UNIVERIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
PROGRAMA DE MESTRADO E DOUTORADO

André Castro Ramos

COMPLEXIDADE E ALGORITMOS DE JOGOS DE BLOCOS

FORTALEZA – CE
Julho de 2014

André Castro Ramos

COMPLEXIDADE E ALGORITMOS DE JOGOS DE BLOCOS

Dissertação de Mestrado apresentada ao Programa de Mestrado e Doutorado em Ciência da Computação, do Departamento de Computação da Universidade Federal do Ceará, como requisito parcial para obtenção do Título de Mestre em Ciência da Computação. Área de concentração: Otimização (Teoria da Computação).

Orientador: Prof. Dr. Rudini Menezes Sampaio

FORTALEZA – CE
Julho de 2014

Agradecimentos

Agradeço a todos os membros do ParGO, que forneceram um ambiente propício ao desenvolvimento deste trabalho, que tem significou um momento crucial em minha existência. Especialmente ao professor Rudini, que orientou e tornou possível que os resultados fossem como são agora. Agradeço também a todos os parentes e amigos fora do grupo que incentivaram que eu ingressasse e chegasse ao passos finais de um mestrado. Em especial a meus pais, que estiveram comigo em todos os momentos.

"Como modelos computacionais, jogos e quebra-cabeças trazem uma nova visão sobre uma teoria e mostram que ela pode ser mais concreta do que a matemática simbólica que a define."

Resumo

A noção de jogo eletrônico remete a entretenimento reservado às horas vagas, mas, além de uma indústria bilionária, também é origem potencial de diversos temas de pesquisa, tanto voltados a suas respectivas áreas quanto de interesse da própria indústria de jogos. Nesse contexto, nas últimas décadas, foram produzidos trabalhos que lidam com esse tipo de produto como base para problemas a serem tratados pela teoria dos algoritmos. Neste trabalho trazemos resultados de complexidade e algoritmos relacionados a 3 jogos com características em comum, Bloxorz, On The Edge e Bobbin 3D.

Abstract

The notion of electronic game is often related to mere entertainment for free time, but, not only a bilionaire industry, electronic gaming is also basis to many research themes focused on their own respective fields or on the gaming industry itself. In that context, in the last few decades, research has been made dealing with that kind of product as a basic framework for stating problems in Computer Science terms. In this work we show complexity results and algorithms about 3 games with important similarities, Bloxorz, On The Edge and Bobbin 3D.

Sumário

1 Introdução

Na área de Algoritmos, utilizam-se objetos matemáticos como vetores, matrizes e grafos. Um exercício interessante para testar os limites da área de Algoritmos é trabalhar com conceitos diferentes para definir os problemas. Uma fonte possível de conceitos matemáticos são os jogos eletrônicos, que apesar de todas as questões motoras e artísticas, possuem regras matemáticas. Nesta dissertação, analisamos vários problemas de jogos eletrônicos sob o enfoque da área de Algoritmos e Complexidade Computacional. Após uma revisão introdutória, apresentamos nossos resultados a partir do Capítulo 3 sobre uma família específica de jogos, a saber, jogos de bloco. Os nomes protegidos por direito autoral utilizados aqui o são com propósito acadêmico, portanto de forma legal.

O estudo teórico de objetos entedidos como recreativos tem já um longo histórico. Um caso clássico é o do problema das oito damas, uma questão baseada nas regras do xadrez originalmente proposta por Max Bezzel e estudada por vários matemáticos, incluindo Gauss. No que se refere a estabelecer problemas computacionais a partir de jogos eletrônicos e similares e analisar a complexidade deles sob a ótica da teoria de Algoritmos, abaixo são cidados trabalhos relevantes relacionados ao assunto.

Em [Demaine, Demaine et al., 2007], é demonstrada a NP-completude de problemas que representam quebra-cabeças, onde busca-se completar uma imagem com peças dotadas de propriedades de encaixe.

Em [Demaine, Demaine et al., 2004] estuda-se um jogo concebido em 2001, chamado *Clobber*, que faz parte de competições de IA. Nesse jogo, há um tabuleiro onde algumas posições possuem uma peça preta ou uma peça branca. Uma jogada possível é mover uma peça branca para uma peça preta adjacente, removendo a preta do tabuleiro (ou vice-versa, movendo a preta para a posição da branca). Prova-se que é NP-Completo decidir se existe uma sequência de jogadas que deixa o tabuleiro com apenas uma única peça.

Em [Demaine, Demaine et al., 2006] é demonstrada a inaproximabilidade por um fator $n^{(1-\varepsilon)}$, para qualquer $\varepsilon > 0$, de um problema extraído de um jogo de lápis e papel conhecido como *Join Five* ou *Morpian Solitaire*, onde n é o número de pontos iniciais. Nesse jogo há alguns pontos desenhados sobre um grid (na versão mais popular, os pontos iniciais formam uma cruz). Uma jogada possivel é incluir um ponto novo no grid que torne possível traçar um segmento (na horizontal, vertical ou diagonal) passando por 5 pontos vizinhos do grid, incluindo o ponto novo. Além do resultado de inaproximabilidade, são obtidos no artigo limites superiores e inferiores relacionados.

Em [Flake e Baum, 2002], estuda-se o jogo *Rush Hour*, um jogo cujo objetivo é tirar um carro especial de um "estacionamento" cheio de outros carros. Os carros podem mover-se apenas horizontalmente ou verticalmente de cada vez. O objetivo é levar o carro especial

até a saída do estacionamento. Provou-se que é PSPACE-Completo decidir se é possivel tirar o carro especial do estacionamento.

Em [Fraenkel e Goldschmidt, 1987], é provada a PSPACE-completude de diferentes problemas em jogos competitivos cujo "tabuleiro" é um grafo direcionado.

Em [Aloupis et al., 2012], os autores conseguiram traduzir jogos clássicos da Nintendo em problemas computacionais com regras claras e, com isso, obtiveram resultados algorítimicos e de complexidade. Especificamente, os jogos-base foram os das franquias Mario, Donkey Kong, Legend of Zelda, Metroid e Pokémon. O objetivo é saber se é possivel terminar uma determinada fase do jogo, ou seja, se há uma sequência de movimentos que levem até o fim da fase. Os principais resultados obtidos foram a NP-completude de cada problema. Além disso, com relação aos jogos da franquia Zelda, também foi provada a $PSPACE$-completude do problema de decisão.

Já em [Demaine et al., 2004], artigo relacionado ao Tetris, famoso jogo eletrônico, prova-se a NP-completude e a inaproximabilidade por um fator $n^{(1-\varepsilon)}$, para qualquer $\varepsilon > 0$, onde n é o número de peças envolvidas no jogo. Nesse artigo, é apresentada uma formalização matemática mais precisa, visto que o jogo Tetris é mais fácil de formalizar do que os jogos da Nintendo. Em linhas gerais, esse modelo coloca como entrada uma matriz que representa a tela do jogo, que não precisa estar inicialmente em branco, e uma sequência de peças de Tetris. A saída do problema é uma sequência de movimentos válidos (noção também definida no modelo) com o objetivo de minimizar ou maximizar uma função especifica do problema em questão, como maximizar o número de linhas eliminadas. Esse artigo se concentra em 4 funções objetivo: número de linhas eliminadas, número de peças utilizadas até um movimento que cause a derrota, número de movimentos que eliminem 4 linhas simultaneamente e altura máxima de uma posição ocupada da matriz ao longo da sequência de movimentos, sendo que apenas nesse último exemplo o problema é tratado como de minimização.

O artigo [Dor e Zwick, 1999] é um dos mais antigos voltados a jogos eletrônicos, no caso o jogo *Sokoban*, um jogo cujo objetivo é posicionar blocos corretamente. Nesse artigo, existe o cuidado explícito de separar o problema, que imediatamente é classificado como um problema de planejamento de movimentos, do jogo em si. É provada a NP-completude da abstração considerada e a $PSPACE$-completude de uma versão alternativa onde é possível mover 2 blocos de uma vez.

Um modo versátil de se estudar matematicamente jogos eletrônicos é encontrado e aplicado sucessivamente no livro "*Games, puzzles and Computation*" [Hearn e Demaine, 2009] e também aplicado no artigo [Demaine e Hearn, 2005]. Esse estudo se baseia em um tipo de "jogo" sobre um grafo não orientado. É dada uma configuração (que é basicamente uma orientação do grafo original) e cada jogada possível é a inversão de um arco da configuração. Sem entrar em muitos detalhes, nem todos os arcos podem ser invertidos em determinada configuração, pois é preciso satisfazer algumas condições

relacionadas a pesos nos arcos e nos vértices. Em termos gerais, o objetivo é alcançar, através do encadeamento de jogadas válidas, uma configuração onde um determinado arco especial esteja invertido. Em versões de mais de um jogador, os arcos são alternadamente invertidos por jogadores diferentes e cada jogador tem seu arco objetivo. Em versões com movimentos limitados, um arco só pode ser invertido uma única vez.

É possivel ainda uma versão desse jogo com 0 jogadores, onde a cada passo existe apenas uma única possibilidade de arco para ser invertido. Em um jogo com pelo menos 2 jogadores, cada jogador precisa tentar vencer "pensando" nas possíveis jogadas de seus adversários (e possivelmente evitando a vitória dos adversários antes dele).

No livro "*Games, puzzles and Computation*" [Hearn e Demaine, 2009] prova-se as complexidades das variantes desse jogo, descritas na tabela abaixo.

Movimentos	Zero jogadores	Um jogador	Dois jogadores	time ou informação parcial
Ilimitados	PSPACE	PSPACE	EXPTIME	RE
Limitados	P	NP	PSPACE	NEXPTIME

Resumidamente, P é a classe dos problemas computacionais de decisão que podem ser resolvidos em tempo polinomial. NP é a classe dos problemas que podem ter uma solução verificada em tempo polinomial. $PSPACE$ é a classe dos que podem ser resolvidos com uma quantidade polinomial de armazenamento (memória). $EXPTIME$ é a classe dos que podem ser resolvidos em tempo exponencial com expoente polinomial. RE (ou Recursivamente Enumerável) é a classe dos problemas de decisão para os quais existe um algoritmo que, dada uma instância sim, consegue determinar que a instância dada é realmente sim. $NEXPTIME$ é a classe daqueles que podem ter uma solução verificada em tempo exponencial de expoente polinomial.

Finalmente, o livro "*Games, puzzles and Computation*" [Hearn e Demaine, 2009] obtém reduções polinomiais entre variantes desse jogo e vários outros jogos, como Sokoban e Rush-Hour. Com isso, obtém provas alternativas de complexidade computacional (de acordo com a tabela acima, dependendo da variante utilizada na redução) para tais jogos.

Existe uma conferência bienal, FUN (Fun with Algorithms), onde se concentram trabalhos de algoritmos e complexidade computacional voltados a jogos. No Capítulo 2, apresentamos as definições e terminologia que servirão de base para o desenvolvimento deste trabalho. No Capítulo 3, apresentamos nossos resultados sobre problemas inspirados no jogo Bloxorz. No Capítulo 4, apresentamos nossos resultados sobre um problema inspirado no jogo On The Edge. No Capítulo 5, apresentamos nossa formulaçao de problemas inspirados no jogo Bobbin 3D.

2 Conceitos Básicos

2.1 Grafos

Um grafo G é um par ordenado $(V(G), E(G))$ composto de um conjunto, não vazio, $V(G)$ de vértices e um conjunto $E(G)$, disjunto de $V(G)$, de arestas. Cada aresta é um par não ordenado de vértices de G. Se o par (u, v) está em $E(G)$, escreve-se $uv \in E(G)$ e diz-se que u e v são *extremidades* da aresta uv.

A denominação grafo vem do fato do mesmo poder ser representado graficamente, os vértices como pontos e as arestas como linhas ligando suas extremidades. Dizemos que dois vértices são *adjacentes* quando eles são extremidade de uma mesma aresta. De maneira análoga duas arestas são adjacentes se compartilham uma extremidade. Uma aresta que possui extremidades iguais é chamada *laço*. Duas arestas distintas com extremidades iguais são chamadas de *arestas múltiplas*.

Um grafo G é *finito* se $V(G)$ e $E(G)$ são finitos. Um grafo G é dito simples se G é finito, não possui laços e não possui arestas múltiplas. A partir de agora, toda vez que mencionarmos grafo, estaremos nos referindo a um grafo simples.

O *grau* de um vértice v em G é o número de vértices que são adjacentes a v em G e é denotado por $d_G(v)$. Se não houver ambiguidade denota-se apenas por $d(v)$. O *grau máximo* de um grafo G é o maior grau de um vértice de G e é denotado por $\Delta(G)$.

Um *caminho* é uma sequência não vazia $W = v_0e_1v_1e_2v_2 \dots e_kv_k$ na qual os termos são alternadamente vértices e arestas, tais que, para $1 \leq i \neq j \leq k$, $e_i = v_{i-1}v_i$. neste caso v_0 e v_k são as *extremidades de* W. Um caminho com n vértices é denotado por P_n e o *comprimento* de um caminho é o número de arestas que este possui. A *distância entre dois vértices* u e v em um grafo G é o comprimento do menor caminho que possui extremidades u e v em G.

