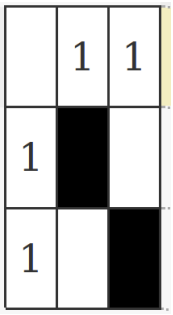


Figura 4.3: Solução de nonograma simples

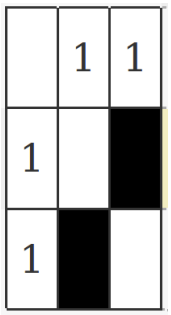


Figura 4.4: Solução alternativa de nonograma simples

Para verificar a corretude e o desempenho de nossa solução fizemos um experimento simples com combinações diferentes de regras e diferentes nonogramas que será explicado em maiores detalhes no capítulo 5.

5 Avaliação Experimental

Para testar o desempenho de nossa proposta, elaboramos um teste com nonogramas de diversos tamanhos e preenchimentos, codificando cada um com diferentes combinações de regras e utilizando três resolvedores SAT diferentes para obter os dados resultantes de nossa solução.

5.1 Sugar

O arquivo ".csp" gerado conforme mostrado no capítulo 4 é então passado para o programa *Sugar*, que serve para transformar a descrição das restrições em um arquivo ".cnf" contendo uma representação em CNF do nonograma, que é passado para um resolvedor SAT automático.

O resolvedor a ser utilizado é passado como um parâmetro do programa, e parâmetros adicionais podem ser passados para que os arquivos temporários (o ".cnf" incluso) não sejam removidos automaticamente pelo *Sugar* após ele dar a resposta, e para que ele não chame o resolvedor, apenas gere os arquivos.

Nós optamos por usar o *Sugar* apenas como gerador pois ele é um programa em Java e não queríamos ter que lidar com a necessidade de gerar e resolver o nonograma toda vez que quiséssemos testar (O *Sugar* sempre traduz de CSP para CNF e chama um resolvedor SAT, salvo quando as opções certas são passadas para que ele apenas gere o arquivo em CNF sem tentar resolver).

5.2 Os testes

Para a resolução dos nonogramas, optamos por utilizar três resolvedores diferentes, MiniSAT, Glucose e Lingeling. Geramos nonogramas de 4 tamanhos diferentes (20x20, 30x30, 40x40 e 50x50), com 5 diferentes níveis de preenchimento (10%, 20%, 30%, 40%, 50%, as outras porcentagens a partir de 60% não foram utilizadas pois são, de certa forma, o inverso das porcentagens utilizadas, apenas substituindo preto por branco). Para cada combinação de tamanho e preenchimento foram gerados 128 nonogramas diferentes e cada um foi codificado de acordo com combinações diferentes de regras, cada uma das regras três a seis individualmente, assim como as combinações da regra três com a regra quatro, e de forma similar da regra cinco com a seis, e para cada um desses foi ainda gerado uma versão utilizando a regra dois padrão e a regra dois alternativa.

Todos estes nonogramas foram resolvidos por cada um dos três resolvedores com o tempo limite de execução de cinco minutos antes de serem cancelados. Os nonogramas que excederam o tempo limite não foram considerados para as análises feitas.

5.3 Resultados

Os gráficos 5.1, 5.2 e 5.3 a seguir mostram a média de decisões tomadas pelos resolvedores glucose, lingueling e MiniSAT respectivamente, para resolver os nonogramas de tamanho 30x30, conforme o preenchimento do nonograma aumenta.

