Universidade Federal do Paraná

Ricardo Tavares de Oliveira

Arco Consistência Generalizada em Codificações SAT Relativas

Curitiba PR
2017

Ricardo Tavares de Oliveira

Arco Consistência Generalizada em Codificações SAT Relativas

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Fabiano Silva.

Curitiba PR
2017

Oliveira, Ricardo Tavares de
Arco consistência generalizada em codificações SAT relativas / Ricardo Tavares de Oliveira . – Curitiba, 2017
73 f. : il.; tabs.

Tese (doutorado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Fabiano Silva
Bibliografia: p. 67-71

1. Programação linear – Processamento de dados. 2. Otimiza - ção matemática. I. Silva, Fabiano. II. Título.

CDD 005.133

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós-Graduação INFORMÁTICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **RICARDO TAVARES DE OLIVEIRA** intitulada: **Arco Consistência Generalizada em Codificações SAT Relativas**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO.

Curitiba, 21 de Fevereiro de 2017.

FABIANO SILVA
Presidente da Banca Examinadora (UFPR)

RENATO JOSÉ DA SILVA CARMO
Avaliador Interno (UFPR)

GUSTAVO ALBERTO GIMENEZ LUGO
Avaliador Externo (UTFPR)

MARCOS ALEXANDRE CASTILHO
Avaliador Interno (UFPR)

LUERBIO FARIA
Avaliador Externo (UERJ)

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

Dedicado a Roseli Tavares e
Hélio Luiz de Almeida Oliveira

Agradecimentos

Agradeço ao meu orientador, Prof. Dr. Fabiano Silva, pela grande orientação, por sempre demonstrar apoio, e pela grande disponibilidade para compartilhar valiosos conhecimentos e importantes conselhos durante o desenvolvimento de todo o trabalho. Agradeço também, em particular, aos Profs. Drs. Renato Carmo e Marcos Castilho, por também contribuírem para o desenvolvimento do trabalho durante encontros, de maneira sábia e objetiva.

Agradeço aos meus pais, Roseli Tavares e Hélio Oliveira, assim como a meu irmão, Rafael Oliveira, por sempre demonstrarem total apoio a todos meus projetos e iniciativas.

Agradeço aos meus colegas de Laboratório no LIAMF, e em particular a Razer Montaño, Rodolfo Rodovalho, Clariane Menezes, Guilherme Derenievicz, Willian Zalewski, Marcos Schreiner e Franck Benito, pela grande amizade e companhia diária, assim como pelos vários momentos de boa iteração. Agradeço também aos amigos formados no PPGInf como um todo durante todos estes anos, como Elisabete Ferreira, Renato Melo e Carlos Zava.

Agradeço a todos os grandes amigos formados nos últimos anos na universidade, em especial a Jedian Brambilla, pela excelente amizade, assim como a Hamer Iboshi, Leonardo Strozzi, Fernando Erd, Gabriel Cândido, Stephanie Americo, Talita Halboth, Giovanne Santos, Maíra Oneda, Bruno Tissei, Felipe Wu, Marcela e Lucas Oliveira, e a todos com quem compartilho os dias na universidade desde o café da manhã até à noite.

Agradeço a todos os meus amigos que me acompanham desde a graduação na UFPR, em particular aos da turma de 2007 de BCC, Paulo Ferreira, Vinicius Ruoso, Grazielle Vernize, William Komura, Juan Ruibal, Fernando Gielow e Aramis Fernandes.

Agradeço a todos da Maratona de Programação, competição a qual auxiliou no desenvolvimento da criatividade, raciocínio, e capacidade de programação rápida. Em particular, agradeço ao Prof. Dr. André Guedes por encabeçar a maratona na UFPR. Agradeço também a Eduardo Ribas pela grande ajuda nos treinamentos em todos esses anos. Agradeço também a meus colegas de equipe, já citados Fernando, Vinícius e Eduardo, mas também a Flávio Zavan, pela formação de equipes inesquecíveis e pela continuidade da tradição da UFPR na competição, assim como pela grande amizade mantida até hoje. Agradeço também aos demais competidores da UFPR, em particular a Rodrigo Gryzinski, Félix Lyu, Luiz Reis, Thiago Abdo, Ruanito Santos, Jomaro Rodrigues e João Barth, mas também a todos os demais que representam nossa universidade. Agradeço também aos amigos formados pela competição, como Paulo César, André Augusto e Wisllay Vitrio, assim como a todos que contribuem para a competição.

Agradeço a Andrea Pucci pelas excelentes análises e conselhos fundamentais para o desenvolvimento e término do trabalho.

Agradeço à CAPES pelo financiamento deste trabalho. Agradeço também a Loukas Georgiadis, Giuseppe Italiano, Luigi Laura e Federico Santaroni pela disponibilidade da fonte de seus algoritmos de manutenção da árvore de dominantes de um grafo dinâmico.

Agradeço ao C3SL, onde estagiei durante quase toda a graduação, pela oportunidade de aprimorar conhecimentos necessários para o bom desenvolvimento de *software*. Agradeço também aos amigos lá formados, como Diego Pasqualin, Danilo Yorinori, Rubens Sugimoto, Cleide Possamai, Josiney de Souza, Thiago Picharski e todos os demais.

Agradeço a todo o corpo docente do DINF, como os Profs. Drs. Alexandre Direne, Roberto Hexel, Elias Duarte, Carmem Hara e todos os demais, por terem transmitido, de maneira única, todos os conhecimentos necessários para a formação de um cientista da Computação durante todos os 10 anos que passei na universidade.

Por fim, agradeço a todos os alunos das disciplinas que tive a alegria de poder ministrar até aqui, por poder, de alguma forma, contribuir um pouco com a formação de todos.

Resumo

Várias codificações de problemas relevantes para SAT ou suas variações são conhecidas e estudadas pela comunidade. Uma possível maneira de mensurar a eficiência destas codificações consiste em avaliar a manutenção da Arco Consistência Generalizada (ACG) da fórmula resultante pelo resolvedor SAT. Ao melhor de nosso conhecimento, não há estudos sobre a manutenção de tal consistência para codificações relativas, que descrevem relações binárias sobre um dado conjunto de elementos do problema. Neste trabalho, é apresentado um estudo sobre a Arco Consistência Generalizada em codificações SAT relativas. É mostrado que, dependendo das propriedades da relação codificada, as fórmulas obtidas por estas codificações são mantidas ACG pelo procedimento da Propagação Unitária. Conjectura-se também que estas codificações não podem ser polinomialmente restritas para mantê-las ACG. Neste trabalho, é também apresentado um método para manter uma consistência parcial nas fórmulas obtidas pelas codificações relativas. O método é baseado na manutenção da árvore de dominantes do grafo induzido pela relação codificada, durante o processo de busca do resolvedor SAT. Resultados experimentais indicam que o método pode reduzir o número de decisões realizadas pelo resolvedor e, logo, o espaço de busca explorado. Outras contribuições deste trabalho incluem codificações relativas para conectividade em grafos e para o problema da Árvore de Steiner, um *framework* genérico para unificar as codificações relativas conhecidas, um algoritmo polinomial para SAT para fórmulas mantidas ACG, e a implementação de algoritmos que mantêm árvores de dominantes para grafos dinâmicos dentro do código-fonte de um resolvedor SAT no estado-da-arte.

Abstract

Many encodings from relevant problems to SAT or its variants are known and studied by the community. One possible way to measure the efficiency of these encodings consists on evaluating the maintenance of the Generalized Arc Consistency (GAC) of the resulting formula by the SAT solver. To the best of our knowledge, there is no study about such consistency for relative encodings, that describe binary relations over a given set of elements from the problem. In this work, it is presented a study about the Generalized Arc Consistency for relative SAT encodings. It is shown that, depending on the properties of the encoded relation, the formulae obtained by such encodings are maintened GAC by the Unit Propagation procedure. It is also conjectured that these encodings cannot be polynomially restricted in order to maintain them GAC. In this work, it is also presented a method to maintain a partial consistency on the formulae obtained by relative encoding. The method is based on the maintenance of the dominator tree of the underlying graph, during the search procedure of the SAT solver. Experimental evaluations indicate that the method may reduce the number of decisions made by the solver, and, thus, the search space it explores. Other contributions of this work include relative encodings for graph connectivity and for the Steiner Tree problem, a generic framework unifying known relative encodings, a polynomial-time algorithm for SAT for formulae maintened GAC, and the implementation of algorithms that maintain dominator trees for dynamic graphs inside the source code of a state-of-the-art SAT solver.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ACG	Arco Consistência Generalizada (*Generalized Arc Consistency*)
CNF	Forma Normal Conjuntiva (*Conjunctive Normal Form*)
CSP	Satisfação de Restrições (*Constraints Satisfaction Problem*)
DAG	Grafo Direcionado Acíclico (*Directed Acyclic Graph*)
FC	*Forward-Checking*
MaxSAT	Satisfabilidade Booleana Máxima (*Maximum SAT*)
PWMaxSAT	Satisfabilidade Booleana Máxima Ponderada e Parcial (*Partial-Weighted MaxSAT*)
SAT	Satisfabilidade Booleana (*Boolean Satisfability*)
WMaxSAT	Satisfabilidade Booleana Máxima Ponderada (*Weighted MaxSAT*)

Capítulo 1

Introdução

A redutibilidade entre problemas computacionais é um mecanismo elementar que embasa a Ciência da Computação. Como aplicação teórica, é utilizada para definir classes de problemas computacionais. Na prática, um problema pode ser resolvido através da solução de outro problema caso haja uma redução eficiente entre ambos.

Um problema particularmente interessante no contexto de redutibilidade é o problema de Satisfabilidade Booleana (SAT). O problema consiste em, dada uma fórmula booleana, decidir se existe uma valoração de suas variáveis que a satisfaz, isto é, tal que a fórmula assume o valor verdade 1 (verdadeiro) sob a valoração de suas variáveis. Este problema se destaca na área de redutibilidade por estar provadamente em NP-Completo [Cook, 1971], e, logo, todos os problemas em P e NP podem ser reduzidos a SAT em tempo polinomial.

Além do problema SAT, destaca-se também o problema de Satisfabilidade Booleana Máxima (MaxSAT) e suas generalizações. O problema MaxSAT consiste em, dada uma fórmula booleana na Forma Normal Conjuntiva (CNF), determinar o número máximo de cláusulas que podem ser satisfeitas simultaneamente por uma valoração de suas variáveis. Informalmente, o problema MaxSAT é considerado uma variação do problema de decisão NP-Completo SAT para um problema de otimização NP-Difícil.

A comunidade científica tem colocado muito esforço no desenvolvimento de resolvedores SAT e MaxSAT eficientes nas últimas décadas [Davis et al., 1962, Zhang, 1997, Marques-Silva e Sakallah, 1996, Moskewicz et al., 2001, Buro e Büning, 1993, Freeman, 1995, Jeroslow e Wang, 1990, Eén e Sörensson, 2004, Biere, 2013, Audemard e Simon, 2009, Heras et al., 2008, Selman et al., 1992, Smyth et al., 2003, Fu e Malik, 2006, Ansótegui et al., 2009, Di Rosa et al., 2010], o que torna "tratável" a resolução de diversas instâncias consideradas grandes, mesmo com os problemas nas classes NP-Completo e NP-Difícil. Por este motivo, é relevante estudar reduções de problemas considerados difíceis para SAT ou problemas similares, para possibilitar suas resoluções com o uso de resolvedores SAT no estado-da-arte.

Desta forma, diversos problemas de interesse, tanto de decisão quanto de otimização, podem e são reduzidos para SAT ou MaxSAT. Dentre os problemas de interesse, podem ser citados problemas cujas instâncias são ou contêm grafos, como o problema de Coloração, do Ciclo Hamiltoniano ou do Caixeiro Viajante, da Árvore de Steiner, de Clique e de Clique Máxima. A resolução destes problemas tem aplicações teóricas e diversas aplicações práticas, como na área de escalonamento, roteamento em redes, geometria computacional, projeto de circuitos digitais e análise de redes sociais.

Reduções destes problemas para SAT e MaxSAT são conhecidas e estudadas na literatura [Gelder, 2008, Velev, 2007, Prestwich, 2003, Hertel et al., 2007,

Iwama e Miyazaki, 1994, de Oliveira, 2013, de Oliveira et al., 2012, Oliveira e Silva, 2014, Kautz et al., 1996, Lenhardt, 2010, Li e Quan, 2010]. Tais reduções podem ser classificadas de acordo com a natureza das instâncias codificadas.

Codificações absolutas consistem em reduções do Problema de Satisfação de Restrições (CSP) para SAT [De Kleer, 1989, Kasif, 1990, Velev e Gao, 2009]. Em codificações absolutas, uma variável booleana $a_{X,v}$ é criada para toda variável X da instância CSP dada e todo valor v de seu domínio, indicando a relação $X = v$ nas soluções encontradas para o problema.

Codificações relativas, por sua vez, consistem em reduções que codificam uma relação binária, comumente sobre arestas de um grafo dado [Prestwich, 2003, Velev e Gao, 2010, Bryant e Velev, 2002, Oliveira e Silva, 2014, de Oliveira e Silva, 2015]. Como exemplo, um caminho hamiltoniano de um grafo pode ser descrito como uma relação binária transitiva, antisimétrica e total dos seus vértices. Nestas codificações, uma variável booleana $r_{a,b}$ é criada para indicar a relação entre um par de elementos a e b na relação codificada.

Por fim, também é possível codificar operadores de cardinalidade em fórmulas booleanas utilizando vários métodos conhecidos na literatura [Abío et al., 2012, Bailleux et al., 2009, Sinz, 2005]. Estas codificações descrevem restrições com relação ao número de variáveis que assumem o valor 1 (verdadeiro) ou o valor 0 (falso) em um dado conjunto de variáveis.

Naturalmente, a eficiência de uma redução pode ser medida pelo tamanho da fórmula gerada pela mesma. Entretanto, além do tamanho da fórmula gerada, outras propriedades das instâncias obtidas podem ser analisadas para mensurar a eficiência de uma redução. Em particular, é possível analisar a fórmula obtida, informalmente, em relação a quão poderosos são os procedimentos polinomiais de inferência dos resolvedores SAT ao resolvê-la. Tal análise pode ser feita pela manutenção da Arco Consistência Generalizada da fórmula obtida pela redução.

O conceito de Arco Consistência Generalizada (ACG), comumente estudado no contexto do problema CSP, pode ser também descrito no contexto do problema SAT [Mackworth, 1977]. No contexto do problema CSP, manter uma instância ACG consiste em utilizar um procedimento polinomial a cada passo da busca para remover, dos domínios das variáveis da instância dada, valores que não satisfazem as restrições presentes na instância. Quando o problema é reduzido para SAT através de codificações absolutas, tal manutenção é equivalente a valorar com 0 (falso) toda variável $a_{X,v}$ tal que não existe uma solução na qual o valor (inconsistente) v é atribuído à variável X. Tal valoração pode ser realizada pelo procedimento polinomial da Propagação Unitária, implementada e utilizada por todos os resolvedores SAT no estado-da-arte. Ao manter uma instância ACG pela Propagação Unitária, todo o espaço de busca inconsistente que contém tal valoração é podado, o que pode reduzir o tamanho do espaço de busca explorado pelos resolvedores. A manutenção da Arco Consistência Generalizada pela Propagação Unitária sobre codificações absolutas é conhecida e estudada pela comunidade [Walsh, 2000, Gent, 2002].

O conceito de Arco Consistência Generalizada e sua manutenção também é estudado para codificações de operadores de cardinalidade [Abío et al., 2012, Bailleux e Boufkhad, 2003, Bailleux et al., 2009]. A Arco Consistência Generalizada da fórmula resultante de uma codificação de um operador de cardinalidade é mantida pela Propagação Unitária quando tal procedimento valora com 0 (falso) toda variável que deve necessariamente assumir este valor para que a fórmula resultante seja satisfeita.

Embora o conceito de manutenção da Arco Consistência Generalizada seja conhecido e estudado na literatura para codificações absolutas e codificações de operadores de cardinalidade, ao melhor do nosso conhecimento, não há um estudo sobre Arco Consistência Generalizada para codificações relativas. Desta forma, o principal objetivo deste trabalho consiste em apresentar um estudo sobre a manutenção da Arco Consistência Generalizada em fórmulas booleanas obtidas por codificações SAT relativas.

As duas principais contribuições deste trabalho consistem em:

- Uma análise teórica sobre a Arco Consistência Generalizada em codificações SAT relativas;
- Apresentação, implementação e discussão de resultados experimentais de uma técnica para manter um nível parcial de consistência para codificações SAT relativas.

Neste trabalho, prova-se que, se a relação binária codificada tiver um conjunto de propriedades específicas, a fórmula obtida pela codificação relativa é mantida ACG pela Propagação Unitária. Além disso, se tal relação tiver um outro conjunto de propriedades, tal fórmula não é mantida ACG pelo mesmo procedimento. Conjectura-se também que não é possível adicionar à fórmula informações redundantes, de tamanho polinomial, para fazê-la ser mantida ACG pela Propagação Unitária, mesmo se o problema original sendo reduzido for polinomial. Alguns indícios são apresentados para dar suporte à conjectura.