Um *caminho hamiltoniano* $W = v_0e_1v_1e_2v_2 \dots e_kv_k$ de um grafo G é um caminho tal que $\forall v \in V(G)$, existe um único i tal que $v_i = v$.

H é um *subgrafo* de G se G e H são grafos onde $V(H) \subseteq V(G))$ e $E(H) \subseteq E(G)$.

Um subgrafo H de G é *induzido* se, caso $u \in V(H)$ e $v \in V(H)$, $uv \in E(H) \Leftrightarrow uv \in E(G)$.

O *produto cartesiano* $G\square H$ entre os grafos G e H é o grafo $(V_\square, E_\square)$ tal que $V_\square = V(G) \times V(H)$ e $(u, u')(v, v') \in E_\square$ se ocorre um dos seguintes:

- $u = v$ e $u'v' \in E(H)$.
- $u' = v'$ e $uv \in E(G)$.

Um *grid* é o produto cartesiano entre grafos correspondentes a um P_m e um P_n.

Um subgrafo induzido G_s de um grid qualquer é um *grid sólido* se G_s pode ser mapeado no plano cartesiano de maneira que as arestas liguem pontos com distância 1 entre si e as faces internas formadas possuam área 1.

Um *grafo ponderado* é uma generalização do conceito de grafo onde, além do par (V, E), existe uma função $w : E \to \mathbb{R}$. O valor de w para uma aresta é conhecido como seu peso. Em grafos ponderados, o comprimento, nesse caso geral conhecido como custo, de um caminho P_n é $\sum_{i=1}^{n-1} w(e_i)$.

Um *grafo direcionado* é uma alteração do conceito de grafo onde $E(G)$, chamado neste caso de conjunto de arcos, é um conjunto de pares ordenados, ou seja, o arco $(u, v) \neq (v, u)$. Em um grafo direcionado G, o vértice v é adjancente ao vértice u se $(u, v) \in E(G)$ e também se diz nesse caso que e sai de u.

3 Bloxorz

3.1 O problema

Bloxorz é um jogo eletrônico desenvolvido originalmente para a plataforma Flash pela DX Interactive onde o jogador controla um paralelepípedo de dimensões na escala 2:1:1 e deve levá-lo entre 2 pontos de um cenário. Embora a visualização implementada seja uma projeção que dá noção de profundidade, a mecânica envolvida é totalmente bidimensional. A Figura 1 exemplifica o que se pode esperar ver no jogo.

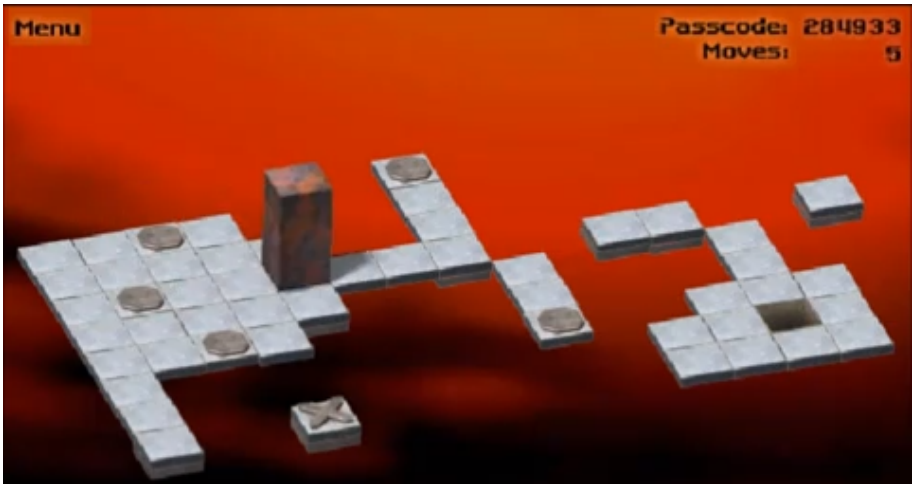
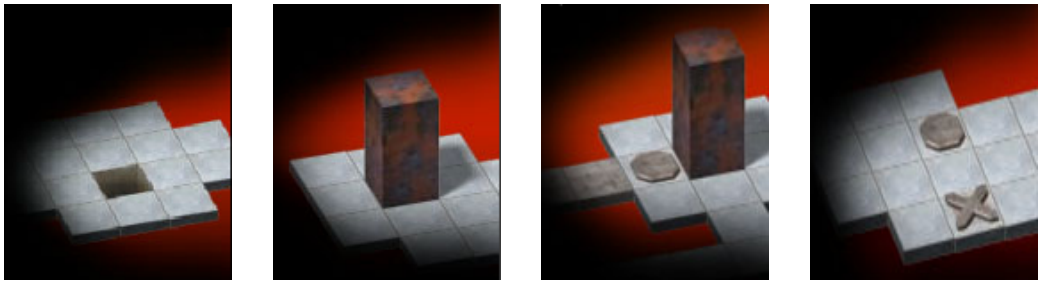
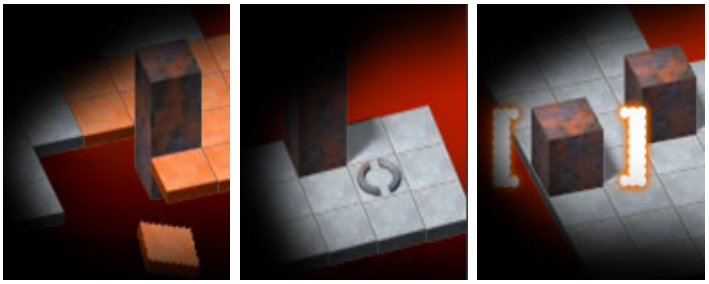


Figura 1 – Uma visão geral do jogo

O objetivo é fazer o bloco, ou seja, o paralelepípedo, cair no buraco, o que é equivalente no caso a ser deixado em pé sobre ele, no final, que é ilustrado na Figura 2a, de cada fase. O bloco, mostrado na Figura 2b, pode ser rolado para frente, para trás, para esquerda ou para a direita, mas não pode sair dos limites do piso da fase. Caso isso aconteça, a fase reinicia. As fases podem ter chaves e pontes como na Figura 2c. As chaves são ativadas quando pressionadas pelo bloco. Pontes então podem mudar de estado e se manter no novo estado, fechadas, por exemplo, mesmo se o bloco for movido. Existem 2 tipos básicos de chaves: *pesadas* ou *leves*. As leves, representadas da Figura 2d pela circular, são ativadas quando qualquer parte do bloco as pressiona. As pesadas, que aparecem em forma de "x"como na Figura 2d, são ativadas apenas se o bloco estiver em pé sobre elas. Quando ativadas, as chaves podem apresentar diferentes comportamentos. Algumas alternarão o estado de pontes entre fechada e aberta a cada uso (que serão chamadas de *alternates*) e outras mudarão o estado de pontes permanentemente (que serão chamadas de *não-alternates*). Ao longo do texto, esses estados das pontes serão também chamados de ligada e desligada. Ligada se referindo a fechada, ou seja, permitindo passar. Piso de madeira, ilustrado na Figura 2e, é mais frágil que o resto do cenário e, se o bloco ficar em



(a) Final da fase (b) Bloco (c) Chave e ponte (d) Dois tipos de chave



(e) Madeira (f) Teletransporte (g) Bloco pequeno

Figura 2 – Elementos de uma fase

pé em cima dele, ele irá cair, tirando o bloco dos limites do piso da fase. Também existem nas fases as chaves especiais de teletransporte como a que aparece na Figura 2f, elas são ativadas como chaves pesadas e teletransportam o bloco para 2 lugares diferentes e assim o dividem em 2 blocos cúbicos menores, um para cada ponto do cenário. Eles podem ser controlados individualmente e se tornam um bloco normal caso colocados um ao lado do outro. Blocos pequenos como os da Figura 2g podem ativar chaves leves, mas não chaves pesadas. Além disso, blocos pequenos não podem passar pelo buraco final da fase.

O problema computacional que é estudado neste capítulo procura traduzir a seguinte pergunta:

Dada uma fase hipotética de Bloxorz, ou seja, uma construção arbitrária que respeite as regras expostas acima sobre o que pode haver em uma fase do jogo, mas sem que se considere possíveis restrições práticas, memória por exemplo, existe uma sequência de jogadas que leve o jogador a passar de fase?

Exemplos de soluções, ou seja, sequências de jogadas, para fases concretas do jogo estão ilustrados nas Figuras 3 e 4.

Na figura 3, o número $i > 1$ marca um bloco que vem em sequida do marcado com $i - 1$ na sequência, 1 marca o primeiro movimento, se i está entre 2 elementos, marca um bloco deitado sobre eles 2 e se está centralizado sobre um elemento marca um bloco em pé sobre ele.

Figura 3 – Curioso caso com pontes

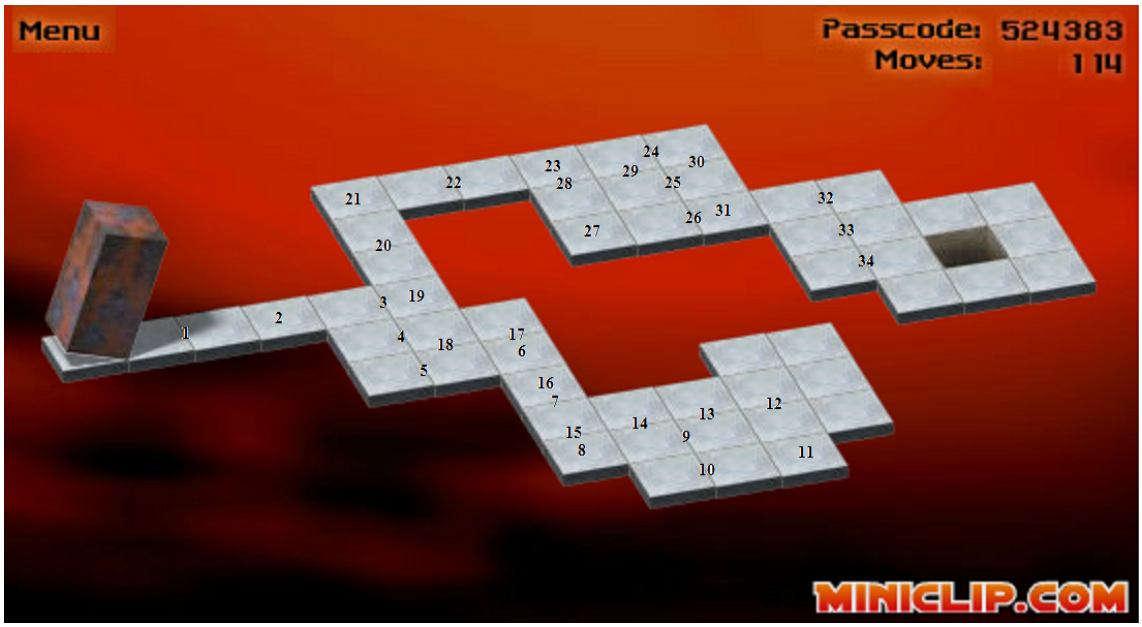


Figura 4 – Passos até o final da fase

3.2 Uma formulação matemática

Definição 3.1. *Sejam a e b números naturais positivos, $A = \{1, ..., a\}$, $B = \{1, ..., b\}$, conjuntos que representam os limites da fase,*

$I = \{(teletransporte, x_1, y_1, x_2, y_2) : x_1, x_2 \in A \text{ e } y_1, y_2 \in B\}$ *e*

$J = \{(chave, ((x_1, y_1), ..., (x_k, y_k)), alt, peso) : x_1, ..., x_k \in A, y_1, ..., y_k \in B, alt, peso \in \{0, 1\}, k \in \mathbb{N}$ *e* $chave \in \{1, ..., |A| * |B|\}\}$, *uma* fase do Bloxorz generalizado *é uma função* $f : A \times B \to T$ *onde* $T = \{\emptyset, piso, madeira, inicial, final\} \cup I \cup J$ *tal que existem* x_i, y_i, x_f *e* y_f *tais que sejam os únicos valores para os quais* $f(x_i, y_i) = inicial$ *e* $f(x_f, y_f) = final$,

não existem (x_{c_1}, y_{c_1}) *e* (x_{c_2}, y_{c_2}) *diferentes tais que* $f(x_{c_1}, y_{c_1}) = (z, K_1, alt_1, peso_1)$ *e* $f(x_{c_2}, y_{c_2}) = (z, K_2, alt_2, peso_2)$. *Além disso, se* $f(x, y) = (z, K, alt, peso), z \in 1, ..., |J \cap Im(f)|$ *e, se* $(x_p, y_p) \in K$, $f(x_p, y_p) = piso$ *ou* $f(x_p, y_p) = \emptyset$. *Essas últimas restrições sobre chaves estabelecem uma numeração única e "sem buracos"para cada uma e que elas operam sobre posições vazias ou de piso.*

Considere o algoritmo abaixo que tem como entrada a matriz f que representa uma fase do Bloxorz generalizado como definido acima, x_i e y_i como definidos acima, o valor d que representa a quantidade de posições (i, j) tais que $f[i][j] \in J$ e o vetor S de elementos de $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ e, como saída, $((x_1, y_1, x_2, y_2), p)$, onde $x_1, x_2 \in A$ e $y_1, y_2 \in B$, um par formado pela situação final do bloco e quantos movimentos foram executados com sucesso. A linguagem em que esse algoritmo está expresso não é bem definida, mas seus passos são precisos para aqueles familiarizados com a semântica de linguagens de programação e pseudocódigo de forma a corresponder a uma determinada máquina de Turing. Uma observação é que a matriz deve representar uma fase válida, ou seja, atendendo todos os requisitos da definição. Por exemplo, uma fase não pode ter dois finais ou ser desprovida de final.