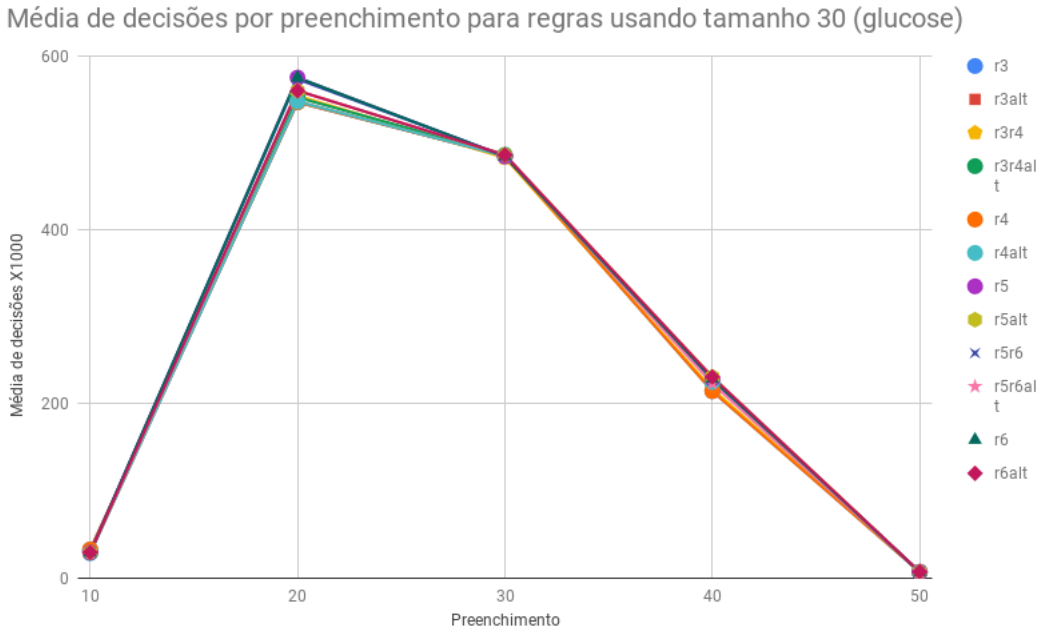


Figura 5.1: Média de decisões tomadas para cada regra aplicada em nonogramas 30x30 pelo resolvedor glucose

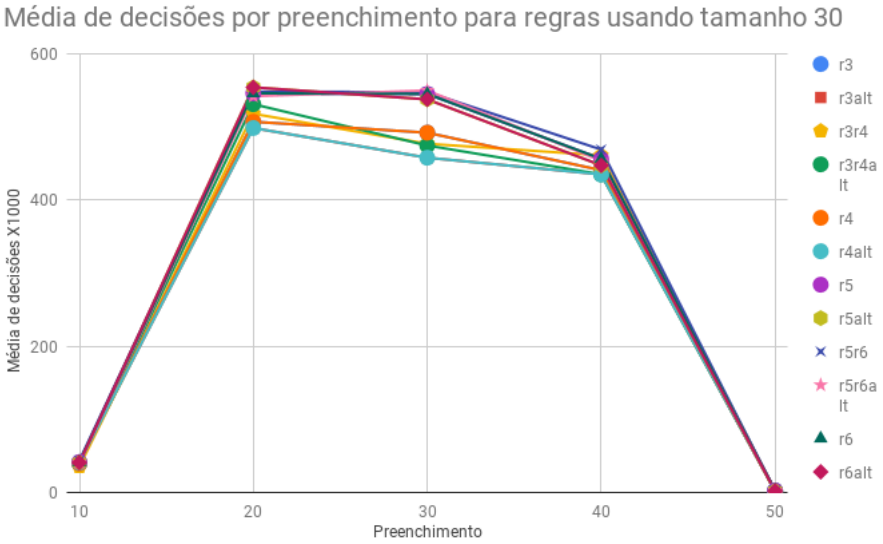


Figura 5.2: Média de decisões tomadas para cada regra aplicada em nonogramas 40x40 pelo resolvedor lingueling