Mesmo não sendo possível manter a Arco Consistência Generalizada pela Propagação Unitária para algumas codificações relativas, é possível manter um nível parcial de consistência. Este nível é obtido através de um procedimento polinomial utilizado para valorar algumas (possivelmente não todas) variáveis que devem assumir 0 (falso) para que a fórmula obtida seja satisfeita, e cujo valor não é inferido pela Propagação Unitária. Neste trabalho, utiliza-se como procedimento a manutenção da árvore de dominantes do grafo induzido pela relação binária codificada.

Além das duas principais contribuições apresentadas, também são contribuições deste trabalho:

- A apresentação de codificações relativas para o problema de conectividade de grafos e para o problema da Árvore de Steiner, sugeridas e publicadas pelos autores deste trabalho [de Oliveira e Silva, 2015, Oliveira e Silva, 2014];
- A unificação das codificações relativas conhecidas na literatura em um *framework*;
- A apresentação de um algoritmo polinomial para SAT para fórmulas mantidas ACG pela Propagação Unitária;
- A alteração no código-fonte de um resolvedor SAT aberto para a implementação dos algoritmos para manutenção da árvore de dominantes em grafos dinâmicos.

O restante deste documento está organizado da seguinte maneira: O capítulo 2 apresenta definições preliminares necessárias para o entendimento do trabalho. O capítulo 3 apresenta as codificações SAT conhecidas na literatura, assim como o *framework* proposto para as codificações relativas em particular. O capítulo 4 apresenta o estudo sobre a manutenção da Arco Consistência Generalizada para codificações relativas, primeira principal contribuição deste trabalho. O capítulo 5 apresenta e discute a técnica de manutenção parcial de tal consistência, segunda principal contribuição do mesmo. Por fim, o capítulo 6 conclui o trabalho e apresenta possíveis trabalhos futuros.

Capítulo 2

Preliminares

Este capítulo apresenta definições e conceitos inerentes ao problema da Satisfabilidade Booleana (SAT), assim como aos problemas relacionados de Satisfabilidade Booleana Máxima (MaxSAT) e Satisfação de Restrições (CSP). As definições apresentadas são utilizadas nos capítulos seguintes.

2.1 Lógica Proposicional e Satisfabilidade Booleana (SAT)

Esta seção apresenta definições preliminares a respeito da Lógica Proposicional, do problema da Satisfabilidade Booleana (SAT) e de suas soluções.

Primeiramente, uma variável *booleana* é uma variável cujo domínio consiste nos *valores verdade* 0 (falso) e 1 (verdadeiro). Além disso, uma *fórmula* é recursivamente definida como uma variável booleana ou uma sequência de (sub)fórmulas conectadas por um *operador*. Um *operador* (de aridade k), por sua vez, é um símbolo que conecta (k) (sub)fórmulas.

A Lógica Proposicional define três operadores básicos:

- O operador unário de *negação* ($\neg$), representando o "não" lógico;
- O operador $k-$ário de *disjunção* ($\vee$), representando o "ou" lógico.
- O operador $k-$ário de *conjunção* ($\wedge$), representando o "e" lógico;

Além destes operadores, os símbolos $\rightarrow$ (*implicação*) e $\leftrightarrow$ (*dupla implicação*) também podem ser utilizados para simplificar a representação de algumas fórmulas. A notação $x \rightarrow y$ representa a fórmula $(\neg x) \vee y$, enquanto $x \leftrightarrow y$ representa a fórmula $((\neg x) \vee y) \wedge ((\neg y) \vee x)$.

Um *literal* l é uma variável x (dito literal *positivo*) ou sua negação $\neg x$ (dito literal *negativo*).

Uma variável *ocorre* em uma fórmula se a fórmula consiste naquela variável ou se a variável ocorre em alguma de suas subfórmulas. Neste texto, $X(F)$ denota o conjunto de variáveis que ocorrem na fórmula F, e $L(F) = \bigcup_{x \in X(F)} \{x, \neg x\}$ denota o conjunto dos literais das variáveis que ocorrem em F.

Uma *valoração* (ou *interpretação*) sobre $X(F)$ é uma função $A : X'(F) \rightarrow \{0$ (falso), 1(verdadeiro)$\}$, com $X'(F) \subseteq X(F)$. Desta forma, uma valoração é uma função que mapeia alguma(s) variável(is) em $X(F)$ para valores verdade. Uma valoração A também pode ser descrita como um conjunto[1] de literais A tal que $x \in A$ se a variável x assume 1 (verdadeiro) sob a valoração, e $\neg x \in A$ se a variável x assume 0 (falso) sob a mesma.

[1]Com esta notação, é possível utilizar símbolos da álgebra de conjuntos ($\in$, $\subset$, etc.)

Uma valoração A é *completa* se todas as variáveis em $X(F)$ são mapeadas, isto é, se $X'(F) = X(F)$, ou *parcial* caso contrário, isto é, se $X'(F) \subset X(F)$. Alternativamente, uma valoração é completa se $x \in A$ ou $\neg x \in A$ para todo $x \in X(F)$, e parcial caso contrário. Além disso, uma valoração A é uma *extensão* de uma valoração parcial A' (e, logo, A' pode ser *estendida* para A) se $A' \subset A$.

Dada uma valoração A, uma fórmula F assume o valor 0 (falso) se:

- A fórmula é uma variável e a variável assume 0 (falso) na valoração dada, isto é, se $F = x$ e $A(x) = 0$ (ou $\neg x \in A$); ou
- A fórmula é uma negação e sua subfórmula assume o valor 1 (verdadeiro); ou
- A fórmula é uma disjunção e todas suas subfórmulas assumem o valor 0 (falso); ou
- A fórmula é uma conjunção e ao menos uma de suas subfórmulas assume o valor 0 (falso).

De forma análoga, sob uma valoração A, uma fórmula F assume o valor 1 (verdadeiro) se:

- A fórmula é uma variável e a variável assume 1 (verdadeiro) na valoração dada, isto é, se $F = x$ e $A(x) = 1$ (ou $x \in A$); ou
- A fórmula é uma negação e sua subfórmula assume o valor 0 (falso); ou
- A fórmula é uma disjunção e ao menos uma de suas subfórmulas assume o valor 1 (verdadeiro); ou
- A fórmula é uma conjunção e todas suas subfórmulas assumem o valor 1 (verdadeiro).

Por fim, uma valoração A *satisfaz* uma fórmula F se F assume o valor 1 (verdadeiro) sob a valoração A.

Uma fórmula é *válida* se todas as valorações completas possíveis sobre suas variáveis a satisfazem, e *não válida* caso contrário. Além disso, uma fórmula é *satisfatível* se alguma valoração completa de suas variáveis a satisfaz, e *insatisfatível* caso contrário. Uma valoração completa que satisfaz uma fórmula F é um *modelo* de F.

Uma *cláusula* é uma disjunção de literais $C = (l_1 \vee l_2 \vee ... \vee l_{|C|})$. Alternativamente, uma cláusula pode ser descrita por um conjunto finito de literais $C = \{l_1, l_2, .., l_{|C|}\}$. O tamanho de uma cláusula C, denotado por $|C|$, é dado pelo número de literais presentes na cláusula. Uma cláusula C é *binária* se $|C| = 2$, *unitária* se $|C| = 1$ e *vazia* se $|C| = 0$. A cláusula vazia é denotada por $\square$. É válido observar que uma valoração A satisfaz uma cláusula C se $A \cap C \neq \emptyset$. Além disso, a cláusula vazia $\square$ sempre assume o valor 0 (falso) e, portanto, não é satisfeita por nenhuma valoração possível.

Uma fórmula na *Forma Normal Conjuntiva (CNF)* consiste em uma conjunção ($\wedge$) de cláusulas. Assim como uma cláusula pode ser descrita como um conjunto de literais, uma fórmula em CNF pode ser descrita como um conjunto de cláusulas. Entretanto, o *tamanho* de uma fórmula F, denotado por $|F|$, é o número total de literais que suas cláusulas contém, isto é, $|F| = \sum_{C_i \in F} |C_i|$.

A *regra de resolução* é uma operação que, dadas duas cláusulas $(A \vee x)$ e $(B \vee \neg x)$, resulta na cláusula $(A \vee B)$. Uma *árvore de resolução* para uma fórmula F em CNF é uma árvore binária cujas folhas consistem nas cláusulas de F e cujos nodos internos consistem em cláusulas resultantes da regra de resolução com seus filhos.

Uma árvore de *refutação* é uma árvore de resolução cuja raiz consiste na cláusula vazia (□). A figura 2.1 exemplifica uma árvore de refutação para a fórmula $F = (\neg y) \wedge (x \vee y) \wedge (\neg x \vee z) \wedge (\neg z)$. Se uma fórmula F em CNF admite uma árvore de refutação, então $\neg F$ é uma fórmula válida [Davis et al., 1962].

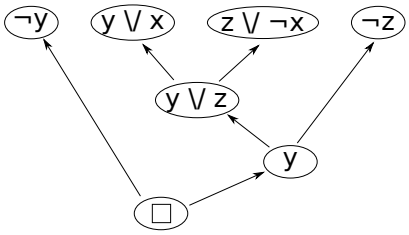

Figura 2.1: Árvore de refutação para a fórmula $F = (\neg y) \wedge (y \vee x) \wedge (z \vee \neg x) \wedge (\neg z)$

Por fim, o problema (de decisão) de *Satisfabilidade Booleana* (SAT) consiste em, dada uma fórmula, determinar se ela possui algum modelo, isto é, se existe alguma valoração completa sobre suas variáveis que satisfaz a fórmula dada. Se a fórmula dada estiver em CNF, o problema consiste em encontrar uma valoração que satisfaça todas as suas cláusulas simultaneamente. O problema SAT foi o primeiro problema cuja pertinência à classe de problemas NP-Completo foi provada [Cook, 1971].

Embora o problema esteja em NP-Completo, muito esforço foi colocado no desenvolvimento de resolvedores SAT nas últimas décadas [Davis et al., 1962, Zhang, 1997, Marques-Silva e Sakallah, 1996, Moskewicz et al., 2001, Buro e Büning, 1993, Freeman, 1995, Jeroslow e Wang, 1990, Eén e Sörensson, 2004, Biere, 2013]. Isto torna o problema "tratável" para diversas instâncias práticas do mesmo, o que motiva o estudo de reduções polinomiais de outros problemas computacionais de decisão para o problema SAT.

Reduzir polinomialmente um problema de decisão para SAT consiste em, dada uma instância I do problema original, definir uma fórmula booleana F, de tamanho polinomial no tamanho da instância I, tal que F é satisfatível se e somente se a instância I é positiva, isto é, se a resposta do problema original para tal instância é "sim". Além disso, deve ser possível obter um certificado para a instância dada a partir de um modelo da fórmula construída, que deve ser verificável em tempo polinomial.

Informalmente, reduzir um problema para SAT consiste em "traduzir" uma descrição do mesmo para a linguagem definida pela Lógica Proposicional. Neste contexto, o termo *codificação* é utilizado para definir o processo de tradução de uma descrição para a Lógica Proposicional, seja a descrição de um problema ou a descrição de uma restrição.

Desta forma, codificar uma restrição ou condição consiste em construir uma fórmula tal que a condição ou restrição é satisfeita se e somente se a fórmula construída é satisfeita. Da mesma forma, codificar um problema de decisão consiste em construir uma fórmula booleana satisfatível se e somente se a instância pode ser "satisfeita", isto é, tem "sim" como resposta. Assim, codificar um problema de decisão consiste, efetivamente, em reduzí-lo para SAT.

Os resolvedores SAT mais eficientes no estado-da-arte resolvem instâncias na Forma Normal Conjuntiva [Biere, 2013, Audemard e Simon, 2009, Zhang, 1997, Marques-Silva e Sakallah, 1996]. Por este motivo, em codificações para SAT, pode ser necessário converter uma fórmula que não está na Forma Normal Conjuntiva para CNF. Embora seja possível realizar tal conversão com o uso de equivalências lógicas, como a propriedade

distributiva e as leis de DeMorgan[2], este procedimento pode gerar uma fórmula de tamanho exponencial no tamanho da fórmula original.

A transformação de Tseitin, por sua vez, converte uma fórmula para CNF em tempo linear em seu tamanho, através da criação de novas variáveis booleanas [Tseitin, 1968]. Cada operador da fórmula é substituído por uma nova variável, e cláusulas indicando a equivalência entre a variável e o operador são criadas. A subfórmula $(x_1 \wedge x_2)$, por exemplo, é substituída por uma nova variável y_i, e a relação $y_i \leftrightarrow (x_1 \wedge x_2)$ é codificada, pela criação das cláusulas $y_i \rightarrow (x_1 \wedge x_2) = \neg y_i \vee (x_1 \wedge x_2) = (\neg y_i \vee x_1) \wedge (\neg y_i \vee x_2)$, e com a cláusula $(x_1 \wedge x_2) \rightarrow y_i = (\neg x_1 \vee \neg x_2 \vee y_i)$. A fórmula obtida é satisfatível se e somente se a fórmula original é satisfatível.

Também é possível codificar a condição de que o valor verdade assumido por uma subfórmula F_i deve ser fixado, isto é, a condição de que a subfórmula F_i deve necessariamente assumir o valor 0 (falso) ou o valor 1 (verdadeiro) em todos os modelos da fórmula construída. Este fato pode ser codificado com a adição de uma cláusula unitária $(\neg y_i)$ ou (y_i), onde y_i é a variável criada para representar o operador que descreve F_i. É suficiente, neste caso, codificar apenas uma dentre as relações $y_i \rightarrow F_i$ e $F_i \rightarrow y_i$, de acordo com o valor verdade que a subfórmula deve assumir [Plaisted e Greenbaum, 1986]. Outras técnicas comuns de codificações SAT são apresentadas no Capítulo 3.

Os resolvedores SAT no estado-da-arte mais eficientes conhecidos resolvem instâncias em CNF e utilizam, como procedimento base, o algoritmo DPLL [Davis et al., 1962]. O algoritmo DPLL, por sua vez, utiliza a técnica de *backtracking*. A cada passo, uma variável $x_i \in X(F)$ é escolhida (ou *decidida*), e um valor verdade é associado a ela. Esta associação descreve o literal $\neg x_i$ (se x_i é associado a 0 (falso)) ou x_i (se x_i é associado a 1 (verdadeiro)). Este literal é então *propagado* na fórmula.

O processo de propagação consiste em substituir as ocorrências da variável pelo valor escolhido, potencialmente tornando a fórmula menor. O problema é então resolvido recursivamente. Caso a fórmula resultante da propagação seja insatisfatível, um *conflito* é encontrado, um *retrocesso* é realizado, e propaga-se o literal oposto do valor propagado anteriormente para x_i. O problema é novamente resolvido recursivamente. Se a fórmula resultante desta propagação também é insatisfatível, então a fórmula referente àquele passo do algoritmo também o é. Durante o processo, a *valoração parcial atual* A é mantida, contendo todas as propagações realizadas na fórmula atualmente. É válido notar que, se um modelo da fórmula pode ser encontrado no ramo atual da árvore de busca, então tal modelo é uma extensão da valoração parcial atual A.

A propagação de um literal l consiste em remover da fórmula em CNF as cláusulas que contém l, e remover $\neg l$ das cláusulas que contém tal literal. Conclui-se que uma fórmula é insatisfatível se contém uma cláusula vazia, e que a mesma é satisfatível se não contém cláusulas.

A cada passo do algoritmo DPLL, antes da decisão de uma variável, o procedimento de *Propagação Unitária*, descrito pelo algoritmo 1, é aplicado. O procedimento pode simplificar a fórmula através da propagação dos literais que ocorrem em cláusulas unitárias. Primeiramente, todos os literais que ocorrem em cláusulas unitárias da fórmula são enfileirados em uma fila Q (linhas 1 a 3). Então, enquanto a fila Q não estiver vazia, um literal l é removido de Q e propagado na fórmula (linhas 4 a 6). Se a fórmula resultante contiver outros literais que ocorrem em cláusulas agora unitárias, estes literais são enfileirados em Q (linhas 7 a 9), e propagados eventualmente.

Considere como exemplo a fórmula $F = (x) \wedge (\neg x \vee \neg y) \wedge (y \vee w \vee z)$. Como (x) é uma cláusula unitária, o literal x é propagado na fórmula, resultando em $(\neg y) \wedge (y \vee w \vee z)$.

[2] $\neg(x \vee y) = (\neg x \wedge \neg y)$, e $\neg(x \wedge y) = (\neg x \vee \neg y)$

Algoritmo 1: Procedimento de Propagação Unitária

Entrada: Uma fórmula F
Saída: A fórmula F possivelmente simplificada

```
1  para todo l tal que (l) ∈ F faça
2      enfilera(Q, l)
3  fim
4  enquanto Q não está vazia faça
5      l = desenfilera(Q)
6      F = propagação de l em F
7      se existe l' tal que (l') ∈ F e l' ∉ Q então
8          enfilera(Q, l')
9      fim
10 fim
11 retorna F
```

Como a cláusula $(\neg y)$ é agora unitária, o literal $\neg y$ é propagado em seguida, o que resulta na fórmula $(w \vee z)$.