Esse algoritmo é análogo àquilo descrito anteriormente como as regras que definem o resultado de sequências de comandos no Bloxorz e parte da definição de fase dada em seguida.

O problema principal explorado neste capítulo, o $BLOXORZ$, é aquele onde, dada uma fase f do Bloxorz generalizado, deseja-se saber se existe uma sequência S de elementos de $\{\uparrow, \downarrow, \leftarrow, \rightarrow, \_\}$ tal que $|S| \leq |f|$ e, se colocada como parte de uma entrada consistente a com f para o algoritmo descrito, ele retorna $((x_f, y_f, x_f, y_f), |S|)$.

3.3 Linguagem utilizada nas próximas demonstrações

Na Figura 5, são mostrados os símbolos que representarão a estrutura de uma fase ou conjunto de fases da forma como foi definido na seção anterior, cada uma representando uma possibilidade para $f(x, y)$ com a observação de que na verdade pontes abertas significam $f(x, y) = \emptyset$ e pontes fechadas significam $f(x, y) = piso$, o que identifica uma posição como ponte é pertencer a algum ch_2 onde $ch \in J \cap Im(f)$. Na representação, abaixo de todo $tele \in I \cap Im(f)$ estarão indicados $tele_2, tele_3, tele_4$ e $tele_5$ e, abaixo de toda ponte, os rótulos das chaves que satisfazem a condição que a torna uma ponte.

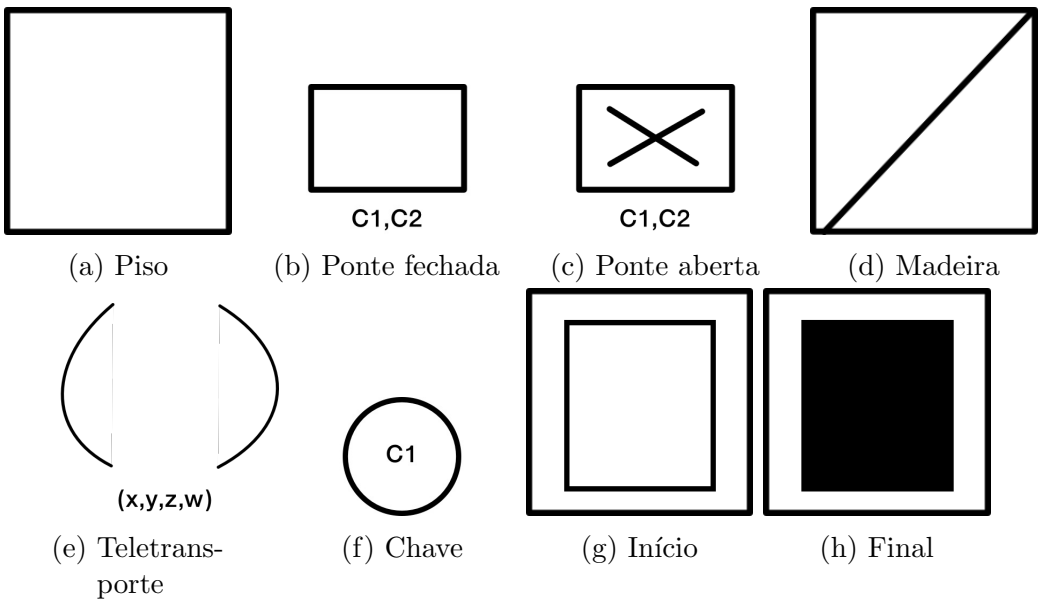
3.4 NP-completude do $BLOXORZ$

Nesta seção, trataremos do problema geral descrito nas seções 3.1 e 3.2. Mais especificamente, trataremos do problema de decisão que chamamos BLOXORZ, que pode ser descrito alternativamente da forma a seguir:

Algoritmo 1 Mecanismos do Bloxorz

Entrada: $f[l][a], x_i, y_i, d, S[s]$
$x_1, x_2, y_1, y_2 := x_i, x_i, y_i, y_i$
$cubo := 0$ $chaves := [0] * d$
for $i := 1, ..., s$ **do**
  $x_3, x_4, y_3, y_4 := x_1, x_2, y_1, y_2$
  $x_1, x_2, y_1, y_2 := rolarbloco(S[i], (x_1, x_2, y_1, y_2), cubo)$
  if $S[i] =$ _ **then**
    if $cubo = 1$ **then**
      $cubo := 2$
    if $cubo = 2$ **then**
      $cubo := 1$
    $i := i + 1$
    vá para a linha 5
  if $f[x_1][y_1] = \emptyset$ *or* $f[x_2][y_2] = \emptyset$ **then**
    $x_1, y_1, x_2, y_2 := x_3, y_3, x_4, y_4$
    return $((x_1, y_1, x_2, y_2), i - 1)$
  if $f[x_1][y_1] = madeira$ *and* $x_1 = x_2$ *e* $y_1 = y_2$ **then**
    O mesmo que o caso anterior
  if $|x_1 - x_2| < 2$ *e* $|y_1 - y_2| < 2$ *e* $(x_1 = x_2$ *ou* $y_1 = y_2)$ **then**
    $cubo := 0$
  if $tamanho(f[x_1][y_1]) = 5$ *e* $x_1 = x_2$ *e* $y_1 = y_2$ **then**
    $cubo := 1$
    $tele := f[x_1][y_1]$
    $x_1, x_2, y_1, y_2 := tele[2], tele[3], tele[4], tele[5]$
  if $tamanho(f[x_1][y_1]) = 4$ *and* $(f[x_1][y_1][3] = 0$ *or* $chaves[f[x_1][y_1][1]] = 0)$
  and $((x_1 = x_2$ *e* $y_1 = y_2)$ *or* $f[x_1][y_1][4] = 0)$ **then**
    $pontes := f[x_1][y_1][2]$
    for $j := 1, ..., tamanho(pontes)$ **do**
      if $pontes[j] = \emptyset$ **then**
        $f[pontes[j]] := piso$
      if $pontes[j] = piso$ **then**
        $f[pontes[j]] := \emptyset$
      $chaves[f[x_1][y_1][1]] := ((chaves[f[x_1][y_1][1]) + 1) \mod 2$
    end
  else
    if $tamanho(f[x_2][y_2]) = 4$ *and* $(f[x_2][y_2][3] = 0$ *or* $chaves[f[x_2][y_2][1]] = 0)$
    e $f[x_2][y_2][4] = 0$ **then**
      O mesmo que o caso anterior trocando x_1, y_1 por x_2, y_2.
  end
end
return $((x_1, y_1, x_2, y_2), s)$

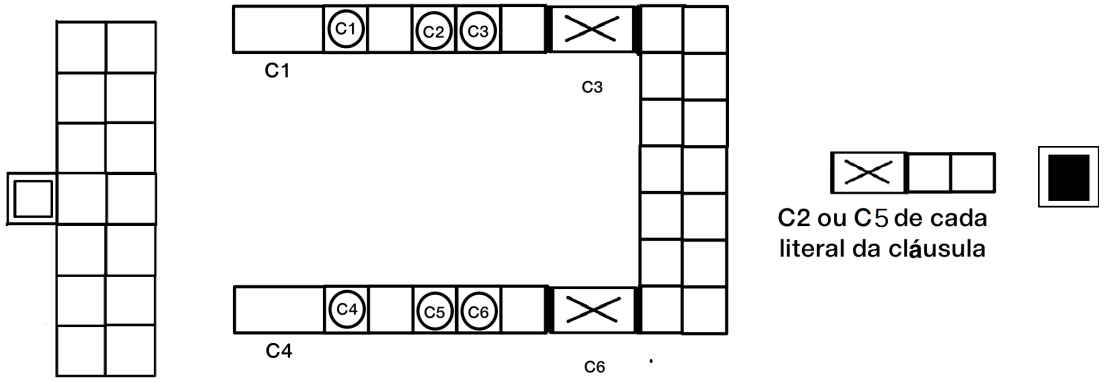
Rotina $rolarbloco(direcao, posicao, cubo)$
$x_1, x_2, y_1, y_2 := posicao[1], posicao[2], posicao[3], posicao[4]$
if $direcao = \rightarrow$ **then**
  if $cubo = 0$ **then**
    if $x_1 = x_2$ **then**
      $x_1 := x_1 + 1$
      if $y_1 = y_2$ **then**
        $x_2 := x_2 + 2$
    else
      $x_2 := x_2 + 1$
    end
    else
      $x_2 := x_2 + 1$
      $x_1 := x_1 + 2$
    end
  if $cubo = 1$ **then**
    $x_1 := x_1 + 1$
  if $cubo = 2$ **then**
    $x_2 := x_2 + 1$
if $direcao = \leftarrow$ **then**
  mesmo que $S[i] = \rightarrow$ trocando '+' por '-'
if $direcao = \uparrow$ **then**
  mesmo que $S[i] = \rightarrow$ trocando 'x' por 'y'
if $direcao = \downarrow$ **then**
  mesmo que $S[i] = \uparrow$ trocando '+' por '-'
return (x_1, x_2, y_1, y_2)



(a) Piso (b) Ponte fechada (c) Ponte aberta (d) Madeira

(e) Teletransporte (f) Chave (g) Início (h) Final

Figura 5 – Elementos de uma fase



(a) Início (b) Variável (c) Cláusula (d) Fim

Figura 6 – Engrenagens da redução

- **Problema de Decisão BLOXORZ:**
- **Entrada:** Uma fase f do Bloxorz generalizado.
- **Pergunta:** Existe uma sequência com no máximo $|f|$ movimentos de modo que o bloco, começando na posição inicial da fase f, chegue na posição final da fase f?

O teorema abaixo prova a NP-Completude do problema de decisão BLOXORZ, mesmo restrito a fases que não possuem teletransportes.

Teorema 3.1. *$SAT \leq_p BLOXORZ$.*

Demonstração. Dada uma fórmula lógica φ na FNC (forma normal conjuntiva), vamos construir uma fase do Bloxorz utilizando as engrenagens (gadgets) descritas abaixo. Essas fórmulas são no formato $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$. Cada C_j é chamado de cláusula e é do formato $l_{1,j} \vee l_{2,j} \vee \ldots \vee l_{m_j,j}$ onde cada $l_{i,j}$ é chamado literal e é uma variável x_i ou o complemento $\overline{x_i}$ de uma variável.

A Figura 6 ilustra a redução. Nela temos que as partes que compõem a fase construída são colocadas uma imediatamente à direita da outra, primeiramente a engrenagem de início, depois uma de variável para cada variável e finalmente, na linha central, uma de cláusula para cada cláusula e o elemento de fim da fase na mesma linha das engrenagens de cláusula. As chaves são leves e não-alternantes. A ponte da engrenagem de uma cláusula C_j é ligada pela chave C2 da engrenagem de uma variável x_i, se $x_i \in C_j$, e é ligada pela chave C5 da engrenagem de uma variável x_i, se $\neg x_i \in C_j$. Observe que as pontes superior e inferior de cada engrenagem de variável começam ligadas e que elas são desligadas respectivamente pelas chaves C1 e C4 da engrenagem da mesma variável. Note ainda que a ponte de cada engrenagem de cláusula começa desligada. Finalmente, observe que, para conseguir chegar a posição final da fase, será preciso ligar todas as pontes das engrenagens de cláusula.

Suponha que a fórmula φ é satisfatível. Ou seja, existe uma atribuição de valores verdadeiro ou falso às variáveis de modo que todas as cláusulas sejam satisfeitas. Para a fase construída para φ, existe uma sequência de movimentos que permite atravessar as engrenagens de variável ativando as chaves correspondentes aos literais satisfeitos pela valoração. Além disso, note que é impossível obter uma sequência de movimentos que ative as chaves C2 e C5 de uma mesma engrenagem de variável. Com isso, como a valoração satisfaz a fórmula φ, então todas as pontes das engrenagens de cláusulas serão ligadas e com isso será possível chegar ao fim da fase. Adicionalmente, como a sequência só tem movimentos para a direita, para cima e para baixo e, se há um movimento para cima, não há para um para baixo na mesma coluna e vice-versa, ela não envolve posições repetidas e portanto a sequência tem cardinalidade menor do que a fase.