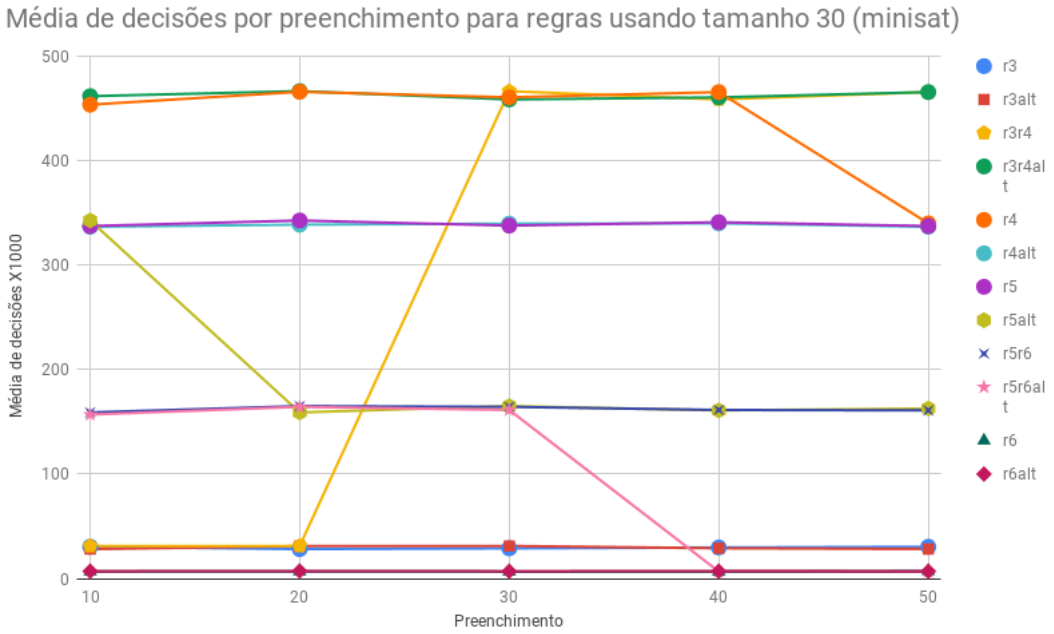


Figura 5.3: Média de decisões tomadas para cada regra aplicada em nonogramas 30x30 pelo resolvedor MiniSAT

Como o resolvedor MiniSAT foi o único a apresentar diferença no número de decisões para cada regra decidimos focar nossa análise nos resultados obtidos por ele.

Olhando para os gráficos 5.3 e 5.4 é possível verificar que em grande parte dos casos as instâncias que foram geradas utilizando a versão alternativa da regra dois tomaram muito menos decisões do que suas contrapartes com a regra dois padrão. Assim como é fácil de ver que as instâncias geradas com as regras cinco e seis em conjunto ou apenas com a regra seis conseguiram resolver os nonogramas com pouquíssimas decisões (corroborando a proposta de Oliveira [11]), principalmente para os nonogramas com preenchimentos mais próximos a 50% (contrário ao que se esperava pela proposta de Batenburg *et al.* [4]).

O gráfico 5.5 abaixo mostra em média o tempo em segundos que cada regra levou para resolver os nonogramas de diferentes preenchimentos. No geral, em termos de tempo as regras diferentes mantiveram um tempo estável ao longo de diferentes preenchimentos, há porém três exceções notáveis a este padrão, a regra cinco pareada com a versão alternativa da regra dois, assim como a combinação das regras três e quatro tiveram um aumento significativo no tempo para achar a resolução a partir de nonogramas com 20% e 30% de preenchimento respectivamente. Outra regra notável foi a combinação das regras cinco e seis que vem uma redução drástica no tempo de resolução para nonogramas mais complexos (de acordo com a complexidade relativa a preenchimento encontrada por Batenburg *et al.* em [4]).