É válido enfatizar que a propagação de um literal realizada por este procedimento também indica a associação de um valor verdade a uma variável booleana, assim como a associação realizada a cada passo do DPLL. Além disso, como propagar um literal consiste em remover sua negação das cláusulas que a contém, note que propagar o literal l devido à cláusula unitária (l) é análogo a aplicar a regra de resolução com a cláusula unitária dada (l) e cada outra cláusula da fórmula $C_i = (C_i' \vee \neg l)$ que contém o literal $\neg l$, o que resulta na cláusula C_i' sem a negação do literal.

Além do procedimento de Propagação Unitária, outros procedimentos são realizados pelos resolvedores SAT no estado-da-arte. Além de procedimentos eficientes, os resolvedores também utilizam estruturas de dados sofisticadas, assim como heurísticas para decisões. Podem ser citadas a estrutura de *literais vigiados* para manutenção de cláusulas [Zhang, 1997], as técnicas de *aprendizado por conflito* e *backtracking* não cronológico [Marques-Silva e Sakallah, 1996], *reinícios* [Huang, 2007], e *heurísticas de ramificação*, como a Bohm [Buro e Büning, 1993], MOM [Freeman, 1995], Jeroslow-Wang [Jeroslow e Wang, 1990] e VSIDS [Moskewicz et al., 2001]. Embora estas técnicas influenciem muito no desempenho de um resolvedor SAT, suas descrições são omitidas deste texto por não pertencerem ao foco do mesmo.

Também é válido citar algumas especializações de SAT. Primeiramente, uma fórmula F está em $k-$CNF se $|C_i| \leq k$ para todo $C_i \in F$, isto é, se toda cláusula contém até k literais. O problema $k-$SAT consiste na especialização de SAT cujas instâncias são fórmulas em $k-$CNF. O problema SAT pode ser reduzido para 3-SAT em tempo linear no tamanho da fórmula original com a transformação de Tseitin [Tseitin, 1968]. Além disso, o problema 2-SAT pode ser resolvido em tempo linear no tamanho total da fórmula [Aspvall et al., 1979]. Por fim, uma *Cláusula de Horn* é uma cláusula que contém no máximo um literal positivo. O problema *Horn-SAT* é a especialização de SAT cuja entrada é uma fórmula em CNF cujas cláusulas são todas cláusulas de Horn. O problema Horn-SAT também pode ser resolvido em tempo linear no tamanho da fórmula [Dowling e Gallier, 1984].

2.2 Satisfabilidade Booleana Máxima (MaxSAT)

Como apresentado na seção anterior, o problema (de decisão) SAT é um problema em NP-Completo para o qual a eficiência de seus resolvedores no estado-da-arte torna a resolução do problema eficiente na prática para diversas instâncias. Logo, há interesse em reduções de outros problemas de decisão para SAT.

Entretanto, variações do problema SAT podem ser estudadas, de forma que outros problemas, não necessariamente de decisão, podem ser polinomialmente reduzidos para problemas cujos resolvedores utilizam técnicas semelhantes às dos resolvedores SAT.

Em particular, o problema da *Satisfabilidade Booleana Máxima* (MaxSAT) é o problema de otimização que consiste em, dada uma fórmula em CNF, obter uma valoração sobre suas variáveis que satisfaz o maior número de suas cláusulas. Note que a resposta consiste em uma valoração que satisfaz o número máximo possível de cláusulas (isto é, o número de cláusulas da fórmula dada) se e somente se a fórmula dada é satisfatível. Portanto, o problema MaxSAT é NP-Difícil.

O problema de Satisfabilidade Máxima *Ponderada* (WMaxSAT), por sua vez, consiste em, dada uma fórmula F em CNF e dada uma função $w : F \rightarrow \mathbb{N}^+$ que associa um *peso* a cada cláusula da fórmula dada, obter uma valoração cuja soma dos pesos das cláusulas satisfeitas é maximizada.

Por fim, no problema de Satisfabilidade Máxima Ponderada *e Parcial* (PWMaxSAT), são dadas duas fórmulas em CNF, F_h e F_s. A fórmula F_h é dita *hard*, enquanto a fórmula F_s é dita *soft*. As cláusulas de F_h são ditas *cláusulas hard*, enquanto as cláusulas de F_s são ditas *cláusulas soft*. Também é dada uma função $w : F_s \rightarrow \mathbb{N}^+$ que associa um peso a cada cláusula *soft* de F_s. O par $(C, w(C))$ denota uma cláusula *soft* C e seu peso $w(C)$. O problema PWMaxSAT consiste em obter um modelo da fórmula *hard* cuja soma dos pesos das cláusulas *soft* satisfeitas é maximizada.

Assim como problemas de decisão podem ser reduzidos para SAT, problemas de otimização também podem ser reduzidos para PWMaxSAT. O problema pode ser resolvido com uma adaptação do algoritmo DPLL para a técnica *branch and bound* [Heras et al., 2008]. Técnicas sofisticadas específicas para este problema também são utilizadas pelos resolvedores MaxSAT no estado-da-arte, como o cálculo de uma *cota superior inicial* [Selman et al., 1992, Smyth et al., 2003], o de uma *estimativa de cota inferior* [Wallace e Freuder, 1996, Larrosa et al., 2008, Heras et al., 2008], e heurísticas de ramificação específicas [Heras e Larrosa, 2006]. Algoritmos para MaxSAT que não têm o DPLL como base também são conhecidos, como o algoritmo de Fu & Malik [Fu e Malik, 2006], os algoritmos WPM1 e PM2 [Ansótegui et al., 2009], algoritmo por satisfabilidade *por preferências* [Di Rosa et al., 2010], dentre outros. Estas técnicas e algoritmos também são omitidos deste trabalho por não pertencerem ao foco do mesmo.

2.3 Satisfação de Restrições (CSP)

Esta seção apresenta conceitos relacionados ao problema de Satisfação de Restrições (*Constraints Satisfaction Problem* (CSP)). Embora este problema não faça parte do foco principal deste trabalho, seus conceitos são necessários para o entendimento de codificações absolutas apresentadas no capítulo 3, e também para a apresentação do conceito de Arco Consistência Generalizada (ACG) para SAT, apresentado no capítulo 4.

Seja $X = \{X_1, X_2, ..., X_n\}$ um conjunto de variáveis (não necessariamente booleanas) e sejam $D_1, D_2, ..., D_n$ os *domínios* das variáveis em X. O domínio D_i de cada variável X_i é um conjunto finito.

No contexto de CSP, uma *valoração* sobre um conjunto de variáveis $X' \subseteq X$ é uma função $A : X' \to \bigcup_{X_i \in X'} D_i$ onde $A(X_i) \in D_i$ para todo $X_i \in X'$. Desta forma, uma valoração associa a cada variável de X' um elemento de seu domínio. De forma análoga ao problema SAT, uma valoração é dita *completa* se $X' = X$, e *parcial* se $X' \subset X$.

Uma *restrição* R_j é dada por um par $R_j = (X_{R_j}, S_{R_j})$ onde $X_{R_j} \subseteq X$ é um conjunto de variáveis e S_{R_j} é um conjunto de valorações sobre X_{R_j}. O conjunto X_{R_j} é dito *escopo* da restrição R_j.

Uma valoração $A : X' \to \bigcup_{X_i \in X'} D_i$ *satisfaz* uma restrição R_j se $X_{R_j} \subseteq X'$ e existe alguma valoração $A' \in S_{R_j}$ que associa os mesmos valores para as variáveis associados por A, isto é, $A(X_i) = A'(X_i)$ para todo $X_i \in X_{R_j}$, ou $A' \subseteq A$.

O *Problema de Satisfação de Restrições* (CSP) consiste em, dado um conjunto de variáveis X, seus domínios $D_1, D_2, ..., D_n$ e um conjunto de restrições R, decidir se existe uma valoração sobre X que satisfaz todas as restrições em R [Mackworth, 1977, Bessière et al., 2002, Prestwich, 2009].

Como exemplo, considere o conjunto de variáveis $X = \{X_1, X_2, X_3\}$ com domínios $D_1 = \{0, 1\}$, $D_2 = \{a, b, c\}$ e $D_3 = \{2, 4\}$. Considere também as seguintes restrições:

- $R_1 = (\{X_1, X_2\}, \{\{X_1 = 0, X_2 = b\}, \{X_1 = 1, X_2 = a\}\})$;
- $R_2 = (\{X_2, X_3\}, \{\{X_2 = a, X_3 = 2\}, \{X_2 = b, X_3 = 2\}, \{X_2 = c, X_3 = 4\}\})$;
- $R_3 = (\{X_1, X_2, X_3\}, \{\{X_1 = 0, X_2 = b, X_3 = 2\}, \{X_1 = 1, X_2 = c, X_3 = 4\}\})$.

A valoração $A = \{X_1 = 0, X_2 = b, X_3 = 2\}$ satisfaz a restrição R_1 pois ela contém a valoração $\{X_1 = 0, X_2 = b\}$; A satisfaz a restrição R_2 devido à valoração $\{X_2 = b, X_3 = 2\}$; Por fim, A satisfaz R_3 pois A está presente nesta restrição. Assim, todas as restrições são satisfeitas, a instância é positiva e A é uma solução para ela.

Informalmente, as restrições em uma instância enumeram as valorações parciais que a solução pode conter. Alguns autores, como em [Prestwich, 2009], alternativamente definem o problema CSP de tal forma que as restrições enumeram as valorações parciais que a solução *não* pode conter. Nesta definição, uma valoração $A : X' \to \bigcup_{X_i \in X'} D_i$ satisfaz uma restrição $R_j = (X_{R_j}, S_{R_j})$ se $X_{R_j} \subseteq X'$ e *não* existe uma valoração $A' \in S_{R_j}$ tal que $A(X_i) = A'(X_i)$ para todo $X_i \in X_{R_j}$.

Além disso, também é comum representar restrições de maneira mais compacta e informal, dependendo de sua semântica. Como exemplo, a restrição $(\{X_i\}, \{\{X_i = 0\}, \{X_i = 1\}, \{X_i = 3\}, \{X_i = 4\}\})$ com $D_i = \{0, 1, 2, 3, 4\}$, pode ser representada simplesmente com $(X_i \neq 2)$. É válido notar que representações compactas podem reduzir o tamanho de uma instância em ordens de grandeza.

O problema SAT pode ser descrito como uma especialização do problema CSP, onde X é o conjunto de variáveis que ocorrem na fórmula e $D_i = \{0, 1\}$ para toda variável X_i. Além disso, uma cláusula $(l_1 \vee l_2 \vee ... \vee l_k)$ pode ser descrita como a restrição que proíbe a valoração parcial contendo $\neg l_1, \neg l_2, ..., \neg l_k$. Desta forma, o problema CSP, assim como SAT, é NP-Completo.

Uma restrição é dita *$k-$ária* se seu escopo contém k variáveis. Em particular, uma restrição é dita *unária* se seu escopo contém apenas uma variável, e *binária* se seu escopo contém duas variáveis. O problema da *satisfação de restrições binárias* (CSP binário) é a especialização de CSP onde todas as restrições da instância são unárias ou binárias.

Além disso, uma instância de CSP é dita *consistente* se existe uma valoração que satisfaz todas as suas restrições, isto é, se sua resposta para o problema CSP é "sim". Uma restrição unária com escopo $\{X_i\}$ é dita *nodo-consistente* se $\{X_i = v_i\}$ satisfaz a restrição para todo $v_i \in D_i$, isto é, se todos os elementos de D_i podem ser atribuídos à variável X_i de acordo com a restrição [Mackworth, 1977]. Como exemplo, a restrição $R = (\{X_1\}, \{\{X_1 = 0\}, \{X_1 = 1\}\})$, com $D_1 = \{0, 1\}$, é nodo-consistente, pois $\{X_1 = v\}$ satisfaz a restrição para qualquer $v \in D_1$. Entretanto, a restrição $R = (\{X_1\}, \{\{X_1 = 0\}\})$ com o mesmo domínio D_1 não é nodo-consistente, uma vez que $1 \in D_1$ e $\{X_1 = 1\}$ não satisfaz a mesma.

Considere uma restrição binária com escopo $\{X_i, X_j\}$. O *suporte* do valor $v_i \in D_i$ para X_i nesta restrição, denotado por $S_{X_j|X_i=v_i}$, é o conjunto de todos os elementos $v_j \in D_j$ tal que $\{X_i = v_i, X_j = v_j\}$ satisfaz a restrição dada. Desta forma, $S_{X_j|X_i=v_i}$ denota os valores que a variável X_j pode assumir se a variável X_i assumir o valor v_i.

Uma restrição binária com escopo $\{X_i, X_j\}$ é dita *arco-consistente* se, para todo elemento $v_i \in D_i$, existe um elemento $v_j \in D_j$ tal que $\{X_i = v_i, X_j = v_j\}$ satisfaz a restrição. Equivalentemente, uma restrição binária é arco-consistente se $S_{X_j|X_i=v_i} \neq \emptyset$ para todo $v_i \in D_i$ [Mackworth, 1977]. Informalmente, uma restrição binária é arco-consistente se, para cada valor possível de uma variável no seu escopo, existe um valor possível para a outra variável no escopo. Como exemplo, considere $D_1 = D_2 = \{0, 1\}$ e a restrição $R = (\{X_1, X_2\}, \{\{X_1 = 0, X_2 = 1\}, \{X_1 = 1, X_2 = 0\}\})$. Esta restrição é arco-consistente, pois, para cada possível valor de X_1 (0 ou 1), existe um valor para X_2 (1 se $X_1 = 0$ ou 0 se $X_1 = 1$) que satisfaz a restrição, e vice-versa. Neste caso, $S_{X_2|X_1=0} = \{1\}$, $S_{X_2|X_1=1} = \{0\}$, $S_{X_1|X_2=0} = \{1\}$ e $S_{X_1|X_2=1} = \{0\}$.

Uma restrição $k-$ária é *arco-consistente generalizada* (*Generalized Arc-Consistent* (ACG)) se, para toda variável X_i em seu escopo e todo valor $v_i \in D_i$, há uma valoração contendo $X_i = v_i$ que satisfaz a restrição. Informalmente, a restrição é ACG se pode ser satisfeita "mesmo" se o valor v_i for atribuído à variável X_i, para toda variável X_i em seu escopo e todo valor v_i no seu domínio. Note que o conceito de arco-consistência generalizada é uma generalização do conceito de arco-consistência para restrições não necessariamente binárias. Como exemplo, se $D_1 = D_2 = D_3 = \{0, 1\}$, então a restrição $R = (\{X_1, X_2, X_3\}, \{\{X_1 = 0, X_2 = 0, X_3 = 0\}, \{X_1 = 1, X_2 = 0, X_3 = 1\}\})$ não é ACG, uma vez que $1 \in D_2$ e não existe valoração contendo $\{X_2 = 1\}$ que satisfaz R. Entretanto, a restrição $R = (\{X_1, X_2, X_3\}, \{\{X_1 = 0, X_2 = 0, X_3 = 0\}, \{X_1 = 1, X_2 = 0, X_3 = 1\}, \{X_1 = 0, X_2 = 1, X_3 = 1\}\})$ é ACG, uma vez que, para toda variável X_i em seu escopo e, para todo valor v_i no domínio da variável, existe uma valoração que satisfaz a restrição contendo $\{X_i = v_i\}$.

Uma instância de CSP é dita ACG se todas as suas restrições são ACG. Note que uma instância ACG, com mais de uma restrição, não é necessariamente consistente. Como exemplo, a instância $X = \{X_1, X_2\}$, $D_1 = D_2 = \{0, 1\}$ e $R_1 = (X_1 = X_2)$, $R_2 = (X_1 \neq X_2)$ é ACG, mas não é consistente.

O problema CSP pode ser resolvido por *backtracking*, no qual uma variável é decidida e valorada com um elemento de seu domínio a cada passo, de forma análoga ao procedimento DPLL apresentado na seção 2.1. Durante o processo de busca, uma instância ACG pode deixar de ser ACG após a propagação de uma dada valoração. Como exemplo, considere uma instância com $X = \{X_1, X_2, X_3\}$, $D_1 = D_2 = D_3 = \{0, 1\}$ contendo a restrição $R_1 = (X, \{\{X_1 = 0, X_2 = 0, X_3 = 1\}, \{X_1 = 0, X_2 = 1, X_3 = 0\}, \{X_1 = 1, X_2 = 0, X_3 = 0\}\})$. Note que a restrição é ACG. Suponha que o processo de busca decide valorar X_1 com 1. A propagação desta decisão na restrição dada elimina suas duas primeiras valorações parciais, uma vez que seria necessário valorar X_1 com 0 para satisfazê-las. Além disso, a terceira valoração parcial dada na restrição é reduzida. Assim, a instância resultante da propagação da valoração $X_1 = 1$ é dada por $X = \{X_2, X_3\}$,

$D_2 = D_3 = \{0, 1\}$ e contém a restrição $(\{X_2, X_3\}, \{\{X_2 = 0, X_3 = 0\}\})$. Note que esta restrição não é ACG, pois existe um valor no domínio de X_2 (no caso, 1) para o qual não existe um valor de X_3 que satisfaz a restrição.