Por outro lado, seja $f(\varphi)$ a fase gerada pela redução em questão para a fórmula φ na CNF. Se existe uma sequência-solução para a fase $f(\varphi)$, existe uma sequência de movimentos que liga todas as pontes das engrenagens de cláusula. Tal sequência atravessa as engrenagens de variável ativando a chave de pelo menos um literal de cada cláusula de φ. Além disso, como já dito, não é possível ativar a chave C2 e C5 de uma mesma engrenagem de variável em nenhuma sequência de movimentos sucessivos, pois, se a chave C2 foi ativada, a chave C1 também foi ativada, pois, devido à ponte ativada pela chave C3, não é possível alcançá-la vindo da direita, e, como existe a chave C1, não é possível ativar a chave C2 e depois retornar à esquerda para ativar a chave C5. Um raciocinio simétrico pode ser aplicado partindo de um literal negado. Portanto, φ é satisfatível. □

Lema 3.1. $BLOXORZ \in NP$

Demonstração. $BLOXORZ$ é um problema de decisão e seu algoritmo verificador é avaliar se, dada uma fase do Bloxorz como entrada e dada uma sequência de movimentos como certificado, esta sequência de movimentos leva o bloxo da posição inicial a posição final da fase e se o número de movimentos é menor ou igual ao tamanho da fase. Como o tamanho da sequência está limitado pelo tamanho da fase, isso pode ser feito em tempo polinomial em relação à entrada. □

Colorário: $BLOXORZ$ é NP-completo.

Pode-se observar que o comportamento segue a tabela da Seção **??**, já que temos um quebra-cabeça de um jogador e adicionamos uma restrição sobre o número de passos. Embora não exista aqui uma prova formal, pode-se conjeturar que tirando essa restrição, o problema se encaixa em $PSPACE$.

Para complementar o entendimento dos raciocínios apresentados na demonstração, é mostrado na Figura 7 um exemplo com uma pequena economia de posições, mas perfeitamente compatível com o esquema ilustrado anteriormente. Nesse exemplo, é mostrada uma fase construída a partir da fórmula $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_2)$.

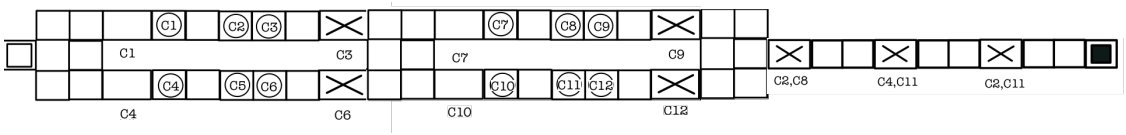


Figura 7 – Exemplo de fase construída a partir de fórmula

3.5 Inaproximabilidade

Nesta seção é considerada uma versão de otimização do problema $BLOXORZ$. Para isso, precisamos receber como entrada uma fase que tenha sempre uma solução (uma sequência de movimentos que leve o bloco da posição inicial até a posição final). Na definição abaixo, usaremos a noção de fase desviável, que é basicamente uma fase generalizada F do Bloxorz que admite uma sequência com no máximo $|F|$ movimentos levando o bloco da posição inicial até a final sem passar por chaves. Para delimitar esse problema, primeiro será feita a seguinte definição:

Definição 3.2. *Uma* fase desviável *de Bloxorz é uma fase f do Bloxorz generalizado tal que existe um conjunto L de posições (x, y) tal que pode-se definir a fase g com um domínio que englobe os valores de L, $g(x, y) = f(x, y)$ se $(x, y) \in L$ e $g(x, y) = \emptyset$ caso contrário, não existe $g(x, y) \in J$ e g é uma instância sim do $BLOXORZ$.*

É fácil ver que, dada uma fase do Bloxorz generalizado, é possível decidir se ela é desviável ou não em tempo polinomial. Uma observação é que uma fase desviável f de Bloxorz é uma instância sim do $BLOXORZ$, pois qualquer sequência-solução de g é sequência solução de f.

A pergunta que define o problema de otimização MINBLOXORZ é: Dada uma fase f desviável de Bloxorz, qual é o menor valor $h = |S|$ onde S é uma sequência de movimentos que levam o bloco da posição inicial até a posição final?

Teorema 3.2. *Se existe um algoritmo $n^{1-\varepsilon}$-aproximativo para $MINBLOXORZ$ para algum $\varepsilon > 0$, então $P = NP$.*

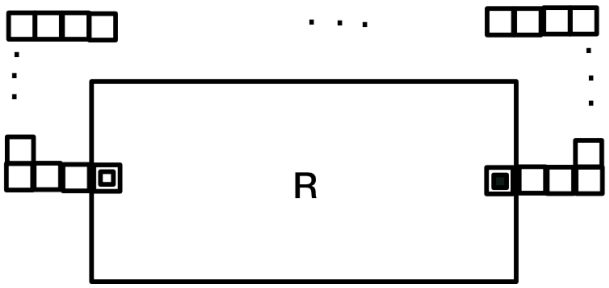


Figura 8 – Construção que indica inaproximabilidade

Demonstração. Vamos aplicar a conhecida técnica do gap em provas de inaproximabilidade. Suponha que existe um algoritmo polinomial aproximativo com fator $n^{1-\varepsilon}$ para algum $\varepsilon > 0$, onde n é o tamanho da fase desviável, para o problema MINBLOXORZ.

Vamos utilizar a redução $SAT \leq_p BLOXORZ$ do Teorema 3.1. Dada uma fórmula φ na CNF, seja F a fase obtida na redução do Teorema 3.1. Provou-se que, se φ é satisfatível, então existe uma sequência com no máximo $|F|$ movimentos que levam o bloco da posição inicial até a posição final da fase F.

Vamos alterar a fase F para que seja uma fase desviável. Acrescente à fase F um "caminho externo" (ilustrado na Figura 8) de tamanho $X = N^{2/\varepsilon} - N$ capaz de levar o bloco da posição inicial até a posição final. Com isso, obtemos uma fase F' desviável de tamanho $|F'| = N + X = N^{2/\varepsilon}$.

Se φ é satisfatível, então a fase F' possui uma sequência de movimentos de tamanho no máximo N. Com isso, o algoritmo aproximativo obtém uma solução de tamanho no máximo $N \cdot fator = N \cdot (|F'|^{1-\varepsilon}) = N \cdot (N^{2/\varepsilon})^{1-\varepsilon} = N^{-1+2/\varepsilon} < N^{2/\varepsilon} - N = X$. Logo o algoritmo aproximativo obtém uma solução de tamanho menor do que X e que consequentemente não utiliza o "caminho externo" (pois este tem tamanho X). Ou seja, o algoritmo obtém um caminho na fase original F.

Se φ não é satisfatível, então o algoritmo aproximativo obtém uma solução de tamanho pelo menos X.

Resumindo, φ é satisfatível se e só se o algoritmo aproximativo retornar uma solução de tamanho menor que X. Como o algoritmo aproximativo é polinomial, ele estaria decidindo o problema SAT e portanto P=NP. □

3.6 Bloxorz com restrições sobre chaves

Nesta seção lidamos com versões alternativas dos problemas anteriores onde se exclui a possibilidade de fases com uma quantidade arbitrária de chaves que abrem e fecham pontes.

Definição 3.3. *k-BLOXORZ é a restrição do BLOXORZ onde consideramos instâncias f cujo conjunto $Im(f)$ possui até k elementos em J.*

Teorema 3.3. *Existe um algoritmo de tempo de execução polinomial que decide k-BLOXORZ.*

Demonstração. Supondo a existência de um algoritmo de tempo de execução $O(a^b\alpha(n))$ para $BLOXORZ$ restrito às devidas instâncias, onde a é uma constante qualquer, b é o número de chaves da fase, $\alpha(n)$ é um polinômio em n e n é o número de elementos da fase, esse é um algoritmo de execução polinomial para k-$BLOXORZ$. Basta então provar que existe tal algoritmo, que será apresentado a seguir.

Esse algorítmo consiste em gerar um grafo direcionado a partir da fase e encontrar um caminho nesse grafo que represente uma sequência-solução. O conjunto de vértices do grafo é o conjunto de estados possíveis das variáveis x_1, y_1, x_2, y_2 no algoritmo que define o $BLOXORZ$ combinadas com as possibilidades para o vetor *chaves* e mais um vértice z. Os arcos de peso 1 ligam dois vértices que podem ocorrer em iterações seguidas no laço principal do algoritmo e os de peso 0 ligam aqueles onde $(x_1, y_1, x_2, y_2) = (x_f, y_f, x_f, y_f)$ ao vértice z.

Encontrar uma solução para a fase que seja menor do que a própria fase é encontrar um caminho de tal tamanho no grafo, pois cada elemento da sequência é processado em uma iteração e cada iteração processa um elemento.

Tal construção é uma abordagem trivial para o problema, pois parte dos estados do algoritmo que o define e busca conexão entre dois estados. O próprio algoritmo definidor do problema define também o procedimento para a construção das arestas neste algorimo.

Sobre a complexidade de tempo, é preciso avaliar o custo de construir os vértices, as arestas e o de encontrar o caminho. Cada vértice ou arco pode ser contruído em tempo $O(1)$, pois cada arco é definido com um simples if do algoritmo. A quantidade de vértices é $2^b n^2$, o número de maneiras válidas de valorar $x_1, y_1, x_2 e y_2$ multiplicado pelo número de maneiras de valorar o vetor *chaves*. O número de arcos é proporcional ao número de vértices, pois o número que sai de cada vértice é limitado por $2|\{\uparrow, \downarrow, \leftarrow, \rightarrow\}| = 8$, logo a geração do grafo tem o mesmo custo de tempo assintoticamente que o número de vértices e que o algoritmo como um todo, já que para detectar o caminho, basta usar um algoritmo de busca. Temos que o algoritmo desta demonstração executa em uma quantidade de passos $O(a^b \alpha(n))$ onde $a = 2$, b é o número de chaves da fase e $\alpha(n) = n^2$, logo esse é um algoritmo $O(n^2)$ para k-$BLOXORZ$.

□

Uma observação é que o algoritmo apresentado no teorema pode ser visto quase como uma força bruta para o $BLOXORZ$, no entanto, ele traz a informação prática de que, se o objetivo é operar sobre um conjunto de fases de Bloxorz onde o número de chaves é muito pequeno, isso pode ser feito em tempo hábil. Muito pequeno pela exponencialidade no pior caso em relação a um crescimento no número de chaves. Fazendo uma estimativa inocente, provavelmente a partir de no máximo 60 chaves o algoritmo já seria totalmente impraticável na tecnologia atual.

Definição 3.4. *$CLOSED$-$BLOXORZ$ é a restrição do $BLOXORZ$ onde as pontes apenas fecham e não há chaves de teletransporte.*

Teorema 3.4. $CLOSED\text{-}BLOXORZ \in P$

Demonstração. Existe um algoritmo que executa em tempo polinomial em relação ao número de elementos para esse problema. A seguir ele será explicado e pode ser visto

como uma versão do algoritmo para o k-$BLOXORZ$ onde não é necessário considerar todas as combinações de ativação das chaves. Aqui é suposto sem perda de generalidade que a fase não possui pontes inicialmente fechadas, pois elas, se existissem, nunca seriam abertas, o que as tornaria equivalentes a piso normal.

Ele consiste nos seguinte passos:

1. Gere um grafo como no algorimo para o $k - BLOXORZ$ a partir da fase obtida da original com a troca das chaves por piso normal e das pontes por posições vazias.
2. Execute uma busca em profundidade no grafo a partir do vértice do bloco inicial para determinar se existe caminho até o buraco final
3. Em caso positivo, retorne sim.
4. Armazene a informação sobre as chaves para as quais a busca alcança um bloco que as ativaria em um vetor A.
5. Se $A = \emptyset$, retorne não.
6. Execute recursivamente este algoritmo para a fase obtida da original com a troca das pontes que as chaves em A fechariam e das próprias chaves de A por piso normal.

Para atestar a completude/corretude desse algoritmo, pode-se considerar a do apresentado para o k-$BLOXORZ$ e mapear as chamadas recursivas para as mudanças de conjunto de vértices no referido algoritmo anterior com a observação de que, um bloco alcançável a partir de uma sequência de movimentos em uma fase, pode ser alcançado na de sua chamada recursiva replicando-se a sequência, já que as mudanças na fase adicionam piso normal e apenas substituem por piso normal chaves que não possuiriam efeito em sua ativação. Além disso deve-se considerar a condição que o algoritmo para o $CLOSED$-$BLOXORZ$ utiliza para identificar uma instância não. Os casos onde não há caminho até o buraco final desprezando-se a ativação das chaves e ao mesmo tempo não há chaves ativáveis de forma a fechar novas pontes são precisamente as instâncias não.