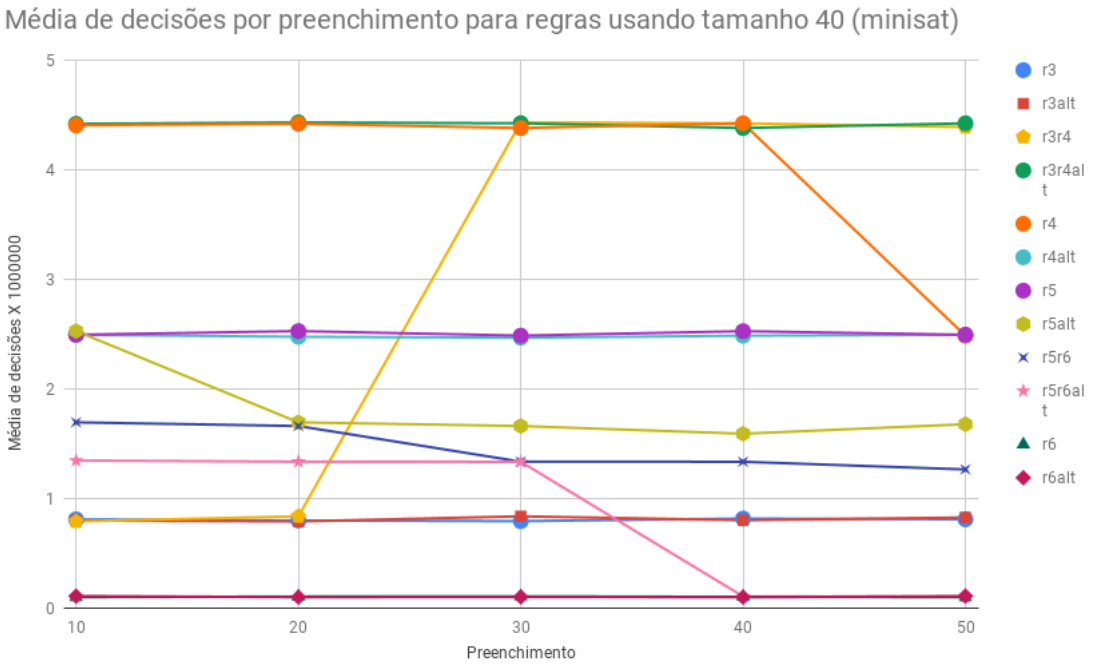


Figura 5.4: Média de decisões tomadas para cada regra aplicada em nonogramas 40x40 pelo resolvedor MiniSAT

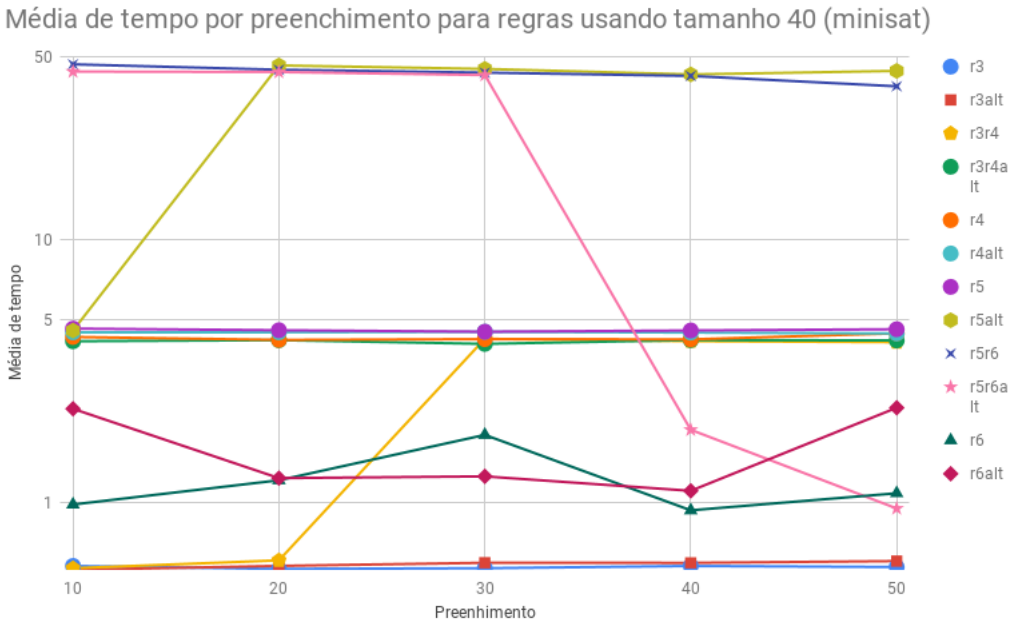


Figura 5.5: Média de tempo levado por regra para cada preenchimento 40x40

Um fator que reforça o aumento da complexidade de nonogramas com porcentagens de preenchimento mais próximas de um determinado preenchimento são os gráficos 5.6 e 5.7, que mostra o aumento significativo no tempo para resolução conforme o preenchimento do nonograma aumenta, o efeito é muito mais visível em nonogramas maiores, porém contrário a proposta de Batenburg *et al.* em [4] o pico foi nos nonogramas com 40% de preenchimento.