Entretanto, a nodo- ou arco-consistência generalizada da instância pode ser *mantida* por um procedimento polinomial executado após a propagação de cada valoração decidida.

Primeiramente, se uma dada instância contém uma restrição unária que não é nodo-consistente, é possível mantê-la nodo-consistente apenas removendo, do domínio da variável em seu escopo, os valores que não satisfazem a restrição.

Se uma instância não é ACG, então existe um valor $v_i \in D_i$ tal que não há valoração contendo $X_i = v_i$ que satisfaz as restrições da instância. Assim, para manter a instância ACG, o valor v_i deve ser removido do domínio D_i de X_i. Algoritmos que detectam e removem tais valores para tornar instâncias ACG são conhecidos.

Um algoritmo simples para tal consiste em iterar nas restrições da instância e, para cada uma, determinar sua arco-consistência generalizada percorrendo o domínio das variáveis de seu escopo e verificando a satisfação da restrição assumindo cada valoração. Caso a restrição não seja ACG, os valores dos domínios que não podem satisfazê-la são removidos. O algoritmo itera em todas as restrições até que as mesmas se tornem ACG. Note que uma mesma restrição pode ser visitada pelo algoritmo mais de uma vez.

Outros algoritmos são conhecidos. O algoritmo AC-3, por exemplo, torna ACG instâncias de CSP binário [Mackworth, 1977]. O algoritmo é baseado no algoritmo simples descrito. No AC-3, entretanto, restrições sem relação com as restrições afetadas recentemente não são revisitadas pelo algoritmo [Mackworth, 1977].

Além disso, existem algoritmos que, embora não mantenham a arco-consistência generalizada de uma instância, mantém, informalmente, um "nível", ou uma forma de arco-consistência generalizada mais *"fraca"* da instância. Estes algoritmos, embora não tornem *todas* as restrições ACG, tornam *algumas* restrições ACG. O algoritmo *Forward-Checking* (FC), por exemplo, é análogo ao algoritmo simples descrito, mas visita cada restrição da instância no máximo uma vez [Bessière et al., 2002]. Embora não mantenha a instância *totalmente* arco-consistente generalizada, o algoritmo é mais rápido que o AC-3 e o nível de consistência mantida por ele pode justificar seu uso.

Manter uma restrição ACG, total ou parcialmente, pode reduzir o espaço de busca explorado por um resolvedor. Se uma valoração contendo a associação $\{X_i = v_i\}$ torna uma restrição inconsistente, e se o valor v_i não é removido do domínio da variável X_i, então o resolvedor pode explorar, desnecessariamente, o espaço de busca contendo a associação $\{X_i = v_i\}$. Entretanto, se a manutenção parcial ou total da arco-consistência remover o valor v_i do domínio de X_i, então tal espaço não é explorado, fazendo com que o espaço total de busca explorado pelo resolvedor seja potencialmente menor.

2.4 Considerações finais

Este capítulo apresenta conceitos preliminares sobre os problemas SAT, (PW)MaxSAT e CSP, assim como sobre os algoritmos que os resolvem. Os problemas SAT e (PW)MaxSAT podem ser utilizados para codificar problemas de decisão e de otimização de interesse. Tais reduções são apresentadas no capítulo 3. Além disso, os conceitos apresentados sobre o problema CSP são utilizados, em particular, para apresentar codificações absolutas no capítulo 3, assim como definir o conceito de Arco-Consistência Generalizada (ACG) para os problemas SAT e MaxSAT, como apresentado no capítulo 4.

Capítulo 3

Codificações SAT e MaxSAT Conhecidas

Este capítulo apresenta reduções e codificações conhecidas na literatura de problemas de interesse para SAT e (PW)MaxSAT. As codificações apresentadas são classificadas em três categorias: codificações *absolutas*, *relativas* e de *operadores de cardinalidade*. Codificações absolutas e de operadores de cardinalidade são apresentadas brevemente, enquanto codificações relativas, foco principal deste trabalho, são apresentadas detalhadamente.

3.1 Codificações absolutas

Codificações *absolutas* são reduções do problema CSP, apresentado no capítulo 2, para o problema SAT. Desta forma, é possível codificar problemas de decisão em SAT reduzindo-os para o problema CSP, e então codificando a instância obtida para SAT através de uma codificação absoluta.

Esta seção apresenta reduções conhecidas de CSP para SAT, isto é, métodos de construção de uma fórmula booleana satisfatível se e somente se a dada instância de CSP é consistente. Além disso, um modelo da fórmula booleana construída deve descrever uma valoração das variáveis que satisfaz todas as restrições no problema CSP original.

Assim como definido inicialmente por Prestwich [Prestwich, 2003], codificações de instâncias de CSP em SAT são ditas "absolutas" neste texto porque a cada variável da instância dada é associado um valor *exato* ("absoluto") presente em seu domínio. O termo também é utilizado por Prestwich, Velev & Gao [Prestwich, 2003, Velev e Gao, 2009, Velev e Gao, 2010] para diferenciar estas reduções de codificações *relativas*, apresentadas na seção 3.2.

Codificações absolutas podem ser classificadas de acordo com o significado da proposição representada por cada variável booleana criada. A principal codificação absoluta utilizada é a dita *direta*, descrita a seguir.

Seja $X = \{X_1, ..., X_n\}$, $D_1, ..., D_n$ e R uma instância de CSP. Uma maneira natural de codificar a instância em SAT é criar, para cada variável $X_i \in X$ e para cada valor $v_j \in D_i$ de seu domínio, uma variável booleana $x_{i,j}$. A variável $x_{i,j}$ assume o valor 1 (verdadeiro) em um modelo da fórmula construída se e somente se, na valoração do problema CSP original, a variável X_i é associada ao valor v_j de seu domínio. Assim, $\sum_{1 \leq i \leq n} |D_i|$ variáveis são criadas nesta codificação. Em particular, se todos os domínios têm o mesmo tamanho m, então $n \times m$ variáveis são criadas.

Em um modelo da fórmula construída, o valor associado a uma variável X_i pode ser obtido verificando-se para qual valor $v_j \in D_i$ a variável $x_{i,j}$ é verdadeira. Esta codificação é dita *direta* pois cada variável booleana codifica a associação direta de uma variável a um valor de seu domínio. A codificação foi inicialmente descrita por Kleer [De Kleer, 1989].

Uma vez definido o conjunto de variáveis booleanas, é necessário descrever as cláusulas que compõe a fórmula construída. Para tal, três condições devem ser garantidas: (i) toda variável $X_i \in X$ assume pelo menos um valor de seu domínio; (ii) toda variável $X_i \in X$ assume no máximo um valor de seu domínio; e (iii) toda restrição da instância é satisfeita.

As duas primeiras condições garantem que toda variável tem *exatamente* um valor de seu domínio associado a ela. Ambas são garantidas com dois conjuntos de cláusulas, denominadas cláusulas *at-least* e cláusulas *at-most* [Velev e Gao, 2009].

As cláusulas *at-least* garantem que toda variável tem *pelo menos* um valor de seu domínio. Para cada variável $X_i \in X$, a seguinte cláusula é criada:

- (*at-least*) $\bigvee\limits_{v_j \in D_i} x_{i,j}$

Desta forma, n cláusulas são criadas, de tamanho total $\sum\limits_{1 \leq i \leq n} |D_i|$.

As cláusulas *at-most* garantem que toda variável tem *no máximo* um valor de seu domínio. Para cada variável $X_i \in X$ e para cada par de valores de seu domínio $v_j, v_k \in D_i$, a seguinte cláusula é adicionada:

- (*at-most*) $\neg x_{i,j} \vee \neg x_{i,k}$

A cláusula indica que, se X_i assume um valor v_j, então X_i não pode assumir um outro valor v_k do seu domínio. São criadas $\sum\limits_{1 \leq i \leq n} \binom{|D_i|}{2} = O(n \times max_{1 \leq i \leq n}\{|D_i|\}^2)$ cláusulas binárias.

Por fim, é necessário garantir que toda restrição da instância é satisfeita. Há duas maneiras conhecidas para codificar as restrições do problema: a codificação absoluta direta *por conflito* e a *por suporte*.

Na codificação absoluta direta por conflito, todas as valorações parciais que *não* satisfazem a restrição dada são enumeradas e negadas [De Kleer, 1989].

Seja $R_j = (X_{R_j}, S_{R_j})$ uma restrição da instância dada, e seja $A' : X_{R_j} \rightarrow \bigcup\limits_{X_i \in X_{R_j}} D_i$ uma valoração parcial que *não* satisfaz R_j. O fato de que A' não pode estar contido na solução buscada pode ser codificado com a seguinte cláusula:

- $\neg(\bigwedge\limits_{X_i \in X_{R_j}} x_{i,A'(X_i)}) = \bigvee\limits_{X_i \in X_{R_j}} \neg x_{i,A'(X_i)}$

Como exemplo, se $\{X_1 = 0, X_2 = 3, X_3 = 7\}$ não satisfaz uma dada restrição, então a cláusula $\neg(x_{1,0} \wedge x_{2,3} \wedge x_{3,7}) = (\neg x_{1,0} \vee \neg x_{2,3} \vee \neg x_{3,7})$ é criada.

O tamanho de cada cláusula é igual ao do escopo da restrição. O número de cláusulas, por sua vez, é igual ao número de valorações que não satisfazem a restrição. Embora este número seja limitado por $\prod\limits_{X_i \in X_{R_j}} |D_i| = O(max_{1 \leq i \leq n}\{|D_i|\}^{|X_{R_j}|})$, o tamanho da fórmula pode ser viável na prática, dependendo, informalmente, de quão "permissiva" ou "proibitiva" a restrição é. Note também que, se a restrição é descrita como uma coleção de valorações parciais proibidas como definido por alguns autores, então o número de cláusulas é linear no tamanho da instância. Além disso, se todas as restrições são binárias, então o número de cláusulas, para cada restrição, é limitado pelo polinômio $max_{1 \leq i \leq n}\{|D_i|\}^2$.

Também é válido observar que, se as restrições da instância CSP dada são codificadas por conflito, então as cláusulas do tipo *at-most* da codificação absoluta podem ser omitidas. A codificação absoluta direta por conflito sem a inclusão das cláusulas *at-most* também é chamada de *codificação multidireta* [Velev, 2007, Velev e Gao, 2009].

Além da codificação por conflito, restrições *binárias* de CSP podem ser codificadas por *suporte*. Enquanto a codificação por conflito enumera valorações proibidas, a codificação por suporte enumera valorações permitidas [Kasif, 1990, Gent, 2002].

Seja $R_j = (\{X_i, X_j\}, S_{R_j})$ uma restrição binária. Para cada valor $v_a \in D_i$, a restrição é satisfeita com $X_i = v_a$ apenas se X_j assumir algum valor no suporte $S_{X_j|X_i=v_a}$. Assim, para cada $v_a \in D_i$, a seguinte cláusula é criada:

- $x_{i,a} \rightarrow (\bigvee_{v_b \in S_{X_j|X_i=v_a}} x_{j,b}) = \neg x_{i,a} \vee (\bigvee_{v_b \in S_{X_j|X_i=v_a}} x_{j,b})$.

Além disso, para cada $v_b \in D_j$, a seguinte cláusula é criada:

- $x_{j,b} \rightarrow (\bigvee_{v_a \in S_{X_i|X_j=v_b}} x_{i,a}) = \neg x_{j,b} \vee (\bigvee_{v_a \in S_{X_i|X_j=v_b}} x_{i,a})$.

O número de cláusulas para cada restrição é igual a $|D_i| + |D_j| = O(max_{1 \leq i \leq n}\{|D_i|\})$. Cada cláusula tem tamanho idêntico ao dos suportes citados, também limitado a $O(max_{1 \leq i \leq n}\{|D_i|\})$. Observa-se que a codificação por suporte gera cláusulas até do mesmo tamanho do domínio de uma variável, que pode ser considerado grande de acordo com o problema modelado. A codificação por conflito, por sua vez, sempre gera cláusulas binárias para restrições binárias.

Também é válido notar que, se $S_{X_j|X_i=v_a}$ é um conjunto vazio, então a codificação por suporte cria a cláusula unitária ($\neg x_{i,a}$), fazendo com que a propagação unitária valore a variável $x_{i,a}$ com 0 (falso) imediatamente. Esta propriedade é importante por manter um nível de consistência da fórmula gerada.

Além da codificação absoluta direta, pode ser citada a codificação absoluta *logarítmica*, na qual uma variável booleana $x_{i,j}$ é criada para cada variável X_i da instância e cada $0 \leq j < \lceil \lg |D_i| \rceil$, indicando o $j-$ésimo *bit* da representação binária do valor associado à variável X_i [Iwama e Miyazaki, 1994, Prestwich, 2009, Gelder, 2008]. Esta codificação cria menos variáveis que a direta, embora suas propriedades de consistência podem ser ineficientes.

Também pode ser citada a codificação absoluta *unária*, na qual uma variável booleana $x_{i,k}$ é criada para cada variável X_i e cada $0 \leq k < |D_i|$ indicando a relação $X_i \geq k$. Esta codificação é comumente utilizada para representar *intervalos* e, por isso, é utilizada para codificar operadores de cardinalidade [Bailleux e Boufkhad, 2003, Björk, 2009, Sinz, 2005, Samer e Veith, 2009], como apresentado na seção 3.3. Por fim, também podem ser citadas as codificações *estrutural* e *híbrida*, onde modelos da fórmula criada descrevem caminhos em árvores que, por sua vez, descrevem a solução encontrada [Velev, 2007, Klieber e Kwon, 2007]. A descrição detalhada destas codificações foi omitida deste texto por não pertencer ao seu foco.

Como exemplos de codificações absolutas, podem ser citadas as reduções de Prestwich [Prestwich, 2003], de Hertel et al. [Hertel et al., 2007] e de Iwaza & Miyazaki [Iwama e Miyazaki, 1994] para o problema do Ciclo Hamiltoniano; as codificações de Gelder [Gelder, 2008] para o problema da $k-$Coloração; e as codificações de conectividade de Kautz et al. [Kautz et al., 1996] e Oliveira & Silva [Oliveira e Silva, 2014], aplicadas a reduções do problema da Árvore de Steiner para (PW)MaxSAT.

3.2 Codificações Relativas

Enquanto codificações absolutas criam variáveis booleanas que codificam o valor exato que uma variável pode assumir, codificações *relativas* criam variáveis que codificam a relação (binária) entre elementos de um dado conjunto. Codificações relativas podem ser usadas para

reduzir vários problemas computacionais, principalmente os relacionados a grafos, para SAT e (PW)MaxSAT.

Esta seção apresenta as codificações relativas conhecidas, foco deste trabalho. A primeira subseção apresenta um *framework* para definir codificações relativas, enquanto a segunda apresenta as reduções conhecidas para SAT e (PW)MaxSAT que utilizam codificações relativas.

3.2.1 *Framework* de Codificações Relativas

Primeiramente, uma codificação *relativa* é uma codificação que descreve restrições sobre uma relação binária $R \subseteq S \times S$ sobre os elementos de um dado conjunto finito S. Desta forma, um modelo para uma fórmula criada por uma codificação relativa descreve uma relação binária $R \subseteq S \times S$ de acordo com as restrições sendo codificadas.

Como um exemplo, considere $S = \{a, b, c\}$. Se a relação codificada deve ser (apenas) transitiva, a fórmula resultante conterá cláusulas tais que toda relação binária transitiva sobre os elementos de S será descrita por um dos modelos da fórmula. Desta forma, a relação $R = \{(a, b), (b, c), (a, c)\}$, por exemplo, será descrita por um de seus modelos. Da mesma forma, outro modelo da mesma fórmula descreverá a relação $R = \{(a, b), (a, c)\}$. Entretanto, nenhum modelo da fórmula descreverá a relação $R = \{(a, b), (b, c)\}$, dado que R não é transitiva.

Em codificações relativas, uma variável booleana $r_{a,b}$ é criada para cada par ordenado de elementos $a, b \in S$. A variável $r_{a,b}$ assume 1 (verdadeiro) em um modelo da fórmula criada se $(a, b) \in R$, isto é, se os elementos a e b (nesta ordem) estão relacionados pela relação codificada. Nesta codificação, exatamente $|S|^2$ variáveis booleanas são criadas.

De acordo com o problema reduzido, a relação codificada deve possuir propriedades específicas. Cada propriedade é garantida com a adição de um conjunto de cláusulas à fórmula.

Para garantir a propriedade de antireflexividade, a seguinte cláusula unitária é adicionada, para cada $a \in S$:

- (*antireflexividade*) $(\neg r_{a,a})$

A cláusula $(\neg r_{a,a})$ indica que o elemento a não deve se relacionar consigo mesmo. Assim, $|S|$ cláusulas unitárias são criadas.

A propriedade de *reflexividade*, por sua vez, pode ser garantida de forma análoga com a seguinte cláusula unitária, para cada $a \in S$:

- (*reflexividade*) $(r_{a,a})$

A propriedade de antissimetria, por sua vez, é garantida com a adição da seguinte cláusula, para cada $\{a, b\} \in \binom{S}{2}$:

- (*antissimetria*) $(r_{a,b} \rightarrow \neg r_{b,a}) = (\neg r_{a,b} \vee \neg r_{b,a})$

Esta cláusula indica que, se a está relacionado com b na relação descrita, então b não deve estar relacionado com a na mesma. Desta forma, $\binom{|S|}{2} < \frac{|S|^2}{2}$ cláusulas binárias são criadas.