Para atestar a polinomialidade do algoritmo, basta um cálculo de sua complexidade de tempo. O número de chamadas recursivas é limitado pelo número de chaves, que por sua vez é limitado pelo número total de elementos da fase, que aqui é chamado de n e é o tamanho da entrada. Dentro de cada chamada recursiva, os únicos passos que não podem ser feitos de forma que executem em tempo $O(1)$ são 1 e 2. Os passos 1 e 2 podem ser executados em tempo total $O(n)$, pois na fase modificada não há chaves ou teletransporte, ou seja, há apenas uma possibilidade para o vetor e nenhum caminho entre o vértice inicial e aqueles onde ($x_1 \neq x_2$ e $y_1 \neq y_2$) ou $2 \leq |x_1 - x_2|$ ou $2 \leq |y_1 - y_2|$, esse fato será explicado um pouco melhor na apresentação dos últimos algoritmos do capítulo. Isso leva a uma complexidade total $O(n)$ para cada chamada recursiva e $O(n^2)$ para o algoritmo completo.

□

Definição 3.5. *MIN-CLOSED-BLOXORZ é o problema com a mesma pergunta do MIN-BLOXORZ, mas aplicada às instâncias sim do CLOSED − BLOXORZ. Observe que uma instância sim do CLOSED-BLOXORZ não é necessariamente uma fase desviável.*

Teorema 3.5. *MIN-CLOSED-BLOXORZ é NP-difícil.*

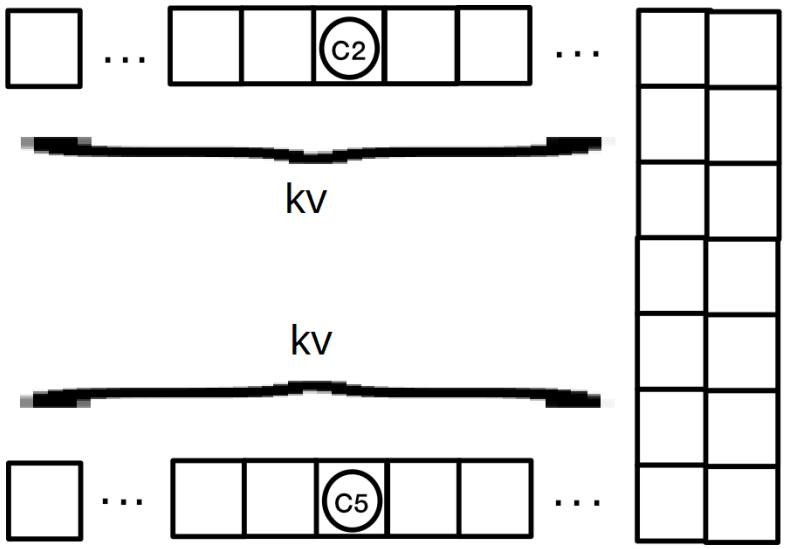


Figura 9 – Engrenagem de variável para o *MIN-CLOSED-BLOXORZ*

Demonstração. Suponha que existe um algoritmo de execução polinomial para o *MIN-CLOSED-BLOXORZ*. Aqui será demonstrado que tal algoritmo decide *SAT*. Dada uma formula φ da lógica proposicional escrita na CNF, pode-se construir uma fase F de Bloxorz como na redução do Teorema 3.1 que prova a NP-completude do *BLOXORZ*, mas com as engrenagens de variável como na Figura 9. A largura de cada engrenagem de variável tem tamanho kv (ilustrado na Figura 9), onde $k = 30$ e v é a quantidade de variáveis em φ Na engrenagem ilustrada, as chaves estão relacionadas às engrenagens de cláusula de forma análoga ao que foi apresentado na construção que provou a NP-completude do *BLOXORZ*.

Note que a fase F construída dessa maneira é uma instância sim do CLOSED-BLOXORZ, visto que o bloco pode ativar todas as chaves de cada engrenagem de variável e com isso ligar todas as pontes relativas às cláusulas mesmo que φ não seja satisfatível.

Note que, como cada engrenagem de variável tem largura $30v$ e que, a cada 2 movimentos na parte superior ou inferior, o bloco percorre 3 posições, então temos $20v$ movimentos (na parte superior ou na parte inferior da engrenagem). Com isso, seja m a quantidade de cláusulas na formula, o número de movimentos é maior que $20v^2 + 2m + 1$, pois cada engrenagem de cláusula precisa de 2 movimentos do bloco para percorrer suas 3 posições e há ainda o último movimento para a posição final.

Observe que, se φ é satisfatível, então existe uma sequência-solução onde, para cada engrenagem de variável, o bloco percorre só a parte de cima ou só a parte de baixo da engrenagem. Além disso, o número de movimentos é menor que $(20v + 7) \cdot v \ + \ 2m + 1$, pois a sequência-solução poderia, no final de cada engrenagem de variável, percorrer o trecho que leva da parte superior até a inferior (ou vice-versa) para iniciar o percurso da próxima engrenagem de variável.

Se φ não é satisfatível, então, para poder chegar até a posição final, o bloco terá que ativar as duas chaves de uma mesma engrenagem de variável. Com isso, seu percurso teria um acréscimo de pelo menos metade de $20v$ movimentos do bloco (ou seja, $10v$ movimentos). Logo o bloco deve percorrer pelo menos $20v^2 + 2m + 1 + 10v \ > \ (20v + 7) \cdot v \ + \ 2m + 1$.

Resumindo, φ é satisfatível se e só existe uma solução de tamanho menor ou igual a $(20v + 7) \cdot v \ + \ 2m + 1$. Como existe um algoritmo polinomial para MIN-CLOSED-BLOXORZ, então P=NP. □

Definição 3.6. *$CLOSED\text{-}BLOXORZ^+$ é a restrição do $BLOXORZ$ onde não há chaves que abram pontes.*

Teorema 3.6. *$CLOSED\text{-}BLOXORZ^+$ é NP-completo.*

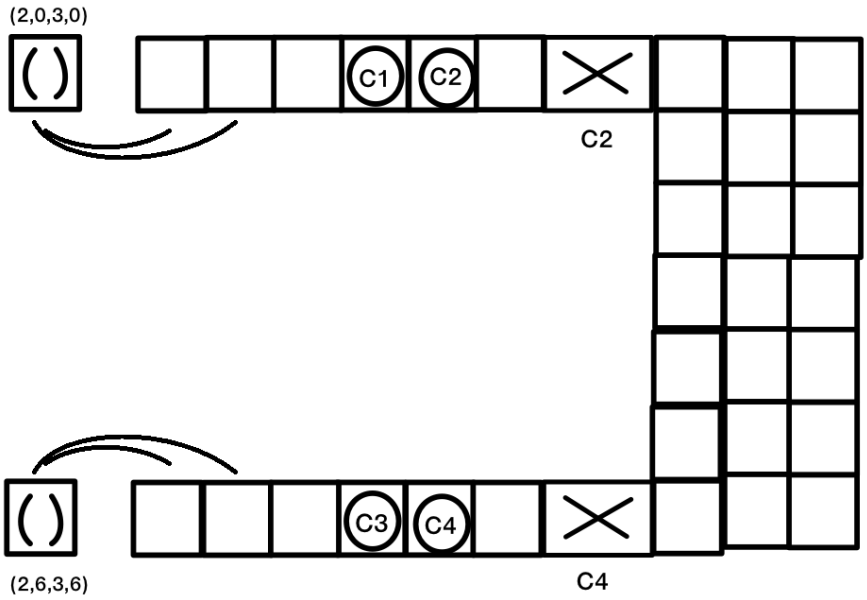


Figura 10 – Engrenagem de variável para o $CLOSED\text{-}BLOXORZ^+$

Demonstração. Analogamente à redução que provou a NP-completude do BLOXORZ, a partir de qualquer fórmula da lógica proposicional na CNF, pode-se construir uma fase do Bloxorz generalizado onde a engrenagem de variável é como na Figura 10. A demonstração de completude/corretude da redução segue análoga à citada observando-se que a sequência-solução construída a partir da valoração será muito semelhante. No início de

cada engrenagem de variável, existem dois teletransportes que podem ser alcançados pelo bloco. Se o bloco utiliza o teletransporte da parte superior, ele irá então ter acesso apenas a parte superior da engrenagem dessa variável. Se o bloco utiliza o teletransporte da parte inferior, ele irá então ter acesso apenas a parte inferior da engrenagem dessa variável. Os números indicados em cada teletransporte são as duas coordenadas para onde as duas metades do bloco são levadas. Dentro de uma engrenagem de variável, um teletransporte será ativado, o que leva x_1 ao valor referido como 2 na imagem e y_1 a 0 ou 6, esses valores são coordenadas relativas que fazem sentido dentro da engrenagem, mas que podem assumir diversos valores efetivos dentro da fase dependendo da posição da engrenagem, o $(0, 0)$ dela. Além disso, a variável *cubo* assumirá o valor 1, gerando o caso especial onde o bloco está em um estado onde poderia mover-se segundo a mecânica de quando $cubo = 0$, mas o algoritmo não levará a isso na próxima iteração. O movimento seguinte sendo para a direita, o bloco mudará para um em pé em (3,0) ou $(3, 6)$. Mais um movimento para a direita e a chave C1 ou C3 será ativada, fechando a ponte das engrenagens de cláusula devidas, mais um e a chave C2 ou C4 será ativada, fechando a ponte alterada por essa chave, o que permite seguir até a próxima engrenagem. Finalmente, deve-se observar que o mesmo mecanismo que impede movimentos de retorno à esquerda de forma a ativar C1 e C3 no mesmo literal, existem nesta redução pois o teletransporte impede o retorno de forma análoga à ponte que correspondia a ele na redução original. □

Considerando o problema onde, a partir do $CLOSED\text{-}BLOXORZ^{+}$, restringimos as instâncias aos casos onde há um número limitado de teletransportes, pode-se conjeturar fortemente que tal problema pertence a P, pois é possível resolvê-lo analizando todas as escolhas possíveis de teletransportes entre o conjunto dos existentes da fase e, para cada um deles, recorrer à estratégia do algoritmo para o $CLOSED\text{-}BLOXORZ$.

3.7 O $BLOXORZ^{--}$

Definição 3.7. *O $BLOXORZ^{--}$ é o BLOXORZ restrito às instâncias sem chaves ou teletransporte.*

Esse na verdade foi o primeiro problema resolvido na produção deste trabalho e o algoritmo que concebemos para resolvê-lo é uma restrição daquele mostrado para o $k - BLOXORZ$.A consideração especial a ser feita é sobre o número de vértices do grafo.

Dada uma fase f do Bloxorz generalizado com as restrições descritas, o grafo G_f a ser contruído no algoritmo será bem mais simples e visualizável do que no $k - BLOXORZ$ de forma geral, pois só há um estado de chaves e não há teletransporte, logo o conjunto $V(G)$ terá tamanho proporcional ao da fase. Vértices onde (x_1, y_1) é distante de (x_2, y_2) poderiam ser criados, mas não haveria caminho do vértice inicial até ele. Adicionalmente, para cada elemento de madeira, um vértice pode ser eliminado, aquele onde $(x_1, y_1) =$

$(x_2, y_2) = (x_m, y_m)$ onde (x_m, y_m) é a posição desse elemento, pois o algoritmo verificador retorna imediatamente caso o bloco esteja em pé sobre tal posição e portanto os caminhos que correspondem a uma solução não incluem tal vértice. Finalmente pode-se observar que neste problema a direcionalidade dos arcos pode ser descartada, já que todos os movimentos que não envolvem ativação de chaves ou teletransporte são reversíveis com outro no sentido oposto.

Teorema 3.7. *Seja G_f o grafo gerado a partir da fase do Bloxorz generalizado f pelo algoritmo em questão, $|V(G_f)| \leq 3n$ onde $n = |f|$.*

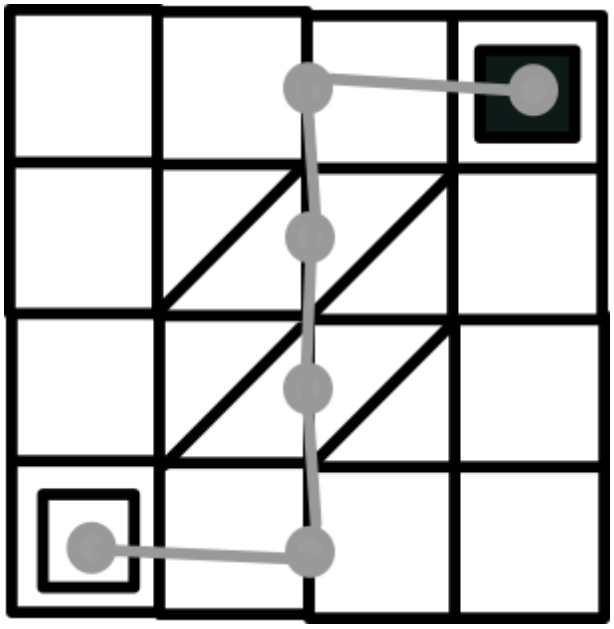


Figura 11 – Caminho-solução para $BLOXORZ^{--}$

Demonstração. G_F possui um vértice para cada elemento de Bloxorz que não seja madeira presente em F. Como F possui n elementos, seja V_e o conjunto de vértices desse tipo, $|V_e| \leq n$. O restante dos vértices de G_F corresponde a pares de elementos vizinhos. Cada elemento possui no máximo 4 vizinhos, 2 na horizontal e 2 na vertical. Se percorrermos os elementos a partir do canto inferior esquerdo, encontraremos como o inicial, pois se trata de um elemento o mais abaixo e o mais à esquerda possível, um elemento com no máximo 2 vizinhos. Então temos no máximo 2 pares de vizinhos envolvendo o elemento inicial e portanto no máximo 2 vértices correspondentes. Seguindo à direita na linha, adicionamos no máximo mais 2 vértices para cada elemento, um por seu vizinho da direita e outro por seu vizinho de cima, pois se trata da linha mais abaixo e o vizinho da esquerda forma um par já considerado quando ele foi percorrido. Ao seguirmos para a linha superior, os vértices que seriam adicionados pelo vizinho de baixo de cada elemento não o são porque já o foram quando esses vizinhos foram percorridos. Assim, indutivamente, seja V_p o conjunto de vértices desse tipo, $|V_p| \leq 2n$. Como $V(G_F) = V_e \cup V_p$, $|V(G_F)| \leq |V_e| + |V_p| \leq n + 2n = 3n$.