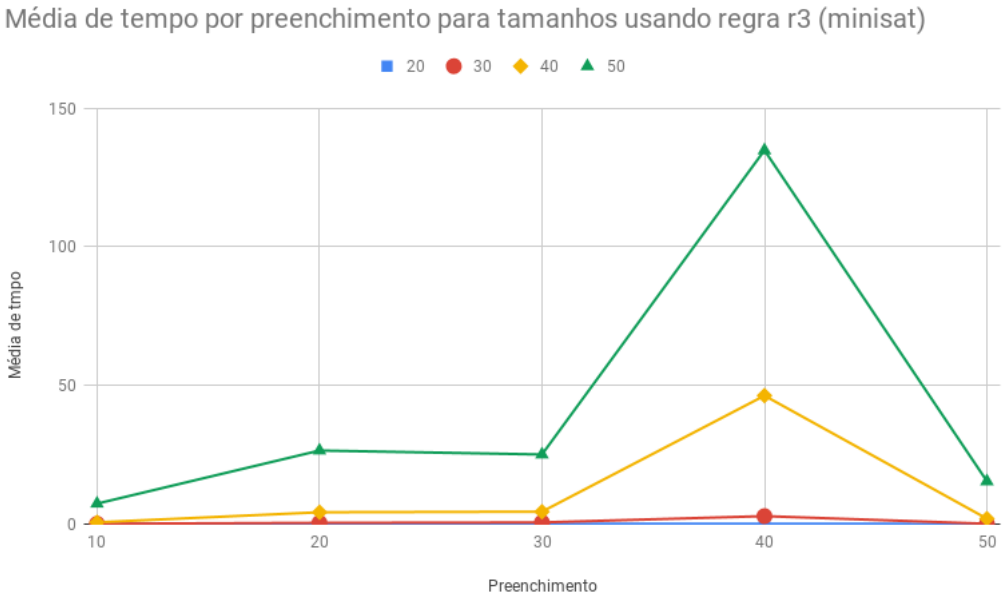


Figura 5.6: Média de tempo levado por preenchimento para cada tamanho pela regra 3

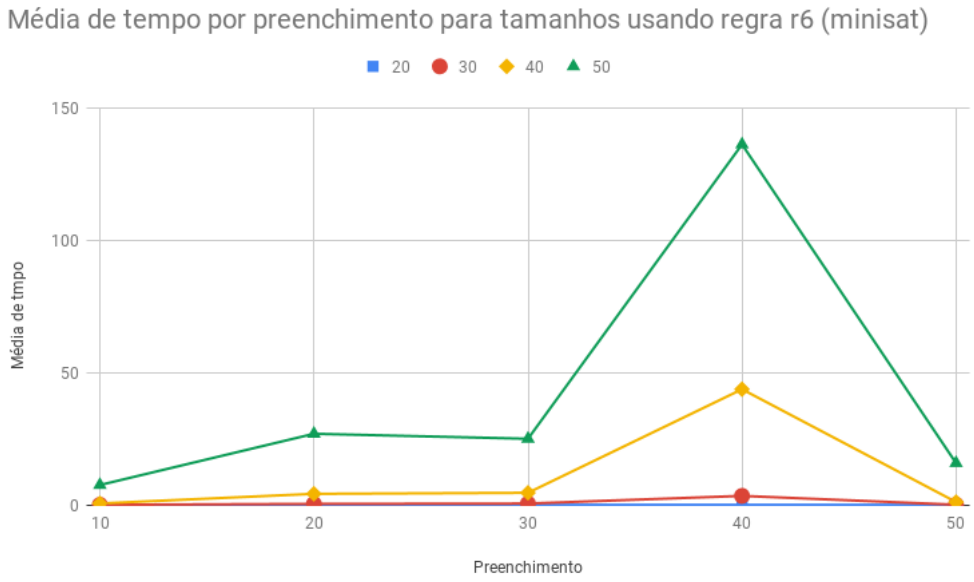


Figura 5.7: Média de tempo levado por preenchimento para cada tamanho pela regra 6

Como é possível ver nos gráficos comparado com a codificação original sugerida por Tamura *et al* [13] para conversão de nonogramas para instâncias do problema SAT, nossa versão alternativa para a regra dois obteve um desempenho melhor em quase todas as instâncias comparado a regra dois padrão, a única exceção sendo na combinação da regra três com a regra quatro, em que a versão padrão começou melhor e a partir dos nonogramas de tamanho 20x20 obteve uma performace igual a alternativa.

Além disso, das regras adicionais que introduzimos, a regra seis obteve resultados muito melhores do que as outras regras (e combinações de regras) que foram utilizadas nos testes, novamente reforçando a proposta de Oliveira [11] sobre instâncias SAT com fecho. Porém, a

regra cinco, sendo o fecho da regra três, precisou de um número muito similar de decisões para resolver os nonogramas aos da regra três, suspeitamos que a diferença (ou falta dela) observada entre o número de decisões pelas regras três e cinco, comparado à diferença grande entre as regras quatro e seis (que é o fecho da regra quatro), se devem a ordenação das variáveis pelo resolvedor na hora de buscar a solução.