De forma análoga, para codificar a propriedade de simetria, basta adicionar a seguinte cláusula, para todo $a, b \in S, a \neq b$:

- (*simetria*) $(r_{a,b} \rightarrow r_{b,a}) = (\neg r_{a,b} \vee r_{b,a})$

A propriedade de transitividade pode ser codificada com a seguinte cláusula, para cada $a, b, c \in S$, $a \neq b \neq c$:

- (*transitividade*) $((r_{a,b} \wedge r_{b,c}) \rightarrow r_{a,c}) = (\neg r_{a,b} \vee \neg r_{b,c} \vee r_{a,c})$

Esta cláusula indica que, se a se relaciona com b e b se relaciona com c, então a deve se relacionar com c na relação descrita. Desta forma, existem $\Theta(|S|^3)$ cláusulas de transitividade com três literais cada.

Por fim, se a relação codificada deve ser total, a seguinte cláusula é adicionada à fórmula, para cada $a, b \in S$, $a \neq b$:

- (*totalidade*) $(r_{a,b} \vee r_{b,a})$

Desta forma, $\binom{|S|}{2} < \frac{|S|^2}{2}$ cláusulas binárias também são criadas.

Além de propriedades sobre a relação R codificada, também pode ser necessário garantir propriedades sobre o *fecho transitivo* R^+ da relação R, como nas codificações apresentadas na subseção 3.2.2.

Para codificar o fecho transitivo da relação codificada, uma nova variável booleana $r^+_{a,b}$ é criada, para cada $a, b \in S$. A variável $r^+_{a,b}$ assume 1 (verdadeiro) em um modelo da fórmula criada se $(a, b) \in R^+$, isto é, se os elementos a e b (nesta ordem) estão relacionados pelo fecho transitivo da relação R. Desta forma, são criadas mais $|S|^2$ variáveis booleanas. Ao todo, $2|S|^2$ variáveis são criadas em codificações que descrevem ambas as relações.

Para garantir que R^+ é, de fato, o fecho transitivo da relação R, é necessário primeiramente indicar que $R \subseteq R^+$, isto é, a relação R^+ contém a relação R. Esta condição é codificada com a seguinte cláusula, para cada $a, b \in S$:

- (*inclusão de conjunto*) $(r_{a,b} \rightarrow r^+_{a,b} = \neg r_{a,b} \vee r^+_{a,b})$

Desta forma, $|S|^2$ cláusulas binárias são criadas. Além disso, para garantir que R^+ é transitiva, basta utilizar as até $|S|^3$ cláusulas de transitividade:

- (transitividade) $((r^+_{a,b} \wedge r^+_{b,c}) \rightarrow r^+_{a,c}) = (\neg r^+_{a,b} \vee \neg r^+_{b,c} \vee r^+_{a,c})$

É possível notar que, se ambas as relações R e seu fecho transitivo R^+ são codificadas, é suficiente criar cláusulas de *transitividade*, *(anti)reflexividade*, *(anti)simetria* e de *totalidade* apenas para a relação R^+.

Além de cláusulas que codificam propriedades da relação binária codificada, codificações relativas também podem conter cláusulas que descrevem restrições sobre o problema sendo codificado. Dentre estas cláusulas, destacam-se cláusulas *at-least* e *at-most*, análogas às clausulas contidas em codificações absolutas apresentadas na seção 3.1.

Seja $a \in S$. Dado um conjunto $L_a \subseteq S$, a seguinte cláusula codifica a restrição de que o elemento a deve estar relacionado a pelo menos um elemento de L_a:

- (*at-least*) $(\bigvee_{l \in L_a} r_{a,l})$

Além disso, dado um conjunto $M_a \subseteq S$, as seguintes cláusulas codificam a restrição de que o elemento a deve estar relacionado a no máximo um elemento de M_a:

- (*at-most*) $\bigwedge_{m,n \in M_a} (\neg r_{a,m} \vee \neg r_{a,n})$

Note que, se $M_a = L_a$, ambas as categorias de cláusulas indicam, simultaneamente, que o elemento a deve estar relacionado a exatamente um elemento de um dado conjunto de elementos. Além disso, cláusulas análogas podem ser criadas para codificar a restrição de que pelo menos um, no máximo um ou exatamente um elemento de um dado conjunto deve estar relacionado a a. São criadas uma cláusula *at-least* de tamanho $|L_a| \leq |S|$, e $\binom{|M_a|}{2} \leq \frac{|S|^2}{2}$ cláusulas binárias *at-most*.

Por fim, se a relação codificada é uma ordem parcial ou total, é possível codificar a restrição de que um dado elemento $a \in S$ é um elemento mínimo ou máximo da relação com as seguintes cláusulas unitárias, para todo $b \in S \setminus \{a\}$:

- (*mínimo*) $(r^+_{a,b})$
- (*máximo*) $(r^+_{b,a})$

Como apresentado na subseção 3.2.2, codificações relativas conhecidas podem ser descritas em termos das cláusulas apresentadas nesta subseção.

Entretanto, caso as relações descritas tenham propriedades específicas, então algumas variáveis booleanas podem não ser definidas pela codificação relativa. Velev & Gao [Velev e Gao, 2010] observam que, se a relação R^+ é antissimétrica e total, então $r^+_{b,a}$ assume o valor oposto de $r^+_{a,b}$ em todo modelo da fórmula construída, para todo $a, b \in S$. Desta forma, apenas a variável $r^+_{a,b}$ necessita ser de fato definida, uma vez que as ocorrências da variável $r^+_{b,a}$ na fórmula podem ser *substituídas* pelo literal $\neg r^+_{a,b}$. Substituição semelhante pode ser realizada caso a relação R^+ seja simétrica: a variável $r^+_{b,a}$ pode ser substituída pela variável $r^+_{a,b}$. Nestes casos, o número de variáveis booleanas necessárias para codificar a relação R^+ é reduzido pela metade. Além disso, as cláusulas que codificam a relação de (anti)simetria deixam de existir, e, por fim, o número de cláusulas necessárias para codificar a propriedade de transitividade é reduzido em 2/3 [Velev e Gao, 2010].

De maneira análoga, a variável $r^+_{a,a}$, para todo $a \in S$, pode ser substituída pelo valor verdade 0 (resp. 1) se a relação R^+ for antireflexiva (resp. reflexiva). Além disso, se uma restrição específica do problema reduzido indica que um dado elemento a não pode se relacionar com um dado elemento b pela relação R, então, ao invés de criar a cláusula unitária $(\neg r_{a,b})$, a variável $r_{a,b}$ pode deixar de ser definida, e suas ocorrências podem ser substituídas pelo valor 0 (falso) na fórmula.

Assim, embora o número de variáveis booleanas criadas seja limitado por $2|S|^2$, o número total de variáveis criadas pode ser menor, de acordo com as propriedades da relação codificada.

Além disso, é válido notar que, em uma fórmula que codifica relações binárias entre elementos de um dado conjunto S que não contém cláusulas *at-least*, todas as cláusulas têm no máximo dois literais cada, exceto pelas cláusulas que codificam a propriedade de transitividade, que têm três literais cada. Além de serem as maiores que ocorrem na fórmula, estas cláusulas são as que mais ocorrem na fórmula, com $\Theta(|S|^3)$ ocorrências ao todo.

Entretanto, se a relação R^+ é simétrica ou antissimétrica, então, além da variável $r^+_{b,a}$, a variável $r^+_{a,b}$ pode *também não ser definida* na codificação, para *alguns* valores de $a, b \in S$, de acordo com o problema reduzido e a instância codificada [Bryant e Velev, 2002, Velev e Gao, 2010]. Informalmente, isto ocorre quando a relação entre os elementos a e b no fecho transitivo não é relevante para determinar a solução da instância dada.

Considere primeiramente que R^+ é simétrica. Seja $\mathfrak{R}^+ \subseteq \{r^+_{a,b} | (a, b) \in S \times S\}$ o conjunto de variáveis sobre o fecho transitivo da relação que são definidas na codificação. Este

conjunto pode ser formado, por exemplo, pelo conjunto de variáveis que ocorrem não *apenas* nas cláusulas de transitividade originalmente, mas *também* nas demais cláusulas da fórmula.

Nota-se que $\mathfrak{R}^+$ não contém ambas as variáveis $r^+_{a,b}$ e $r^+_{b,a}$ para todo par $a, b \in S$, dado que R^+ é simétrica e, portanto, uma das variáveis pode ser omitida.

Seja $G^+ = (S, E^+)$ o grafo no qual cada vértice é um elemento de S, e $E^+ = \{\{a, b\} \in \binom{S}{2} | r^+_{a,b} \in \mathfrak{R}^+\}$, isto é, há uma aresta em G^+ para cada par de elementos a, b tal que a variável $r^+_{a,b}$ é definida.

Considere uma valoração A sobre $\mathfrak{R}^+$. Segundo Bryant & Velev [Bryant e Velev, 2002], a restrição de transitividade é satisfeita por A se e somente se G^+ *não* contém um *ciclo sem cordas* $(a_0, a_1, ..., a_{k-1}, a_0)$ contendo *exatamente* uma aresta $\{a_i, a_{i+1}\}$ tal que $A(r^+_{a_i,a_{i+1}}) = 0$, com as demais arestas $\{a_j, a_{(j+1)mod k}\}$ com $A(r^+_{a_j,a_{(j+1)mod k}}) = 1$.

Como exemplo, considere que G^+ possui um ciclo sem cordas (a, b, c, d, a). Se as variáveis $r^+_{a,b}$, $r^+_{b,c}$, $r^+_{c,d}$, e $r^+_{a,d}$ (que substitui a variável $r^+_{d,a}$, dado que R é simétrica) são valoradas com 1 (verdadeiro), então a propriedade de transitividade é garantida, pois $(a, b) \in R^+$, $(b, c) \in R^+$ e $(c, d) \in R^+$, e também $(a, d) \in R^+$. Entretanto, se a variável $r^+_{b,c}$, por exemplo, for valorada com 0 (falso), então a transitividade não é satisfeita, pois $(c, d) \in R^+$, $(a, d) \in R^+$ (e, por simetria, $(d, a) \in R^+$), e $(a, b) \in R^+$, mas $(b, c) \notin R^+$ (e, por simetria, $(c, b) \notin R^+$).

Assim, é possível identificar todos os ciclos sem cordas do grafo G^+ para determinar quais cláusulas devem ser adicionadas à fórmula para codificar a propriedade de transitividade. Seja $(a_0, a_1, ..., a_{k-1}, a_0)$ um ciclo sem cordas de G^+ de tamanho k. As seguintes k cláusulas são adicionas à fórmula:

- $(\bigwedge_{0 \le i < k, i \neq 0} r^+_{a_i,a_{(i+1)mod k}}) \rightarrow r^+_{a_0,a_1} = \bigvee_{0 \le i < k, i \neq 0} \neg r^+_{a_i,a_{(i+1)mod k}} \vee r^+_{a_0,a_1}$
- $(\bigwedge_{0 \le i < k, i \neq 1} r^+_{a_i,a_{(i+1)mod k}}) \rightarrow r^+_{a_1,a_2} = \bigvee_{0 \le i < k, i \neq 1} \neg r^+_{a_i,a_{(i+1)mod k}} \vee r^+_{a_1,a_2}$
- ...
- $(\bigwedge_{0 \le i < k, i \neq k-1} r^+_{a_i,a_{(i+1)mod k}}) \rightarrow r^+_{a_{k-1},a_0} = \bigvee_{0 \le i < k, i \neq k-1} \neg r^+_{a_i,a_{(i+1)mod k}} \vee r^+_{a_{k-1},a_0}$

Esta codificação gera k cláusulas para cada ciclo sem cordas de G^+ de tamanho k. Além disso, cada cláusula criada tem k literais. Teoricamente, o menor valor que k pode assumir é $k = 3$, o tamanho do menor ciclo que pode existir em um grafo. Assim, esta codificação gera o menor número possível de cláusulas de tamanho mínimo quando todos os ciclos sem cordas de G^+ têm tamanho 3. Isto ocorre quando G^+ é um grafo *cordal*.

Se G^+ não é um grafo cordal, então um conjunto de arestas pode ser adicionado a G^+ de forma a torná-lo cordal. Determinar um conjunto mínimo de arestas a ser adicionado é NP-Difícil [Yannakakis, 1981]. Entretanto, algumas heurísticas podem encontrar um conjunto de tamanho satisfatório. Bryant, Velev & Gao sugerem heurísticas que utilizam arestas obtidas de um procedimento polinomial de eliminação sucessiva de vértices [Bryant e Velev, 2002, Velev e Gao, 2010].

Por fim, se R^+ é uma relação antissimétrica, então a propriedade de transitividade pode ser codificada de maneira similar. Além disso, se a relação é total e o grafo G^+ é (ou foi transformado em) cordal, então, para cada ciclo (a, b, c, a) de tamanho 3 de G^+, as duas seguintes cláusulas podem substituir as três cláusulas que codificam a propriedade de transitividade [Velev e Gao, 2010]:

- $(r^+_{a,b} \wedge r^+_{b,c}) \rightarrow r^+_{a,c} = \neg r^+_{a,b} \vee \neg r^+_{b,c} \vee r^+_{a,c}$

- $(\neg r^+_{a,b} \wedge \neg r^+_{b,c}) \rightarrow \neg r^+_{a,c} = r^+_{a,b} \vee r^+_{b,c} \vee \neg r^+_{a,c}$

Desta forma, a propriedade de transitividade pode ser codificada de maneira mais compacta do que a apresentada anteriormente, quando algumas variáveis sobre o fecho transitivo da relação descrita são omitidas.

3.2.2 Codificações Relativas Conhecidas

Reduções conhecidas de problemas de interesse para SAT ou (PW)MaxSAT podem ser descritas como codificações relativas que utilizam o *framework* apresentado na subseção anterior.

A primeira codificação relativa conhecida é a codificação relativa de Prestwich [Prestwich, 2003] utilizada para reduzir o problema do Ciclo Hamiltoniano para SAT. A principal ideia da codificação é representar o ciclo buscado como uma relação de ordem total (isto é, uma permutação) dos vértices do grafo dado.

Seja $G = (V(G), E(G))$ o grafo dado como instância do problema do Ciclo Hamiltoniano. Para cada $v_i, v_j \in V(G)$, as variáveis booleanas $s_{i,j}$, $s_{j,i}$, $o_{i,j}$ e $o_{j,i}$ são criadas. A variável $s_{i,j}$ (resp. $s_{j,i}$) assume 1 (verdadeiro) no modelo da fórmula construída se o vértice v_j (resp. v_i) é o *sucessor imediato* do vértice v_i na permutação codificada. A variável $o_{i,j}$ (resp. $o_{j,i}$), por sua vez, assume este valor se o vértice v_i (resp. v_j) ocorre *antes* (não necessariamente imediatamente) do vértice v_j (resp. v_i) na permutação. Note que a relação codificada pelas variáveis $o_{i,j}$ é o fecho transitivo R^+ da relação R codificada pelas variáveis $s_{i,j}$ e, logo, as variáveis $s_{i,j}$ (resp. $o_{i,j}$) são análogas às variáveis $r_{a,b}$ (resp. $r^+_{a,b}$) apresentadas na subseção anterior.

Na codificação de Prestwich [Prestwich, 2003], assume-se que um vértice arbitrário $v_0 \in V(G)$ deve ocupar a primeira posição da permutação codificada (isto é, deve ser um elemento mínimo da relação codificada). Note que isto não invalida a corretude da codificação, uma vez que toda permutação que representa um ciclo pode ser rotacionada de tal forma que v_0 ocupe sua primeira posição.