□

Teorema 3.8. *Sejam f uma instância do $BLOXORZ^{--}$ e G_f o grafo gerado pelo algoritmo em questão para f, existe um caminho em G_f entre os vértices v_i e v_f, correspondentes respectivamente ao bloco inicial de I e ao final da fase, se e somente se a resposta de I é sim.*

Demonstração. ($\leftarrow$)

Se f é uma instância com resposta sim, possui uma sequência-solução. Dada uma sequência-solução de Bloxorz para f, há uma sequência de blocos B correspondente que, por ser extraída de uma sequência-solução, se inicia com o bloco inicial de f, é tal que 2 blocos seguidos nela são tais que um é resultado de um movimento válido sobre o outro, termina com um bloco em pé no final da fase de f e não possui nenhum bloco em pé ocupando piso de madeira. Pelas propriedades de B e pela construção de G_f, todo par de blocos seguidos de B tem os vértices correspondentes adjacentes. Indutivamente, todo vértice com correspondente em B é alcançado por arestas a partir de v_i e portanto v_f é alcançado. Percorrendo B eliminando o que há entre as ocorrências de um mesmo bloco e deixando apenas uma, constrói-se um caminho entre v_i e v_f.

($\rightarrow$)

Se existe um caminho entre v_i e v_f em G_f, pela definição de caminho existe uma sequência que alterna vértices e arestas sem repetição, começa com v_i, termina com v_f e a aresta (u, v) é a única que pode aparecer imediatamente entre 2 vértices u e v quaisquer nela. Cada vértice corresponde a um bloco e, pela definição de G_f, se $uv \in E(G_f)$, u e v correspondem a blocos tais que um resulta de um movimento válido sobre o outro. Logo, indutivamente, o buraco final de I é alcançado por uma sequência de movimentos válidos a partir do bloco correspondente a v_i sem que exista nessa sequência um bloco em pé sobre madeira, pois não existe nenhum vértice correspondente a esses blocos, logo f é uma instância com resposta sim.

□

Combinando os dois teoremas, conclui-se que, para decidir o $BLOXORZ^{--}$ em tempo polinomial em relação ao tamanho da instância, basta decidir se existe um caminho entre 2 vértices de um grafo qualquer em tempo polinomial em relação a seu número de vértices. Para esse fim, um algoritmo de busca em largura ou profundiade é o suficiente. Logo a argumentação está completa.

Sendo mais preciso em relação à complexidade de tempo, é possível construir o grafo em tempo $O(n)$ no pior caso, onde n é o número de elementos da fase do Bloxorz generalizado, pois são no máximo $3n$ vértices e $6n$ arestas por um raciocínio análogo ao do primeiro teorema e é possível contruir cada aresta em tempo constante bastando uma checagem das posições que o bloco gerado pelo movimento ocuparia. Um algoritmo de busca executa em tempo $O(|V| + |E|)$, o que no caso significa $O(n)$. Já o algoritmo para encontrar o menor

caminho como em [**?**] executa em tempo $O(|E| + |V|log(|V|))$, ou seja, $O(n + nlog(n)) = O(nlog(n))$, se usadas as devidas estruturas de dados e portanto nosso algoritmo para o $BLOXORZ^{--}$ executa em tempo $O(nlog(n))$ com a troca da busca pelo algoritmo de menor caminho.

Uma observaçao importante é que esse algoritmo, além de resolver o problema, encontra, com a substituição mencionada, se houver, a menor sequência-solução para a fase do Bloxorz generalizado dada como entrada, o que resolve uma versão de otimização do problema.

3.8 O $BLOXORZ^{-}$

Nesta seção, o foco é o caso onde não há pontes, mas pode existir teletransporte além de todos os elementos do $BLOXORZ^{--}$, chamaremos esse novo problema de $BLOXORX^{-}$. Para resolvê-lo, basta recorrer ao algoritmo apresentado para o k-$BLOXORZ$ considerando apenas x_1, y_1, x_2, y_2, pois não há chaves.

O vértice final a ser considerado é aquele onde $(x_1, y_1) = (x_2, y_2) = (x_f, y_f)$, então não existirão os arcos de peso 0.

.

4 On The Edge

4.1 O problema

O tópico deste capítulo é um novo problema extraído de um jogo semelhante ao Bloxorz, na verdade inspirado nele, mas com diferenças importantes. On The Edge é um jogo eletrônico desenvolvido para a plataforma Flash pela Yzi Games onde o jogador controla um cubo. A Figura 12 exemplifica o que pode-se esperar ver no jogo:

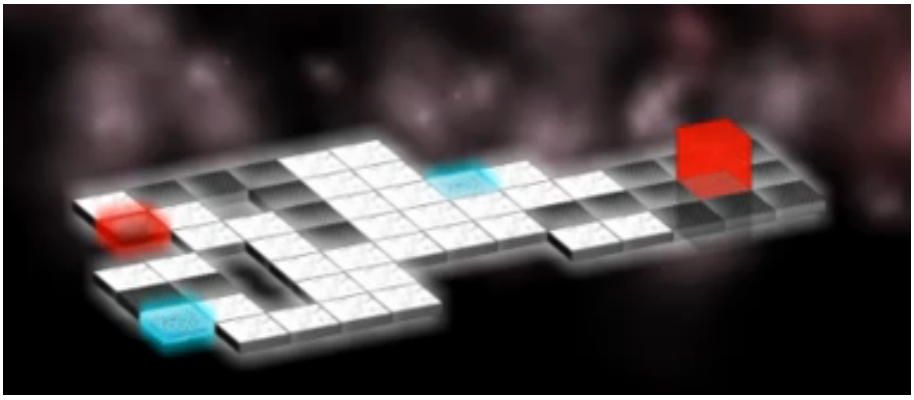


Figura 12 – Uma visão geral do jogo

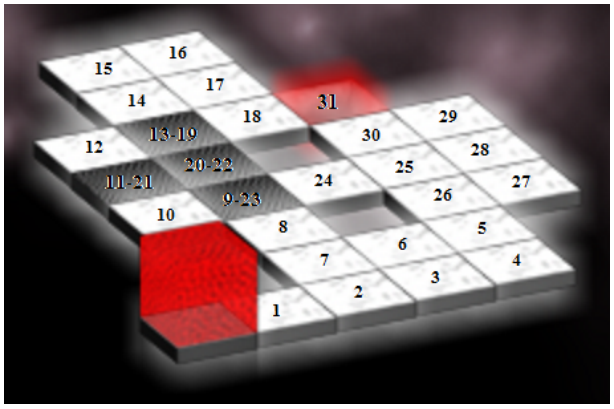
O objetivo é fazer o bloco, ou seja, o cubo, alcançar a posição vermelha no final de cada fase após eliminados todos os elementos restantes. O bloco pode ser rolado para frente, para trás, para esquerda ou para a direita, mas não pode sair dos limites do piso da fase. Caso isso aconteça, a fase reinicia.

Caso o bloco esteja sobre piso normal, ao ser rolado para outra posição, o piso que ele ocupava é eliminado. Em uma fase de On The Edge pode haver posições pretas, que não sofrem esse efeito, mas tornam-se piso normal, e 0 ou 2 posições azuis, que levam o bloco à outra posição azul e são eliminadas após esse efeito, a que foi ativada o é imediatamente e a posição alvo após ser abandonada.

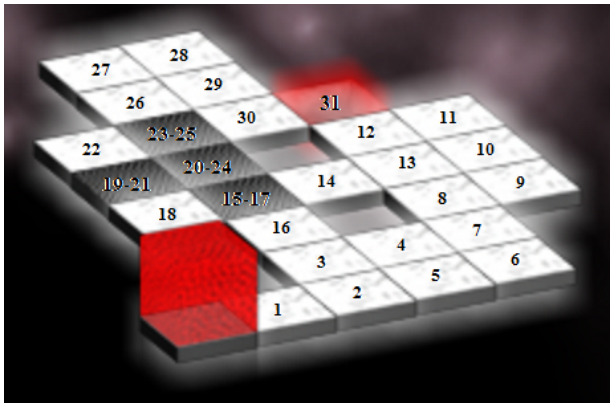
O problema computacional que é estudado neste capítulo, que chamaremos de *ON-THE-EDGE*, corresponde à seguinte pergunta:

Dada uma fase hipotética de On The Edge, ou seja, uma construção arbitrária que respeite as regras expostas acima sobre o que pode haver em uma fase do jogo, mas sem que se considere possíveis restrições práticas, existe uma sequência de jogadas que leve o jogador a passar de fase?

Um exemblo de 2 soluções possíveis para uma mesma instância de *ON-THE-EDGE*, ou seja, ambas significariam uma resposta sim, está ilustrado na Figura 13. Trata-se de uma fase concreta do jogo implementado.



(a) Possibilidade 1



(b) Possibilidade 2

Figura 13 – Mesma fase, dois caminhos

4.2 NP-completude

Nosso resultado sobre o $ON\text{-}THE\text{-}EDGE$ se refere a sua relação com um problema em grafos. Especificamente com o problema de decidir se um subgrafo induzido de um grid possui um caminho hamiltoniano entre os vértices u e v. Chamaremos esse problema aqui de $HP^{SUBGRID}$. Sobre o $HP^{SUBGRID}$, em 1982, A. Itai, C. H. Papadimitriou e J. L. Szwarcfiter demonstraram sua NP-completude. [Itai et al., 1982]

Teorema 4.1. $HP^{SUBGRID} \leq_p ON\text{-}THE\text{-}EDGE$.

Demonstração. Dado um subgrafo H induzido de um grid, pode-se construir uma fase de nosso On The Edge generalizado como na Figura 14. Esse é apenas um exemplo, mas captura o mecanismo geral necessário para a construção.

Na Figura 14, em azul escuro e verde está representado o grafo H. Cada vértice corresponde a uma posição de piso e a vizinhança no grafo corresponde à vizinhança na fase. Os vértices em verde são aqueles entre os quais se deseja verificar a presença de um caminho hamiltoniano. O vértice final corresponderá a uma posição azul. Em cinza está representado o piso normal da fase, em azul claro, as posições azuis e em vermelho o final da fase. A posição inicial corresponde ao retângulo cinza onde está um dos vértices em

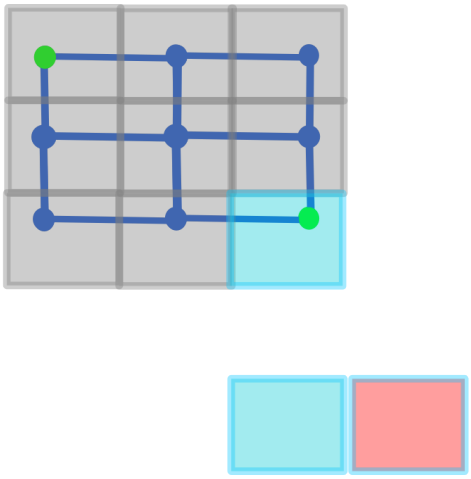


Figura 14 – Redução do problema do caminho hamiltoniano em subgrafos induzidos de grids

verde. Na fase gerada, sempre existe a região correspondente aos vértices do grafo e a região separada, ou seja, inalcançável a partir da primeira sem o efeito de uma posição azul, com uma posição azul e imediatamente à direita o final.

Se existe um caminho hamiltoniano entre u e v em H, existe uma sequência de movimentos que passa por cada posição relacionada a piso normal na fase construída uma única vez e alcança a posição azul da região onde elas se encontram, pois cada movimento válido nessa região corresponde a uma aresta de H, u corresponde à posição inicial e v à posição azul. Além disso, ao alcançar a posição azul, pode-se continuar os movimentos a partir da outra, agora com a primeira eliminada. Basta um movimento a mais para a direita e o final é alcançado com todos os outros elementos da fase eliminados.