Porém ao longo da execução dos testes percebemos que a quantidade de nonogramas que excedem o limite de tempo era mais perceptível nos nonogramas com tamanho 40x40 e 50x50, e é possível que nossos resultados sejam afetados pelo fato que as instâncias que terminaram de executar dentro do tempo limite tenham sido, pelo próprio processo de geração aleatorizado, instâncias mais fáceis de serem resolvidas.

6 Conclusão

Neste trabalho exploramos diversas soluções existentes para resolução de nonogramas, e a partir delas elaboramos nossa própria solução, que envolveu a criação de novas regras para a codificação de nonogramas em instâncias do problema SAT. Elaboramos uma bateria de testes para analisar o desempenho das regras que propusemos e concluímos que a nossa proposta de regra para codificar de forma diferente as limitações das sequências dentro das linhas/colunas obteve resultados muito melhores do que a codificação que utilizamos como base de comparação para as outras regras que elaboramos.

Também fomos capazes de verificar a proposta de Oliveira [11] a respeito de regras SAT com fecho e vimos que, de fato, a aplicação do fecho diminui o número de decisões que precisam ser tomadas pelo resolvedor. A razão pela diferença grande no desempenho entre as regras cinco e seis ainda precisa ser investigada para ver se foi causada pela ordem de escolha de variáveis pelo resolvedor, pelas instâncias escolhidas serem particularmente favoráveis a codificação da regra seis ou se a quantidade de nonogramas que escolhemos resolver foi pequena demais e não representa a dificuldade do problema apropriadamente.

Referências

[1] Habes Alkhraisat and Hasan Rashaideh. Dynamic inertia weight particle swarm optimization for solving nonogram puzzles. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 7(10):277–280, 2016.

[2] Gilles Audemard and Laurent Simon. Glucose 2.1: aggressive-but reactive-clause database management, dynamic restarts. In *International Workshop of Pragmatics of SAT (Affiliated to SAT)*, 2012.

[3] K Joost Batenburg, Sjoerd Henstra, Walter A Kosters, and Willem Jan Palenstijn. Constructing simple nonograms of varying difficulty. *Pure Mathematics and Applications (Pu. MA)*, 20:1–15, 2009.

[4] K Joost Batenburg and Walter A Kosters. On the difficulty of nonograms. *ICGA Journal*, 35(4):195–205, 2012.

[5] Kees Joost Batenburg and Walter A Kosters. Solving nonograms by combining relaxations. *Pattern Recognition*, 42(8):1672–1683, 2009.

[6] Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In Tomáš Balyo, Marijn Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, pages 14–15. University of Helsinki, 2017.

[7] Alicja Bobko and Tomasz Grzywacz. Solving nonograms using genetic algorithms. In *Computational Problems of Electrical Engineering (CPEE), 2016 17th International Conference*, pages 1–4. IEEE, 2016.

[8] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[9] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[10] Luís Mingote and Francisco Azevedo. Colored nonograms: an integer linear programming approach. In *Portuguese Conference on Artificial Intelligence*, pages 213–224. Springer, 2009.

[11] Ricardo Tavares de Oliveira. Arco consistência generalizada em codificações sat relativas. 2017.

[12] Sancho Salcedo-Sanz, Emilio G Ortiz-Garca, Angel M Pérez-Bellido, Antonio Portilla-Figueras, and Xin Yao. Solving japanese puzzles with heuristics. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 224–231. IEEE, 2007.

[13] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear csp into sat. *Constraints*, 14(2):254–272, 2009.

[14] Jinn-Tsong Tsai. Solving japanese nonograms by taguchi-based genetic algorithm. *Applied Intelligence*, 37(3):405–419, 2012.

[15] Nobuhisa Ueda and Tadaaki Nagao. Np-completeness results for nonogram via parsimonious reductions. *preprint*, 1996.

[16] Wen-Li Wang and Mei-Huei Tang. Simulated annealing approach to solve nonogram puzzles with multiple solutions. *Procedia Computer Science*, 36:541–548, 2014.

[17] I-Chen Wu, Der-Johng Sun, Lung-Ping Chen, Kan-Yueh Chen, Ching-Hua Kuo, Hao-Hua Kang, and Hung-Hsuan Lin. 12. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3):251–264, 2013.

[18] Shi-Jim Yen, Tsan-Cheng Su, Shih-Yuan Chiu, and Jr-Chang Chen. Optimization of nonogram's solver by using an efficient algorithm. In *Technologies and Applications of Artificial Intelligence (TAAI), 2010 International Conference on*, pages 444–449. IEEE, 2010.