As seguintes cláusulas são então adicionadas à fórmula:

- $(\neg s_{i,j})$ e $(\neg s_{j,i})$ para todo $\{v_i, v_j\} \in E(\bar{G})$, onde $\bar{G}$ é o complemento do grafo G;
- (*at-least*) $(\bigvee_{v_j \in V(G)} s_{i,j})$, para todo $v_i \in V(G)$. Esta cláusula codifica a condição de que todo vértice deve ter ao menos um sucessor na permutação;
- (*at-most*) $(\neg s_{i,j} \vee \neg s_{i,k})$ para todo $v_i, v_j, v_k \in V(G)$ tal que $\{v_i, v_j\}, \{v_i, v_k\} \in E(G)$. Estas cláusulas codificam a condição de que nenhum vértice possui mais de um sucessor na permutação;
- (*at-least*) $(\bigvee_{v_j \in V(G)} s_{j,i})$, para todo $v_i \in V(G)$. Esta cláusula codifica a condição de que todo vértice deve ter ao menos um antecessor na permutação;
- (*at-most*) $(\neg s_{j,i} \vee \neg s_{k,i})$ para todo $v_i, v_j, v_k \in V(G)$ tal que $\{v_i, v_j\}, \{v_i, v_k\} \in E(G)$. Estas cláusulas codificam a condição de que nenhum vértice possui mais de um antecessor na permutação;
- (*inclusão de conjunto*) $(s_{i,j} \rightarrow o_{i,j}) = (\neg s_{i,j} \vee o_{i,j})$ para todo $v_i, v_j \in V(G)$, $v_j \neq v_0$;
- (*antireflexividade*) $\neg o_{i,i}$ para todo $v_i \in V(G)$;
- (*antissimetria*) $(o_{i,j} \rightarrow \neg o_{j,i}) = (\neg o_{i,j} \vee \neg o_{j,i})$, para todo $v_i, v_j \in V(G)$;

- (*transitividade*) $(o_{i,j} \wedge o_{j,k}) \rightarrow o_{i,k} = (\neg o_{i,j} \vee \neg o_{j,k} \vee o_{i,k})$, para cada tripla de vértices distintos $v_i, v_j, v_k \in V(G)$;

- (*mínimo*) $(o_{0,i})$ para todo vértice $v_i \in V(G)$, $v_i \neq v_0$. Esta cláusula codifica a condição de que v_0 deve ocorrer antes de todos os demais vértices na permutação;

- (*máximo*) $(s_{l,0} \rightarrow o_{i,l}) = (\neg s_{l,0} \vee o_{i,l})$ para cada $v_l \in V(G), v_l \neq v_0$ e para cada $v_i \in V(G), v_i \neq v_l$. Estas cláusulas indicam que, se v_l é o antecessor de v_0 na permutação, então v_l é o último vértice na mesma, e portanto deve ocorrer depois dos demais vértices na permutação.

Com as cláusulas descritas acima, a codificação relativa de Prestwich [Prestwich, 2003] é uma redução correta do problema do Ciclo Hamiltoniano para SAT, que cria $O(|V(G)|^2)$ variáveis, $O(|V(G)|^3)$ cláusulas e uma fórmula de tamanho $O(|V(G)|^3)$. Entretanto, Velev & Gao [Velev e Gao, 2010] sugerem alguns aprimoramentos na codificação relativa de Prestwich. Primeiramente, a variável $s_{i,j}$ é definida apenas se $\{v_i, v_j\} \in E(G)$. As variáveis $s_{i,i}$ e $o_{i,i}$, para todo $v_i \in V(G)$, também podem ser substituídas pelo valor verdade 0 (falso). Além disso, como a relação de ordem total codificada é antissimétrica, então as ocorrências dos literais $o_{j,i}$ podem ser substituídas pelo literal $\neg o_{i,j}$. Esta substituição reduz pela metade o número de variáveis deste tipo.

Também é sugerido o método de codificação de transitividade apresentado na subseção anterior. Para a codificação relativa do problema do Ciclo Hamiltoniano, Velev & Gao [Velev e Gao, 2010] mostram que o conjunto de variáveis $\mathfrak{R}^+$ definidas pode ser dado por $\mathfrak{R}^+ = \{o_{i,j} | \{v_i, v_j\} \in E(G)\} \cup \{o_{0,i} | v_i \in V(G), v_i \neq v_0\} \cup \{o_{i,j} | \{v_0, v_i\} \in E(G), v_j \in V(G), v_j \neq v_0, v_i\}$. Desta forma, uma variável do tipo $o_{i,j}$ é definida (i) para cada aresta $\{v_i, v_j\} \in E(G)$; (ii) para cada par de vértices formado pelo vértice v_0 e um outro vértice do grafo; e (iii) para cada par de vértices formado por um vizinho de v_0 e um outro vértice do grafo [Velev e Gao, 2010]. Com o conjunto $\mathfrak{R}^+$ definido, é possível novamente construir o grafo G^+, torná-lo cordal e codificar a condição de transitividade como apresentado na subseção anterior.

Além desta codificação, podem ser citadas as codificações para árvores de Oliveira & Silva [Oliveira e Silva, 2014]. Dado um grafo $G = (V(G), E(G))$ e um conjunto de vértices $T \subseteq V(G)$, é criada uma fórmula satisfatível se e somente se existe um caminho entre cada par de vértices em T no grafo G, isto é, se todos os vértices de T pertencem à mesma componente conexa do grafo dado. Além disso, cada modelo desta fórmula descreve um subgrafo conexo contendo os vértices de T. É válido notar que, se $T = V(G)$, então a fórmula criada é satisfatível se e somente se G é conexo. Embora determinar tal conectividade é um problema em P, a fórmula obtida é utilizada como fórmula *hard* para reduzir o problema NP-Difícil da Árvore de Steiner para PWMaxSAT [Oliveira e Silva, 2014].

Seja $T = \{v_{t_1}, v_{t_2}, ..., v_{t_{|T|}}\}$ e sejam $G_1, G_2, ..., G_{|T|-1}$ grafos isomorfos a G, com $G_j = (V(G)_j, E(G)_j)$, com $V(G)_j = \{v_{i,j} | v_i \in V(G)\}$ e $E(G)_j = \{\{v_{a,j}, v_{b,j}\} | \{v_a, v_b\} \in E(G)\}$. Informalmente, o grafo G_j $(1 \leq j < |T|)$, isomorfo ao grafo G, é uma "réplica" de G com vértices indexados com o inteiro j.

A codificação relativa *por caminhos* descreve a componente conexa buscada como a união de $|T| - 1$ caminhos [Oliveira e Silva, 2014]. Cada caminho descreve uma sequência de vértices $P_j = (v_{t_i,j}, ..., v_{t_{i+1},j})$, de $v_{t_i,j}$ para $v_{t_{i+1},j}$, no grafo G_j, para $1 \leq i < |T|$. A componente é então dada pelo conjunto de arestas $\{\{v_a, v_b\} \in E(G) | \{v_{a,j}, v_{b,j} \in P_j$, para algum $1 \leq j < |T|\}$.

Cada caminho é então codificado com uma codificação relativa semelhante à codificação de Prestwich [Prestwich, 2003] para o problema do Ciclo Hamiltoniano. Para cada $1 \leq j < |T|$ e para cada aresta $\{v_{i,j}, v_{k,j}\} \in E(G)_j$, duas variáveis booleanas $s_{i,k,j}$ e $s_{k,i,j}$ são criadas. A variável

$s_{i,k,j}$ (resp. $s_{k,i,j}$) assume 1 (verdadeiro) em um modelo da fórmula construída se o vértice $v_{k,j}$ (resp. $v_{i,j}$) é o sucessor do vértice $v_{i,j}$ (resp. $v_{k,j}$) no caminho codificado com o grafo G_j. Além disso, para cada $1 \leq j < |T|$ e cada par de vértices distintos $v_{i,j}, v_{k,j} \in V(G)_j$, uma variável booleana $o_{i,k,j}$ é criada, indicando que o vértice $v_{k,j}$ ocorre antes do vértice $v_{i,j}$ no caminho codificado. Para cada $1 \leq j < |T|$, as seguintes cláusulas são adicionadas à fórmula[1]:

- (*at-least*) $(\bigvee_{v_{k,j} \in N(v_{t_j,j})} s_{t_j,k,j})$. Esta cláusula indica que o vértice $v_{t_j,j}$ deve ter um sucessor no caminho codificado;
- (*at-most*) $(s_{i,k,j} \rightarrow \neg s_{i,l,j}) = (\neg s_{i,k,j} \vee \neg s_{i,l,j})$, e $(s_{k,i,j} \rightarrow \neg s_{l,i,j}) = (\neg s_{k,i,j} \vee \neg s_{l,i,j})$, para todo $v_{i,j} \in V(G)_j$ e todo $v_{k,j}, v_{l,j} \in N(v_{i,j})$. Estas cláusulas indicam que cada vértice deve possuir no máximo um sucessor e um antecessor no caminho codificado;
- (*at-least*) $s_{k,i,j} \rightarrow \bigvee_{v_{l,j} \in N(v_{i,j})} s_{i,l,j} = \neg s_{k,i,j} \vee \bigvee_{v_{l,j} \in N(v_{i,j})} s_{i,l,j}$, para todo $v_{i,j} \in V(G)_j$, $v_{i,j} \neq v_{t_{j+1},j}$ e todo $v_{k,j} \in N(v_{i,j})$. Estas cláusulas indicam que, exceto pelo último vértice do caminho codificado ($v_{t_{j+1}}$), todo vértice que possui um antecessor no caminho também deve possuir um sucessor no mesmo;
- (*inclusão de conjunto*) $(s_{i,k,j} \rightarrow o_{i,k,j})$ para todo $\{v_{i,j}, v_{k,j}\} \in E(G)_j$;
- (*transitividade*) $(o_{i,k,j} \wedge o_{k,l,j}) \rightarrow o_{i,l,j} = \neg o_{i,k,j} \vee \neg o_{k,l,j} \vee o_{i,l,j}$ para todo $v_{i,j}, v_{k,j}, v_{l,j} \in V(G)_j$;
- (*antissimetria*) $o_{i,k,j} \rightarrow \neg o_{k,i,j}$, para todo $v_{i,j}, v_{k,j} \in V(G)_j$;
- (*mínimo*) $(o_{t_j,t_{j+1},j})$. Esta cláusula unitária indica que $v_{t_j,j}$ deve ocorrer antes de $v_{t_{j+1},j}$ no caminho codificado.

É válido observar que não é possível, nesta codificação, substituir ocorrências de uma variável $o_{i,k,j}$ pela negação de $o_{k,i,j}$, uma vez que nem todos os vértices podem estar no caminho codificado, e portanto a relação binária entre alguns vértices pode ser indefinida. Por isso, é possível que que ambas as variáveis $o_{i,k,j}$ e $o_{k,i,j}$ assumam o valor 0 (falso) em um modelo da fórmula construída.

Por fim, uma variável booleana $y_{i,k}$ é criada para cada aresta $\{v_i, v_k\} \in E(G)$, indicando se a aresta está presente na componente conexa descrita por um modelo da fórmula. São adicionadas à fórmula cláusulas que codificam a seguinte relação de equivalência:

- $y_{i,k} \rightarrow \bigvee_{1 \leq j < |T|} (s_{i,k,j} \vee s_{k,i,j})$
- $\bigvee_{1 \leq j < |T|} (s_{i,k,j} \vee s_{k,i,j}) \rightarrow y_{i,k}$

Para reduzir o problema da Árvore de Steiner para PWMaxSAT, a fórmula descrita acima é utilizada como fórmula *hard*. Além disso, uma cláusula *soft* $(\neg y_{i,k}, W(\{v_i, v_k\}))$ é criada para cada aresta $\{v_i, v_k\} \in E(G)$, onde $W(e)$ descreve o peso da aresta e na instância original do problema. Desta forma, um modelo ótimo da instância PWMaxSAT construída descreve um subgrafo conexo de peso mínimo contendo todos os vértices em T, e, portanto, uma solução ótima do problema da Árvore de Steiner [Oliveira e Silva, 2014]. Esta codificação cria $O(|V(G)|^2|T|)$ variáveis, $O(|V(G)|^3|T|)$ cláusulas, e uma fórmula de tamanho $O(|V(G)|^3|T|)$.

[1] $N(v_j)$ denota a vizinhança de um vértice v_j

Além da codificação relativa por caminhos, Oliveira & Silva [Oliveira e Silva, 2014] também sugerem a codificação relativa *por árvore* (*Parental-based*). Na codificação por caminhos apresentada, cada modelo da fórmula *hard* descreve um subgrafo conexo de G contendo os vértices de T, que, informalmente, é minimizado e "transformado" em uma árvore pelas cláusulas *soft* criadas. Com a codificação relativa por árvore, cada modelo da fórmula *hard* construída descreve diretamente uma *árvore* em G que contém os vértices em T.

A árvore codificada é enraizada em um vértice arbitrário $v_r \in T$, e é então codificada como uma *relação de ordem parcial* entre seus vértices. Para cada aresta $\{v_i, v_j\} \in E(G)$, duas variáveis booleanas $p_{i,j}$ e $p_{j,i}$ são criadas. A variável $p_{i,j}$ (resp. $p_{j,i}$) assume 1 (verdadeiro) em um modelo da fórmula *hard* construída se o vértice v_j (resp. v_i) é o *pai* do vértice v_i (resp. v_j) na árvore descrita pelo mesmo. Além disso, para cada par ordenado de vértices distintos $v_i, v_j \in V(G)$, uma variável booleana $a_{i,j}$ é criada, indicando que o vértice v_j é um *ancestral* do vértice v_i na árvore codificada. Observa-se que as variáveis do tipo $a_{i,j}$ codificam o fecho transitivo da relação descrita pelas variáveis do tipo $p_{i,j}$.

A fórmula *hard* construída por esta codificação contém as seguintes cláusulas:

- (*at-least*)$(\bigvee_{v_j \in N(v_t)} p_{s,j})$ para cada vértice $v_t \in T$, $v_t \neq v_r$. Estas cláusulas codificam a condição de que todo vértice em T, exceto a raiz, possui um pai na árvore codificada;

- (*at-most*)$(p_{i,j} \rightarrow \neg p_{i,k}) = (\neg p_{i,j} \vee \neg p_{i,k})$ para cada vértice $v_i \in V(G)$, $v_i \neq v_r$ e para cada par de vértices distintos $v_j, v_k \in N(v_i)$. Estas cláusulas codificam a condição de que nenhum vértice possui mais de um pai na árvore;

- (*at-least*) $(p_{j,i} \rightarrow \bigvee_{v_k \in N(v_i)} p_{i,k}) = (\neg p_{j,i} \vee \bigvee_{v_k \in N(v_i)} p_{i,k})$, para cada vértice $v_i \in V(G), v_i \neq v_r$ e cada $v_j \in N(v_i), v_j \neq v_r$. Estas cláusulas garantem que, se um vértice $v_i \in V(G)$ possui algum filho na árvore codificada e não é sua raiz, então o vértice também deve possuir algum pai na mesma.

- *(inclusão de conjunto)* $(p_{i,j} \rightarrow a_{i,j}) = (\neg p_{i,j} \vee a_{i,j})$, para todo $v_i \in V(G), v_i \neq v_r$ e para todo $v_j \in N(v_i)$;

- *(transitividade)* $(a_{i,j} \wedge a_{j,k}) \rightarrow a_{i,k} = (\neg a_{i,j} \vee \neg a_{j,k} \vee a_{i,k})$ para cada $v_i, v_j, v_k \in V(G)$;

- *(antissimetria)* $(a_{i,j} \rightarrow \neg a_{j,i}) = (\neg a_{i,j} \vee \neg a_{j,i})$ para cada $v_i, v_j \in V(G)$;

- *(mínimo)* $(a_{t,r})$, para todo vértice $v_t \in T$, $v_t \neq v_r$. Estas cláusulas unitárias codificam a condição de que v_r é um ancestral de todos os vértices em T na árvore codificada, e, portanto, é de fato sua raiz.

Assim como com a codificação relativa por caminhos, não é possível substituir as ocorrências de uma variável $a_{i,j}$ pela negação de $a_{j,i}$, uma vez que a relação codificada não é total.

Cada modelo da fórmula criada descreve uma árvore em G que contém todos os vértices de T. Assim como na codificação por caminhos, uma variável booleana $y_{i,j}$ é criada para cada aresta $\{v_i, v_j\} \in E(G)$, e cláusulas são adicionadas à fórmula *hard* para codificar a relação $y_{i,j} \leftrightarrow (p_{i,j} \vee p_{j,i})$. Por fim, uma cláusula *soft* $(\neg y_{i,j}, W(\{v_i, v_j\}))$ é criada para cada aresta $\{v_i, v_j\} \in E(G)$.

Oliveira & Silva também sugerem dois aprimoramentos para a codificação relativa por árvore [de Oliveira e Silva, 2015]. Primeiramente, é possível reduzir o número de variáveis e de cláusulas de *transitividade* ao aplicar a técnica de Bryant & Velev [Bryant e Velev, 2002],

apresentada na subseção anterior, à codificação relativa por árvore. A técnica consiste em criar o grafo G^+ contendo as arestas do grafo G, além de arestas conectando a raiz v_r aos demais vértices em T (caso ainda não existam), torná-lo cordal, e utilizar seus ciclos de tamanho 3 para enumerar as cláusulas que codificam a propriedade de transitividade [de Oliveira e Silva, 2015].

Além disso, também é sugerido adicionar à fórmula *hard* a cláusula unitária $(a_{i,j})$ sempre que o vértice v_i *domina* o vértice v_j em relação à raiz v_r, isto é, se todos os caminhos de v_r para v_j contém o vértice v_i. Este aprimoramento tem como objetivo aumentar o nível de consistência da fórmula obtida. O conceito de consistência para fórmulas SAT é apresentado no capítulo 4.

3.3 Codificações para Operadores de Cardinalidade

Além de codificações absolutas e relativas, há também codificações SAT e MaxSAT que codificam *operadores de cardinalidade*. Informalmente, um operador de cardinalidade $k-$ário é um operador cujo valor verdade depende principalmente da *quantidade* de seus k argumentos que assumem o valor verdade 1 (verdadeiro). Operadores de cardinalidade são comumente utilizados para descrever condições relacionadas à contagem dos valores assumidos por um dado conjunto de literais.