Por outro lado, se existe uma sequência de movimentos válidos que alcança o final da fase com todos os outros elementos eliminados, existe uma que alcança a posição azul logo à esquerda dele já eliminados todos os elementos diferentes desses 2, pois não existem outros elementos vizinhos ao final, logo há uma terceira sequência que alcança a outra posiçao azul e elimina todo o piso normal já que elas se encontram em regiões separadas entre si. Essa sequência por suas propriadades passa por cada posição correspondente a piso normal uma única vez. Como essas posições correspondem também a vértices do grafo original e os movimentos válidos a arestas, a sequência pode ser mapeada para um caminho hamiltoniano em H.

□

5 Bobbin 3D

5.1 O problema

Bobbin 3D é um jogo semelhante ao Bloxorz desenvolvido pela Maplabs para o sistema Android. Nele o bloco a ser levado entre 2 pontos do cenário é um cilindro de altura 1 e raio $\frac{1}{2}$. Apesar de trazer uma impressão ainda maior que os anteriores de tridimensionalidade, a mecânica continua sendo bidimensional. A Figura 15 exemplifica o que pode-se esperar ver no jogo:

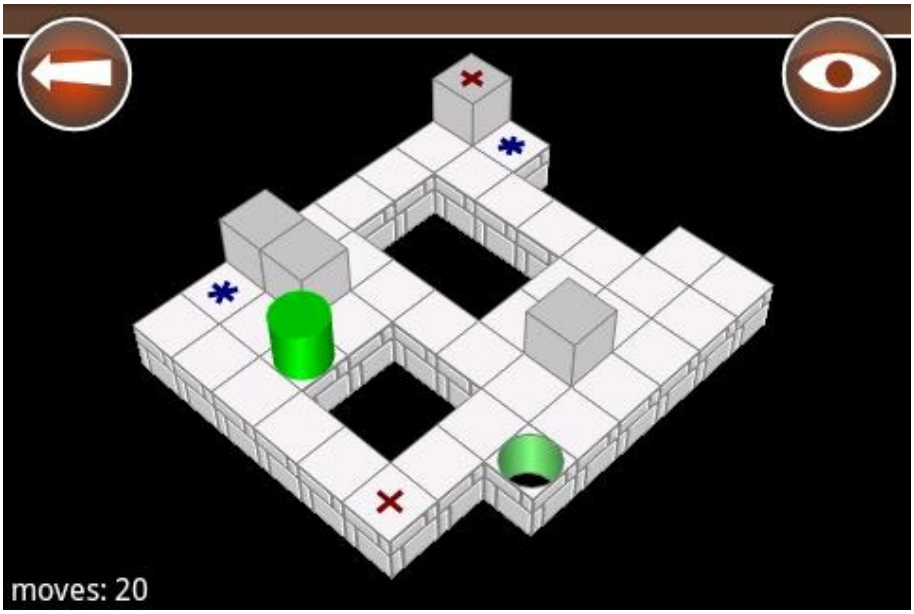


Figura 15 – Uma visão geral do jogo

O objetivo é rolar o cilindro a partir da posição inicial até que ele fique em pé sobre o buraco final. O bloco pode ser rolado para frente, para trás, para esquerda ou para a direita. Com a justificativa geométrica de o bloco ser um cilindro, rolar não significa sempre um movimento restrito a uma vizinhança limitada pelas dimensões do objeto. Se o bloco está deitado e é rolado de forma a se manter deitado, o bloco resultante ocupa somente posição anterior ao primeiro obstáculo presente na linha ou coluna.

Em uma fase de Bobbin 3D, pode haver piso normal e piso elevado, que serve de obstáculo quando o bloco rola. Também pode haver uma quantidade par de chaves '+' elevadas ou abaixadas, que, quando elevadas, funcionam como o piso elevado e quando abaixadas alternam o estado de outra chave '+' pré-definida entre elevada e abaixada. Se uma chave '+' alterna o estado de outra, essa outra alterna o estado dela. Além disso, podem existir posições '*' que transformam o bloco cilíndrico em cúbico por uma quantidade determinada de movimentos. Um bloco cúbico ocupa uma única posição e

pode ser rolado nas mesmas 4 direções, mas sempre no máximo até uma posição vizinha.

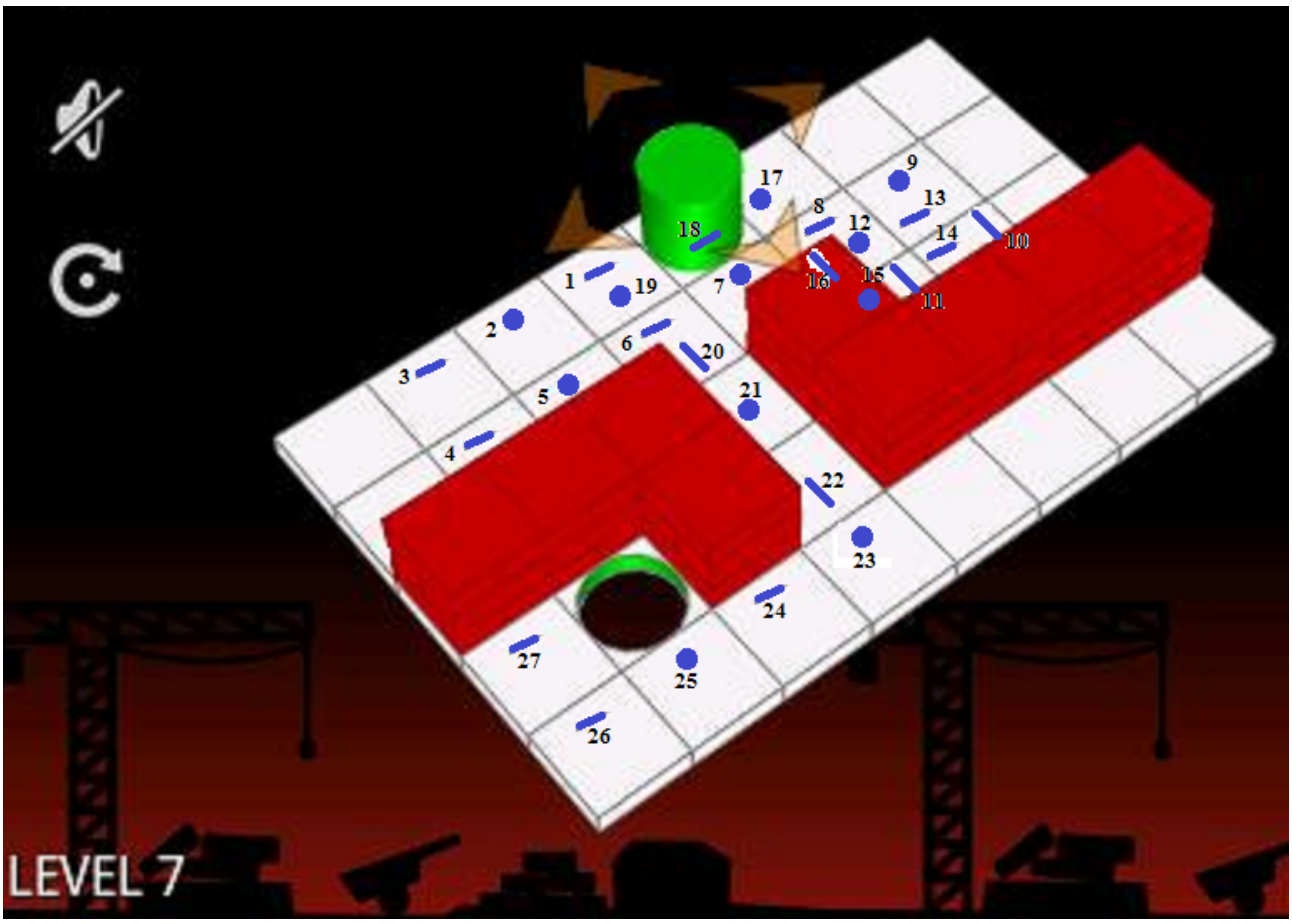


Figura 16 – Passos em fase do Bobbin 3D

O problema computacional que é estudado neste capítulo, que chamaremos de $BOBBIN$, corresponde à seguinte pergunta:

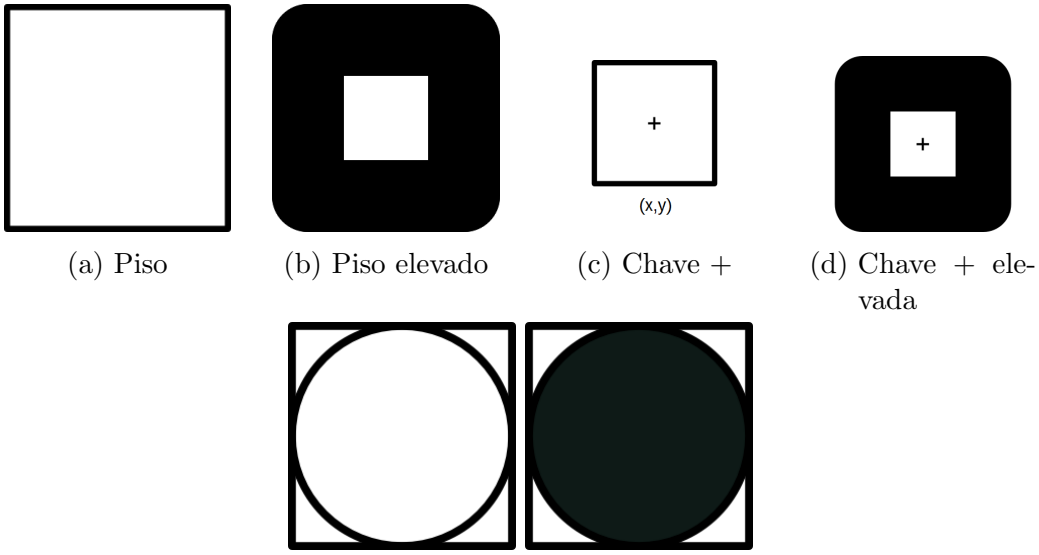
Dada uma fase hipotética de Bobbin 3D, ou seja, uma construção arbitrária que respeite as regras expostas acima sobre o que pode haver em uma fase do jogo, mas sem que se considere possíveis restrições práticas, existe uma sequência de jogadas menor do que a fase que leve o jogador a passar de fase?

A Figura 16 ilustra um exemplo de solução para uma fase concreta do jogo.

Uma observação é que pode-se ser construído para esse jogo um algorimo semelhante ao do capítulo anterior com o objetivo de formalizar as regras dos movimentos e consequentemente delimitar melhor o problema.

Definição 5.1. *$BOBBIN^{--}$ é a restrição do $BOBBIN$ de forma que não haja chaves '+' ou posições '*'*

Definição 5.2. *$BOBBIN^{-}$ é a restrição do $BOBBIN$ de forma que não haja chaves '+'.*



(a) Piso (b) Piso elevado (c) Chave + (d) Chave + elevada

(e) Início (f) Final

Figura 17 – Elementos de uma fase

5.2 Linguagem a ser utilizada a seguir

Esta seção se refere ao formato visual que descreverá as construções apresentadas. Na Figura 17 temos uma representação para cada tipo de elemento de forma similar ao que foi feito para o Bloxorz.

Cada chave '+' em estado normal tem indicada a posição de seu par, as elevadas não precisam, pois pode-se supor sem perda de generalidade que são pareadas cada uma com uma em estado normal que a indica. Se uma fase possui um par de chaves '+' inicialmente elevadas, pode-se trocar por um par de elementos de piso elevado.

5.3 Algoritmo para o $BOBBIN^{--}$

Para as instâncias deste problema, pode-se aplicar a mesma estratégia de nosso algoritmo para o $BLOXORZ^{--}$, mas considerando um grafo direcionado como ilustrado na Figura 18.

Diferentemente do caso do $BLOXORZ^{--}$, cada posição possui 3 vértices diferentes correspondentes. Pode-se excluir aquelas de piso elevado nesse sentido. No algoritmo definidor, isso significaria uma variável a mais marcando o estado do bloco entre deitado horizontalmente, deitado verticalmente ou em pé, cada estado sendo representado no algoritmo que soluciona o problema por um vértice.

Cada arco representa um movimento válido possível na fase hipotética de Bobbin 3D, incluindo aqueles onde o bloco "rola", já que haverá um arco ligando o bloco deitado a outro também deitado, mas na posição anterior no devido eixo ao elemento elevado mais próximo. Além disso, neste algoritmo, a posição final é representada por um vértice e o

caminho mínimo a ser buscado é do bloco inicial, aqui o conjunto de vértices do grafo é um conjunto de blocos, a esse vértice, que é o bloco em pé sobre a posição final.

Com essas observações, o argumento de corretude/completude é o mesmo do algoritmo que decide o $BLOXORZ^{--}$.