Um operador de cardinalidade k-ário é um operador booleano com k argumentos $x_0, x_1, ..., x_{k-1}$, que assume o valor 1 (verdadeiro) se e somente se o número de seus k argumentos que assumem o valor 1 é *menor (ou igual)*, *igual* ou *maior (ou igual)* a um dado *limite* M. Um operador de cardinalidade com k argumentos e limite M é denotado por $(\sum_{0 \leq i < k} x_i) \otimes M$, com $\otimes \in \{<, \leq, =, \geq, >\}$.

Como exemplo, o operador $(\sum_{0 \leq i < 6} x_i) \leq 3$ assume o valor 1 (verdadeiro) se as variáveis x_0, x_2, x_3, e x_4 assumem o valor 0 (falso), enquanto as variáveis x_1 e x_5 assumem o valor 1 (verdadeiro). Isto ocorre pelo fato de que o número de argumentos do operador que assumem 1 é igual a 2, e $2 \leq 3$.

Para codificar o operador $(\sum_{0 \leq i < k} x_i) \otimes M$, é necessário criar uma fórmula booleana que é satisfeita por uma valoração A se e somente se o número de literais $x_i \in \{x_0, ..., x_{k-1}\}$ com $x_i \in A$ respeita o limite M, isto é, é menor que M se $\otimes$ é igual a $<$, é maior que M se $\otimes$ é igual a $>$, etc.

Existem diversas maneiras conhecidas para criar uma codificação de um operador de cardinalidade. Muller & Preparata [Muller e Preparata, 1975] apresentam uma codificação de um somador recursivo baseado em conceitos de projetos de circuitos lógicos. A fórmula resultante é uma codificação em CNF de um somador lógico cujas entradas são definidas pelos argumentos do operador, e cujos *bits* da saída indicam a representação binária da quantidade de suas entradas iguais a 1 (verdadeiro). Como alternativa a este somador, Bailleux & Boufkhad [Bailleux e Boufkhad, 2003] sugerem utilizar um circuito cujos *bits* da saída indicam a representação *unária* de tal número (denominado *somador unário*). Sinz [Sinz, 2005], por sua vez, apresenta uma técnica alternativa de codificação de um somador unário.

Por fim, o operador de cardinalidade $(\sum_{0 \leq i < k} x_i) \leq M$ também pode ser codificado através da construção de um diagrama de decisão binário ordenado e reduzido (ROBDD), como sugerido por Abío et al. [Abío et al., 2012]. Além disso, Bailleux et al. [Bailleux et al., 2009] sugerem codificações sofisticadas de equações lineares 0-1 com coeficientes quaisquer. Estas codificações codificam circuitos que utilizam somadores unários de Bailleux & Boufkhad [Bailleux e Boufkhad, 2003] como módulos internos.

Codificações de operadores de cardinalidade também podem ser utilizadas para reduzir problemas computacionais de interesse para SAT, como o problema de conectividade e da Árvore de Steiner [de Oliveira et al., 2012, Oliveira e Silva, 2014] e o problema do Ciclo Hamiltoniano [de Oliveira, 2013, de Oliveira et al., 2012].

Embora não façam parte do foco principal deste trabalho, codificações de operadores de cardinalidade podem ser citadas devido ao fato de que, assim como codificações absolutas, o nível de (arco) consistência destas codificações é conhecido [Muller e Preparata, 1975, Bailleux e Boufkhad, 2003, Sinz, 2005, Abío et al., 2012]. O conceito de consistência em codificações SAT é apresentado no capítulo 4.

3.4 Considerações finais

Este capítulo apresenta métodos de codificação e codificações conhecidas para problemas de interesse. São apresentadas as codificações absolutas, relativas e de operadores de cardinalidade.

As codificações relativas são parte do foco deste trabalho e são revisadas no capítulo 4. As codificações absolutas e de operadores de cardinalidade também são relevantes pois o nível de consistência destas codificações é conhecido e estudado. Ao melhor de nosso conhecimento, entretanto, tal nível não é estudado na literatura para codificações relativas.

O conceito de consistência em codificações SAT e sua aplicação em codificações relativas são apresentados no capítulo 4.

Capítulo 4

Arco Consistência Generalizada em Codificações Relativas

Este capítulo apresenta o conceito de Arco Consistência Generalizada no contexto do problema SAT. O conceito é apresentado para codificações absolutas e codificações relativas. Além disso, o capítulo apresenta o estudo deste conceito sobre codificações relativas, o que constitui uma das principais contribuições deste trabalho.

4.1 ACG em Codificações Absolutas e de Cardinalidade

Esta seção apresenta a definição de Arco Consistência Generalizada (ACG) em codificações SAT absolutas e de cardinalidade.

Como apresentado na seção 2.3, uma restrição de uma instância CSP é ACG se, para toda variável X_i em seu escopo e todo valor $v_j \in D_i$, existe uma valoração contendo $\{X_i = v_j\}$ que satisfaz a restrição dada. Desta forma, uma restrição *não* é ACG se existe um valor v_j no domínio D_i de alguma variável X_i tal que não existe uma valoração contendo $\{X_i = v_j\}$ que satisfaça a restrição. *Manter* a restrição ACG consiste em, após cada valoração de uma variável, aplicar um algoritmo que remove dos domínios D_i os valores v_j que não satisfazem a mesma.

Como apresentado na seção 3.1, uma codificação absoluta cria, para cada variável X_i e cada valor v_j em seu domínio, uma variável booleana $x_{i,j}$. A variável $x_{i,j}$ assume 1 (verdadeiro) em uma valoração se $X_i = v_j$ na valoração do problema CSP original, e 0 (falso) se $X_i \neq v_j$ na valoração do problema original.

Desta forma, o valor assumido por uma variável booleana $x_{i,j}$ durante o processo de busca de um resolvedor SAT é análogo ao valor assumido por uma variável X_i no processo de busca de um resolvedor CSP: Se $x_{i,j} \in A$, onde A é a valoração parcial atual, então $X_i = v_j$ na valoração equivalente em CSP; Se $\neg x_{i,j} \in A$, então $X_i \neq v_j$ (ou, equivalentemente, o valor v_j é removido do domínio D_i, ou $v_j \notin D_i$) na valoração equivalente em CSP; Se $x_{i,j}$ não é valorada, então $v_j \in D_i$ na valoração equivalente.

Desta forma, *manter* uma fórmula booleana ACG, quando obtida pela codificação absoluta, consiste em, para toda valoração parcial A explorada durante a busca, utilizar um procedimento para valorar a variável $x_{i,j}$ com 0 (falso) sempre que *não* há uma extensão de A para um modelo da fórmula que associa 1 (verdadeiro) à variável $x_{i,j}$. Equivalentemente, manter tal consistência consiste em propagar o literal $\neg x_{i,j}$ sempre que não há uma extensão da valoração parcial atual para um modelo da fórmula contendo o literal positivo $x_{i,j}$.

Como exemplo, considere uma instância CSP com duas variáveis X_1 e X_2 com domínios $D_1 = D_2 = \{a, b\}$ contendo a restrição $(\{X_1, X_2\}, \{\{X_1 = a, X_2 = a\}, \{X_1 = b, X_2 = b\}\}) = (X_1 =$

X_2). Para esta instância, a codificação absoluta por suporte irá criar quatro variáveis booleanas $x_{1,a}$, $x_{1,b}$, $x_{2,a}$ e $x_{2,b}$ e a fórmula em booleana $F = (x_{1,a} \vee x_{1,b}) \wedge (x_{2,a} \vee x_{2,b}) \wedge (\neg x_{1,a} \vee \neg x_{1,b}) \wedge (\neg x_{2,a} \vee \neg x_{2,b}) \wedge (x_{1,a} \rightarrow x_{2,a}) \wedge (x_{1,b} \rightarrow x_{2,b}) \wedge (x_{2,a} \rightarrow x_{1,a}) \wedge (x_{2,b} \rightarrow x_{1,b})$. Considere que o resolvedor SAT valora a variável $x_{1,a}$ com o valor 1 (verdadeiro). Note que, na instância CSP, isto é equivalente a associar à variável X_1 o valor a de seu domínio. Considerando a valoração atual parcial $A = \{x_{1,a}\}$, note que não existe uma extensão de A que satisfaz a fórmula contendo o literal positivo $x_{2,b}$, mas há uma extensão satisfazendo a fórmula contendo o literal negativo $\neg x_{2,b}$. Portanto, o literal $\neg x_{2,b}$ deve ser propagado para que a fórmula seja satisfeita. Se a fórmula é mantida ACG por algum procedimento, então tal procedimento deve propagar, neste ponto do algoritmo, o literal $\neg x_{2,b}$. Note que isto é equivalente, na instância CSP original, a remover o valor b do domínio da variável X_2.

Em um resolvedor SAT com o procedimento do DPLL como base, esta propagação deve ser realizada após a valoração de cada variável decidida pelo procedimento. Como apresentado na seção 2.1, os resolvedores SAT executam, neste ponto, o procedimento de Propagação Unitária, descrito pelo algoritmo 1. Desta forma, uma fórmula obtida pela codificação absoluta é mantida ACG pela Propagação Unitária se tal procedimento propaga o literal $\neg x_{i,j}$ sempre que há um modelo contendo este literal, mas não há um modelo contendo o literal $x_{i,j}$.

No exemplo dado, após a propagação do literal $x_{1,a}$, a fórmula resultante da propagação se torna $F = (x_{2,a} \vee x_{2,b}) \wedge (\neg x_{1,b}) \wedge (\neg x_{2,a} \vee \neg x_{2,b}) \wedge (x_{2,a}) \wedge (x_{1,b} \rightarrow x_{2,b}) \wedge (x_{2,b} \rightarrow x_{1,b})$. Como a fórmula contém a cláusula unitária $(x_{2,a})$, o literal positivo $x_{2,a}$ será propagado pela Propagação Unitária, o que tornará a cláusula $(\neg x_{2,a} \vee \neg x_{2,b})$ também unitária, contendo o literal $\neg x_{2,b}$. Logo, a Propagação Unitária também irá propagar o literal negativo $\neg x_{2,b}$, necessário para satisfazer a fórmula original obtida. Se o procedimento da Propagação Unitária não propagasse tal literal, então a fórmula não seria mantida ACG por tal procedimento.

De fato, fórmulas obtidas por uma codificação absoluta por suporte são mantidas ACG pela Propagação Unitária [Gent, 2002]. Além disso, consistência análoga a de *Forward-Checking* é mantida pelo mesmo procedimento para fórmulas obtidas por codificação absoluta direta [Walsh, 2000], isto é, a Propagação Unitária propaga alguns, possivelmente não todos, literais negativos que devem ser propagados para que a fórmula seja satisfeita.

O conceito de Arco Consistência Generalizada pode ser estendido para fórmulas obtidas através de outras codificações. Desta forma, uma fórmula booleana (qualquer) é *mantida ACG pela Propagação Unitária* [Abío et al., 2012] quando, para qualquer valoração parcial A e qualquer variável booleana x não valorada por A, se:

- $A \cup \{\neg x\}$ pode ser estendida para um modelo da fórmula; e
- $A \cup \{x\}$ *não* pode ser estendida para um modelo da fórmula,

então a Propagação Unitária irá propagar o literal $\neg x$. Informalmente, uma fórmula é mantida ACG pela Propagação Unitária se tal procedimento valora com 0 (falso) toda variável que deve necessariamente assumir este valor para que a fórmula seja satisfeita.

Quando uma fórmula é mantida ACG pela Propagação Unitária, várias valorações necessárias para satisfazer a fórmula poderão ser antecipadas pelo procedimento durante o processo de busca, o que pode não ocorrer se tal procedimento não mantém tal consistência.

Bjork exemplifica este comportamento com uma codificação para o operador *mux* [Björk, 2009]. Um *mux* é um circuito lógico que recebe como entrada três valores verdade s, a e b, e produz como saída y o valor de a se $s = 1$, ou o valor de b se $s = 0$. O operador pode ser codificado com quatro cláusulas: $(s \wedge a) \rightarrow y$, $(s \wedge \neg a) \rightarrow \neg y$, $(\neg s \wedge b) \rightarrow y$, e $(\neg s \wedge \neg b) \rightarrow \neg y$, ou, equivalentemente, $(\neg s \vee \neg a \vee y)$, $(\neg s \vee a \vee \neg y)$, $(s \vee \neg b \vee y)$, e $(s \vee b \vee \neg y)$. Considere

que as variáveis a e b são valoradas com 0 (falso) durante o processo de busca. A propagação de $\neg a$ e $\neg b$ resulta nas cláusulas $(\neg s \vee \neg y)$ e $(s \vee \neg y)$. Para que ambas as cláusulas sejam satisfeitas simultaneamente, a variável y deve necessariamente ser valorada com 0 (falso). Como a Propagação Unitária não realiza tal valoração com essas cláusulas, o procedimento não mantém a fórmula ACG. Assim, a valoração de y não ocorre de forma imediata durante o processo de busca. Entretanto, se as cláusulas *redundantes* $(a \wedge b) \rightarrow y$ e $(\neg a \wedge \neg b) \rightarrow \neg y$ (ou $(\neg a \vee \neg b \vee y)$ e $(a \vee b \vee \neg y)$) são adicionadas à fórmula original, então $\neg y$ será propagado caso a e b forem valorados com 0 (falso), devido à cláusula $(a \vee b \vee \neg y)$, que se torna unitária. Da mesma forma, a propagação de y, dadas as valorações de a e b com 1 (verdadeiro), ocorre de maneira análoga. De fato, com a adição das cláusulas redundantes, esta codificação para o operador *mux* é mantida ACG pela Propagação Unitária [Björk, 2009].

Assim como em codificações absolutas, o conceito de Arco Consistência Generalizada é conhecido e estudado para codificações de operadores de cardinalidade, apresentadas na seção 3.3. As fórmulas obtidas pela codificação de Bailleux & Boufhad [Bailleux e Boufkhad, 2003], de Sinz [Sinz, 2005] e de Abío et al. [Abío et al., 2012] são mantidas ACG pela Propagação Unitária [Bailleux e Boufkhad, 2003, Sinz, 2005, Abío et al., 2012], enquanto que a fórmula obtida pela codificação de Muller & Preparata [Muller e Preparata, 1975] não é mantida ACG pelo procedimento [Sinz, 2005].

Entretanto, ao melhor de nosso conhecimento, não há um estudo sobre o conceito de Arco Consistência Generalizada para codificações SAT relativas. A seção a seguir apresenta um estudo para esta classe de codificações em particular.

4.2 ACG em Codificações Relativas

Esta seção apresenta o estudo sobre a Arco Consistência Generalizada em codificações relativas. Como demostrado a seguir, a formula resultante de uma codificação relativa é mantida ACG pela Propagação Unitária dependendo de quais subfórmulas ela contém. Em particular, se a fórmula contém cláusulas *at-least* e *at-most*, ela *pode não ser* mantida ACG por este procedimento e, neste caso, conjectura-se que não é possível mantê-la ACG pelo procedimento através da criação de novas variáveis e cláusulas, em quantidade polinomial ao tamanho da fórmula resultante, mesmo quando o problema codificado é um problema polinomial.

Primeiramente, é possível demostrar que uma fórmula obtida por uma codificação relativa que contém apenas cláusulas que codificam as propriedades de *antireflexividade*, *antissimetria*, *transitividade* e *totalidade* é mantida ACG pela Propagação Unitária.

Teorema 1: Uma fórmula que codifica uma relação $R \subseteq S \times S$ antireflexiva, antissimétrica, transitiva e total é mantida ACG pela Propagação Unitária.

Prova: Seja F a fórmula obtida pela codificação relativa, e seja A uma valoração parcial tal que F não assume 0 (falso) sob A e tal que F não contém cláusulas unitárias após a propagação de A. Seja $G_T(A) = (V(G_T(A)), E(G_T(A)))$, com $V(G_T(A)) = S$ e $E(G_T(A)) = \{(a, b) \in S \times S | r_{a,b} \in A\}$ o grafo direcionado *induzido* pela valoração A, isto é, há um arco (a, b) em $G_T(A)$ se e somente se a variável $r_{a,b}$ assume 1 (verdadeiro) na valoração A.

Devido ao fato de a relação ser transitiva e antissimétrica, $G_T(A)$ não contém ciclos. Como o grafo é direcionado, $G_T(A)$ é, portanto, um Grafo Direcionado Acíclico (DAG).