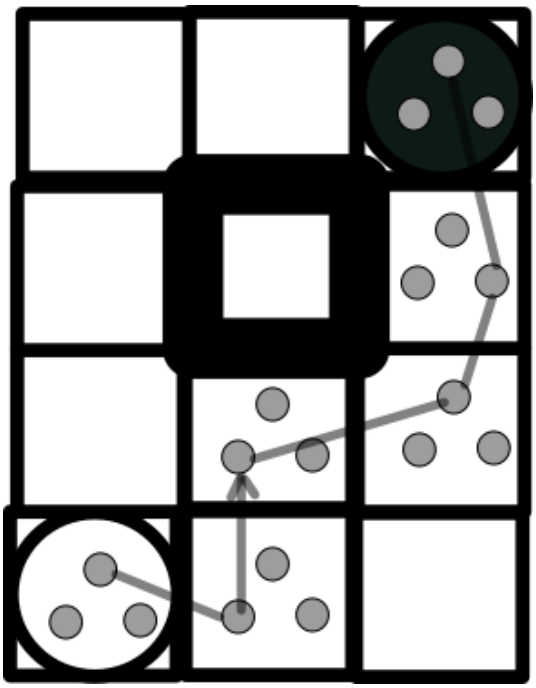
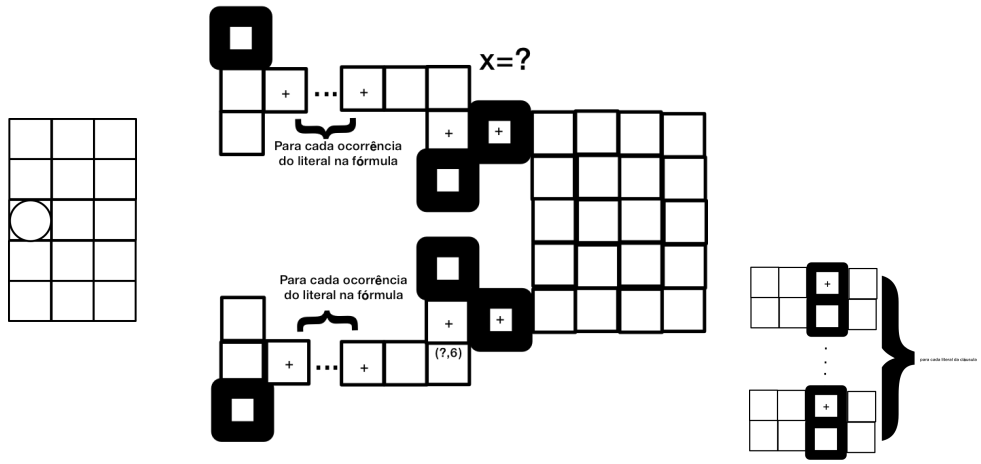


Figura 18 – Resolvendo um caso simples

5.4 Algoritmo para o $BOBBIN^{-}$

Para as instâncias deste problema, pode-se aplicar uma estratégia similar a nosso algoritmo para o $BLOXORZ^{-}$. Saindo dos vértices correspondentes às posições '*', vale observar que há 3 deles para casa posição '*', há arcos que os ligam a vértices correspondentes a uma situação marcada por uma nova variável, situação onde os movimentos respeitam o padrão dos cubos isolados do Bloxorz considerando que um movimento do cubo é uma rotação do cilindro resultante no fim do processo. Na condição de existir um caminho onde se esgota a quantidade de movimentos que a posição '*' em particular permite, os arcos ligarão o vértice onde isso acontece de volta à região do grafo onde se mostram as regras de movimento do bloco cilíndrico. Vale ressaltar a unidirecionalidade dos arcos que entram e saem da dita "região do cubo". Os caminhos encontrados podem eventualmente não refletir uma sequência válida no caso de saírem da "região do cubo", que pode ser imaginada como uma camada extra do grafo sobre a fase, no número errado de passos, mas esse caminho indica ainda assim a existência da sequência válida. Para a construção da "região do cubo"no grafo, é necessário registrar todas as formas de se mover a partir da devida posição no estado de cubo pela dada quantidade de passos, o que



(a) Parte inicial (b) Para cada variável (c) Para cada cláusula

Figura 19 – Engrenagens da redução

teria custo potencialmente exponencial se feito utilizando força bruta, uma alternativa é recorrer a uma estratégia de programação dinânica onde regitra-se as subsequências de tamanho k e adiciona-se um movimento para obter as de tamanho k + 1. Essa construção será o passo mais custoso do algoritmo e demandará, seja n o tamanho da fase, até $n-2$ tabelas com n valores demarcando o número de passos como cubo necessários para alcançar um dado elemento a partir da posição '*' em questão. Ou seja o custo total de tempo é $O(n^2)$.

5.5 BOBBIN

Teorema 5.1. $SAT \leq_p BOBBIN$

Demonstração. Dada uma fórmula da lógica proposicional na CNF, pode-se construir uma fase hipotética do Bobbin 3D como na Figura 19. Nela, são mostradas as engrenagens da construção analogamente ao feito com o $BLOXORZ$, mas esta exige mais cuidado na análise.

O pareamento das chaves '+' , que não fica totalmente explícito, é o seguinte: se uma chave '+' é da região correspondente a um literal e não possui par na própria, seu par é uma chave '+' elevada presente na engrenagem de cláusula de uma daquelas onde o literal aparece. '?' se refere a uma coluna específica cujo valor depende de quantas vezes o literal mais frequente envolvendo a variável aparece na fórmula. Pela necessidade de simetria na construção e pela necessidade de manter a paridade da sequência de chaves, deve-se preencher as potenciais posições faltantes com piso normal. Além disso, deve-se considerar o encaixe das engrenagens. As unidades mais à esquerda de piso normal da primeira engrenagem de variável ficam na mesma altura das mais à direita da engrenagem

de início. As engrenagens de variável são encaixadas em altura igual uma à direita da outra. A primeira engrenagem de cláusula é encaixada à direita da última de variável na linha mais abaixo que possui piso normal. À direita das engrenagens de cláusula, na mesma linha de uma chave '+', será encaixado o buraco final.

Se a fórmula é satisfatível, a sequência de movimentos correspondente a passar pelas engrenagens de variável ativando chaves '+' da correspondentes ao literais satisfeitos por uma valoração que satisfaz a fórmula descreve uma sequência-solução para a fase, pois ao ativar essas chaves, seus pares nas engrenagens de cláusula serão abaixados permitindo a passagem do cilindro. Como toda cláusula tem um literal satisfeito pela valoração, todas as engrenagens de cláusula possuem uma chave '+' que será abaixada tornando isso verdade, permitindo que o cilindro alcance o buraco final. Resta observar que é possível passar pelas engrenagens de variável da forma descrita, pois, com a construção possuindo o número adequado de elementos de piso normal, o cilindro pode ser rolado até ficar deitado sobre o piso na mesma coluna que a posição elevada esquerda da construção para o literal e então com um movimento "bater"na posição elevada para em seguida ativar as chaves. Além disso, o cilindro pode passar pela chave elevada da construção para o literal, pois seu par está imediatamente à esquerda. Ou seja, a fase é uma instância sim do $BOBBIN$.

Por outro lado, se a fase contruída é uma instância sim do $BOBBIN$, existe uma sequência-solução para ela, ou seja, o buraco final pode ser alcançado, o que implica que pelo menos uma chave '+' de cada engrenagem de cláusula pode ser abaixada, ou seja, é possível passar pelas engrenagens de variável ativando as chaves correspondentes a pelo menos um literal de cada cláusula sem ativar chaves correspondentes a literais contraditórios entre si, pois, a partir de um bloco alcançável a partir de movimentos válidos que esteja na região de um literal, não é possível ir até a região da negação desse literal nem pela esquerda, pois uma sequência de movimentos para esquerda enquanto válidos leva a um bloco deitado na horizontal, e nem pela direita, pois, após abaixar a chave '+' na extrema direita da engrenagem de variável e abandoná-la, a chave '+' que abaixaria a chave elevada da região da negação do literal não pode ser alcançada. Conclui-se que a fórmula é satisfatível.

□

6 Tabela de problemas

Abaixo mostramos uma tabela com um resumo dos resultados de complexidade estabelecidos neste trabalho.

Problema	Decisão	Min
$BLOXORZ^{--}$	$O(n)$	$O(nlog(n))$
$BLOXORZ^{-}$	$O(n^2)$	$O(n^2log(n))$
$BLOXORZ$	NP-completo	inaprox $n^{1-\epsilon}$
k-$BLOXORZ$	$O(n^2)$	$O(n^2log(n))$
$CLOSED$-$BLOXORZ$	$O(n^2)$	NP-difícil
$CLOSED$-$BLOXORZ^{+}$	NP-completo	NP-difícil
ON-THE-$EDGE$	NP-completo	NP-difícil
$BOBBIN^{--}$	$O(n)$	$O(nlog(n))$
$BOBBIN^{-}$	$O(n^2)$	$O(n^2log(n))$
$BOBBIN$	NP-completo	NP-difícil

7 Conclusão

A partir do estudo de problemas conduzido neste trabalho, pode-se observar peculiaridades desse tipo de esforço e similaridades matemáticas entre jogos.

No que se refere às peculiaridades, destaca-se o desafio de trabalhar matematicamente regras que, apesar de dependerem de computadores, são quase sempre percebidas de um ponto de vista mais prático do que o exibido aqui. Ao longo da escrita do trabalho, foi repensado e reescrito várias vezes o formato das definições e até mesmo dos teoremas mesmo que a intuição sobre os resultados fosse clara. Além disso, os algoritmos utilizados para efetivamente resolver os problemas recorrem a algoritmos clássicos de busca. Outra característica peculiar deste tipo de estudo são as construções de instâncias de tamanhos e estruturas incompatíveis com a forma como os jogos realmente são jogados.

Sobre as similaridades entre os jogos, elas residem essecialmente nos problemas aos quais eles são tipicamente relacionados. Neste trabalho, muito recorreu-se ao SAT, problema a partir do qual foi possível fazer as devidas reduções a problemas extraídos do Bloxorz e do Bobbin 3D, o que revelou semelhanças de classes de complexidade e de algoritmos entre os jogos e os problemas definidos. Em estudos mencionados aqui, recorre-se a uma família de problemas diferente, mas onde continua sendo importante o conceito de operadores lógicos. Divergindo dessa linha, exploramos tambem neste trabalho um problema extraído do jogo On The Edge e seu relacionamento com o problema do caminho hamitoniano. Sobre esse jogo esperávamos inclusive um aprofundamento dos resultados, que se concentraram mais no Bloxorz e repercutiram em seu similar Bobbin 3D.

Como trabalhos futuros, podem ser sugeridos um esclarecimento maior sobre a complexidade de uma versão alternativa do BLOXORZ onde não haja o limite de movimentos, investigação sobre possíveis algoritmos aproximativos para o MIN-CLOSED-BLOXORZ, elaboração sobre possível algoritmo exponencial no número de teletransportes para o CLOSED-BLOXORZ+ e inclusão de versões alternativas para o BOBBIN.

Referências

G. Aloupis, E. D. Demaine e A. Guo, Classic Nintendo Games are (NP)-Hard, $\langle http://arxiv.org/abs/1203.1895 \rangle$ (2012).

E. D. Demaine, S. Hohenberger e D. Liben-Nowell,Tetris is Hard, Even to Approximate, *International Journal of Computational Geometry and Applications 14:1-2* (2004) 41–68.

E. D. Demaine, M. L. Demaine e R. Fleisher, Solitaire Clobber, *Theoretical Computer Science 313:3* (2004) 325–338

E.D. Demaine, M. L. Demaine, A. Langerman e Stefan Langerman, Morpion Solitaire, *Theory of Computing Systems 39:3* (2006) 439–453

E. D. Demaine e M. L. Demaine, Jigsaw Puzzles, Edge Matching and Polyomino Packing: Connections and Complexity, *Graphs and Combinatorics 23 (Supplement)* (2007) 195–208

E.W. Dijkstra, A Note on Two Problems in Connexion with Graphs, *Numerische Mathematlk 1*(1959) 269–271.

D. Dor e U. Zwick, SOKOBAN and other motion planning problems,*Computational Geometry: Theory and Applications - COMGEO, vol. 13, no. 4* (1999) 215–228.

G. W. Flake e E. B. Baum, Rush Hour is PSPACE-complete or 'Why You Should Generously Tip Parking Lot Attendants', *Theoretical Computer Science 270:1-2* (2002), 895–911

A. S. Fraenkel e E. Goldschmidt, PSPACE-Hardness of Some Combinatorial Games, *Journal of Combinatorial Theory, Series A 46* (1987), 21 – 38

R. A. Hearn e E. D. Demaine, PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation, *Theoretical Computer Science, "Game Theory Meets Theoretical Computer Science"Special Issue, 343:1-2* (2005) 72–96

R. A. Hearn e E. D. Demaine, Games, Puzzles & Computation (2009), Editora AKPeters.

A. Itai, C. H. Papadimitriou e J. L. Szwarcfiter, Hamilton Paths in Grid Graphs, *Society for Industrial and Applied Mathematics Journal on Computing - SICOMP, vol. 11, no. 4* (1982)