Sejam $a, b \in S = V(G_T(A))$ dois vértices do grafo. Se existe um caminho de a a b no grafo $G_T(A)$, então o arco (a, b) está presente neste grafo. Este fato pode ser demonstrado por indução: considere que há um caminho de tamanho k ($k \geq 2$) de a para b, mas o arco (a, b) não pertence ao grafo. Vamos provar que existe uma cláusula unitária contendo o literal $r_{a,b}$ e, portanto, a Propagação Unitária irá propagar tal literal, fazendo tal cláusula inexistir e fazendo

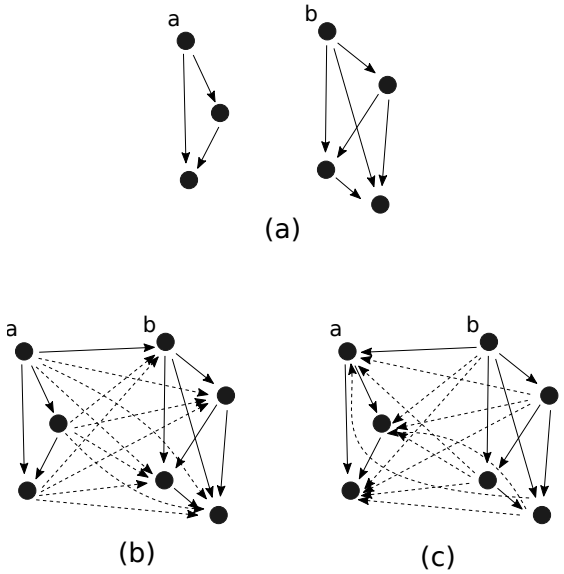

Figura 4.1: (a) $G_T(A)$ sem caminhos entre os vértices a e b; (b) Grafo contendo o arco (a, b); (c) Grafo contendo o arco (b, a). Arcos em pontilhado são incluídos pela Propagação Unitária.

também com que o arco (a, b) seja inserido no grafo. Seja $(v_0 = a, v_1, ..., v_{k-1}, v_k = b)$ um caminho de a para b de tamanho $k \geq 2$. Se $k = 2$, a existência do caminho (a, v_1, b) indica que $r_{a,v_1} \in A$ e $r_{v_1,b} \in A$. Logo, a cláusula de transitividade $(\neg r_{a,v_1} \vee \neg r_{v_1,b} \vee r_{a,b})$ torna-se a cláusula unitária $(r_{a,b})$. Se $k > 2$, então, pela hipótese da indução, o arco (a, v_{k-1}) está presente no grafo, assim como o arco (v_{k-1}, b). Logo, a cláusula de transitividade $(\neg r_{a,v_{k-1}} \vee \neg r_{v_{k-1},b} \vee r_{a,b})$ torna-se também a unitária $(r_{a,b})$, fazendo com que o literal $r_{a,b}$ seja propagado e o arco seja inserido no grafo. Logo, há um arco direto de a para b em $G_T(A)$.

Desta forma, se existe um caminho de a para b ou de b para a em $G_T(A)$, então ou $r_{a,b}$ (e $\neg r_{b,a}$) ou $r_{b,a}$ (e $\neg r_{a,b}$) estão valorados por A, dado que há um arco entre este par de vértices, como descrito anteriormente.

Considere então o caso em que não há um caminho de a para b nem de b para a em $G_T(A)$. Neste caso, nem $r_{a,b}$ e nem $r_{b,a}$ estão valorados por A. A figura 4.1(a) exemplifica a situação.

Seja M, com $A \subset M$, um modelo da fórmula estendido de A, e $G_T(M)$ seu grafo induzido. Como $G_T(M)$ é acíclico, é possível definir uma ordenação topológica dos vértices de $G_T(M)$. Além disso, como a relação binária codificada é total, a ordenação topológica dos vértices de $G_T(M)$ é única.

Além disso, também pelo fato da relação codificada ser total, há necessariamente um arco entre os vértices a e b em $G_T(M)$. A orientação deste arco depende da ordenação topológica dos vértices de $G_T(M)$: se o vértice de a precede o vértice b na ordenação topológica de $G_T(M)$, então há um arco de a para b em $G_T(M)$. Se o vértice b precede o vértice a, por sua vez, então há um arco de b para a em $G_T(M)$.

Desta forma, A pode ser estendida para pelo menos dois modelos de F; um dos modelos contém os literais $r_{a,b}$ e $\neg r_{b,a}$, enquanto o segundo contém os literais $\neg r_{a,b}$ e $r_{b,a}$. A figura 4.1(b) exemplifica o grafo $G_T(M)$ contendo o arco (a, b), indicando a existência de um modelo da fórmula contendo os literais $r_{a,b}$ e $\neg r_{b,a}$. A figura 4.1(c), por sua vez, exemplifica o grafo $G_T(M)$ contendo o arco (b, a), indicando a existência de um modelo da fórmula contendo os literais $r_{b,a}$ e $\neg r_{a,b}$. É válido observar que a propagação de um literal $r_{a,b}$ implica também na possível adição de outros arcos ao grafo induzido pela valoração atual.

Assim, se a fórmula resultante da propagação de uma valoração parcial A não contém cláusulas unitárias e existe uma variável não valorada $r_{a,b}$ tal que $A \cup \{\neg r_{a,b}\}$ pode ser estendida para um modelo de F, então $A \cup \{r_{a,b}\}$ também pode ser estendida para um modelo da mesma fórmula.

Logo, se existe uma valoração parcial A e alguma variável $r_{a,b}$ não valorada por A tal que $A \cup \{\neg r_{a,b}\}$ pode ser estendida para um modelo de F enquanto $A \cup \{r_{a,b}\}$ não pode, então certamente existe um caminho de b para a em $G_T(A)$. Se tal caminho não existisse, então $A \cup \{r_{a,b}\}$ poderia ser estendida para um modelo da fórmula, como apresentado anteriormente.

A existência de tal caminho fará com que a Propagação Unitária propague o literal $r_{b,a}$, e, em particular, o literal $\neg r_{a,b}$, como apresentado anteriormente. Logo, o literal $\neg r_{a,b}$ é propagado sempre que $A \cup \{\neg r_{a,b}\}$ pode ser estendida para um modelo de F enquanto $A \cup \{r_{a,b}\}$ não pode, e, portanto, a Propagação Unitária mantém a fórmula ACG. ■

É possível observar que, de maneira análoga à prova do Teorema 1, a fórmula também é mantida ACG quando ambas as relações R e seu fecho transitivo R^+ são codificadas com a inclusão de cláusulas de inserção de conjunto.

Também é possível mostrar que a Propagação Unitária mantém a fórmula ACG quando o método de Bryant & Velev [Bryant e Velev, 2002], apresentado na seção 3.2, é utilizado para reduzir o número de cláusulas de transitividade. De maneira análoga à prova do Teorema 1, se existe um caminho de um vértice a para um vértice b em $G_T(A)$, então o arco (a, b) está presente no grafo devido à propagação do literal $r_{a,b}$, realizada pela Propagação Unitária. Este fato também pode ser demonstrado por indução no tamanho k do caminho de a para b: se $k = 2$, então uma cláusula de transitividade contendo o literal $r_{a,b}$ se tornará unitária, e tal literal será propagado. Caso contrário, existe no grafo G^+, o grafo induzido pelas variáveis criadas como definido na subseção 3.2.1, um ciclo contendo o caminho existente entre a e b e a aresta $\{a, b\}$. De acordo com a técnica de Bryant & Velev [Bryant e Velev, 2002], este grafo é cordal, e, portanto, há um vértice v_j no caminho de a a b tal que existe um caminho a a v_j, um caminho de v_j a b e as arestas arestas $\{a, v_j\}$ e $\{v_j, b\}$ no grafo G^+. Os arcos (a, v_j) e (v_j, b) são adicionados a $G_T(A)$ pela hipótese da indução, enquanto o arco (a, b) é adicionado ao mesmo grafo devido ao caminho de tamanho $k = 2$ de a a b contendo o vértice v_j. Desta forma, se existe um caminho de a a b em $G_T(A)$, então o literal $r_{a,b}$ será propagado. Caso contrário, há um modelo estendido de A contendo $r_{a,b}$ e outro contendo $r_{b,a}$ de acordo com a ordenação topológica assumida pelos elementos dados, o que justifica a manutenção da consistência na fórmula de maneira análoga à prova do Teorema 1.

O Teorema 1 indica que fórmulas que codificam uma relação (apenas) antireflexiva, antissimétrica, transitiva e total são mantidas ACG pela Propagação Unitária. Entretanto, fórmulas contendo cláusulas que codificam mais propriedades, e, em particular, as que contém cláusulas *at-least* e *at-most*, podem ser ou podem não ser mantidas ACG pelo mesmo procedimento, dependendo da estrutura da relação dada.

Teorema 2: Fórmulas obtidas por codificações relativas contendo cláusulas *at-least* unitárias são mantidas ACG pela Propagação Unitária.

Prova: Considere uma fórmula que codifica uma relação $R \subseteq S \times S$ antireflexiva, antissimétrica, transitiva e total, contendo ainda, para cada $a \in S$, a cláusula *at-least* $(\bigvee_{l \in L_a} r_{a,l})$ e as cláusulas *at-most* $\bigwedge_{m,n \in M_a} (\neg r_{a,m} \vee \neg r_{a,n})$, onde L_a (resp. M_a) é o conjunto de elementos tal que a deve se relacionar com pelo menos um (resp. no máximo um) elemento nele.

Considere $L_a = M_a$ e $|L_a| = |M_a| = 1$. Neste caso, todas as cláusulas *at-least* são unitárias e, logo, a Propagação Unitária irá propagar seus literais, removendo estas cláusulas da

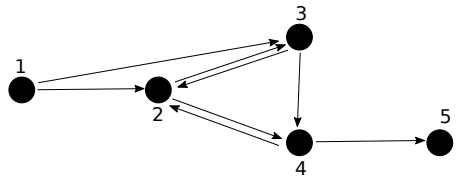

Figura 4.2: Grafo de exemplo para a prova do Teorema 3

fórmula. Após este procedimento, a fórmula será mantida ACG como demonstrado pelo Teorema 1. ■

O Teorema 2 indica que existem fórmulas contendo cláusulas *at-least* e *at-most* que são mantidas ACG pela Propagação Unitária. Entretanto, embora existam fórmulas com tais cláusulas que são mantidas ACG pela Propagação Unitária, outras fórmulas contendo os mesmos tipos de cláusulas podem não ter tal manutenção realizada, como demonstrado pelo Teorema 3.

Teorema 3: Há fórmulas obtidas por codificações relativas contendo cláusulas *at-least* e *at-most* que não são mantidas ACG pela Propagação Unitária.

Prova: Como exemplo, considere o grafo apresentado na figura 4.2 e a fórmula contendo, para cada vértice $a \in S$, cláusulas *at-least* e *at-most* para todos os vértices b tal que há um arco de a para b no grafo. Considere uma valoração parcial A contendo o literal $r_{1,5}$ indicando, por transitividade, que deve haver um caminho do vértice 1 ao vértice 5 no grafo induzido pelos modelos da fórmula.

O vértice 4 precede o vértice 5 em todos os caminhos possíveis e, logo, o literal $r_{4,5}$ (e, logo, $\neg r_{5,4}$) deve ser propagado. Entretanto, a Propagação Unitária não irá propagar este literal neste caso, dado que não há cláusula unitária na fórmula gerada, pois o grau de entrada e de saída de todos os vértices do grafo é 0 ou 2. ■

Desta forma, fórmulas contendo cláusulas *at-least* e *at-most* não são mantidas ACG pela Propagação Unitária no caso geral. Neste trabalho, é apresentada a conjectura de que a Propagação Unitária pode continuar não mantendo a fórmula ACG mesmo se alguma informação redudante, de tamanho polinomial no tamanho da fórmula, for adicionada à mesma.

A seguinte definição é necessária para a apresentação da conjectura:

Definição: (*polinomialmente restrita*) Uma fórmula F pode ser *polinomialmente restrita* a outra fórmula $\mathcal{F}$ se existe uma árvore de resolução para F, de tamanho polinomial em $|F|$, contendo as cláusulas de $\mathcal{F} \setminus F$.

Como definido no capítulo 2, uma árvore de resolução para F tem como folhas as cláusulas de F, e como nodos internos cláusulas resultantes da regra de resolução com outras cláusulas na árvore. Por isso, se F é polinomialmente restrita a $\mathcal{F}$, então:

- F é uma subfórmula de $\mathcal{F}$;
- $\mathcal{F}$ tem o mesmo conjunto de variáveis de F;
- $\mathcal{F}$ tem o mesmo conjunto de modelos de F (e, logo, $\mathcal{F}$ é satisfatível se e somente se F é);
- $|\mathcal{F}|$ é polinomial em $|F|$;

Se F pode ser polinomialmente restrita a $\mathcal{F}$, então $\mathcal{F}$ é uma *restrição polinomial* de F.

Informalmente, afirmar que uma fórmula pode ser polinomialmente restrita à outra mantida ACG pela Propagação Unitária é equivalente a afirmar que é possível adicionar à fórmula informações *redundantes*, de tamanho polinomial, para fazê-la ser mantida ACG por tal procedimento.

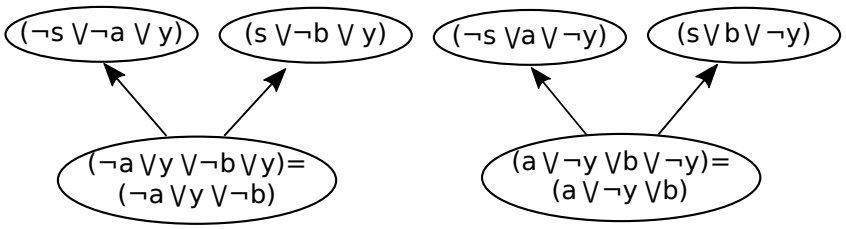

Figura 4.3: Árvore de resolução da codificação de Bjork [Björk, 2009] de um *mux*, exemplificando uma restrição polinomial para mantê-la ACG

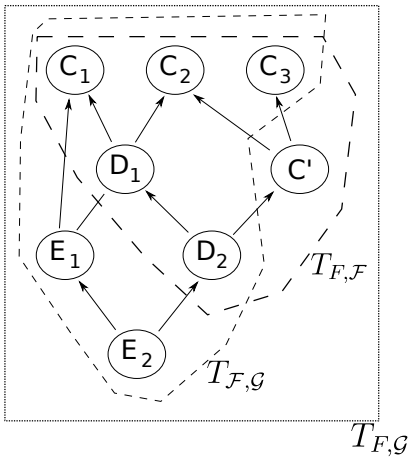

Figura 4.4: Árvore de resolução que restringe $F = (C_1 \wedge C_2 \wedge C_3)$ a $\mathcal{G} = \mathcal{F} \wedge (E_1 \wedge E_2)$ com $\mathcal{F} = F \wedge (D_1 \wedge D_2)$, utilizada como exemplo para prova do Teorema 4.

Como exemplo, considere a codificação de Bjork [Björk, 2009] para um *mux* apresentada na seção 4.1. A fórmula F contendo as quatro cláusulas originais da codificação não é mantida ACG pela Propagação Unitária, enquanto a fórmula $\mathcal{F}$, resultante da adição de duas cláusulas redundantes à fórmula F, é mantida ACG pelo mesmo procedimento. Neste caso, é possível notar que: $F \subset \mathcal{F}$; que $\mathcal{F}$ é satisfatível se e somente se F é (dado que as cláusulas adicionadas consistem em informações redundantes à fórmula); $|\mathcal{F}|$ é polinomial em $|F|$; e que existe uma árvore de resolução para F que contém as duas cláusulas de $\mathcal{F} \setminus F$, como apresentada pela figura 4.3. Logo, a fórmula $\mathcal{F}$ é uma restrição polinomial da fórmula F, mantida ACG pela Propagação Unitária.

A relação entre fórmulas polinomialmente restritas é transitiva:

Teorema 4: Sejam F, $\mathcal{F}$ e $\mathcal{G}$ fórmulas booleanas. Se $\mathcal{F}$ é uma restrição polinomial de F e $\mathcal{G}$ é uma restrição polinomial de $\mathcal{F}$, então $\mathcal{G}$ é uma restrição polinomial de F.

Prova: Seja $T_{F,\mathcal{F}}$ a árvore de resolução que restringe polinomialmente F em $\mathcal{F}$, e seja $T_{\mathcal{F},\mathcal{G}}$ a árvore de resolução que restringe polinomialmente $\mathcal{F}$ em $\mathcal{G}$. Seja $T_{F,\mathcal{G}} = T_{F,\mathcal{F}} \cup T_{\mathcal{F},\mathcal{G}}$ a união de ambas as árvores. A árvore $T_{F,\mathcal{G}}$ contém tanto os nodos quanto os arcos de ambas e apenas ambas as árvores $T_{F,\mathcal{F}}$ e $T_{\mathcal{F},\mathcal{G}}$.

Como exemplo, considere uma fórmula $F = (C_1 \wedge C_2 \wedge C_3)$, restrita polinomialmente a uma fórmula $\mathcal{F} = F \wedge (D_1 \wedge D_2)$, que, por sua vez, é restrita polinomialmente a uma fórmula $\mathcal{G} = \mathcal{F} \wedge (E_1 \wedge E_2)$. A figura 4.4 exemplifica as árvores $T_{F,\mathcal{F}}$, $T_{\mathcal{F},\mathcal{G}}$ e $T_{F,\mathcal{G}}$. A cláusula C' é uma cláusula auxiliar de exemplo.

Como $|T_{F,\mathcal{F}}|$ é polinomial em $|F|$, e como $|T_{\mathcal{F},\mathcal{G}}|$ é polinomial em $|\mathcal{F}|$, então $|T_{F,\mathcal{G}}|$ é polinomial em $|F| + |\mathcal{F}|$. Entretanto, como $T_{F,\mathcal{F}}$ é uma árvore de resolução polinomial em F, $|\mathcal{F}|$ também é polinomial em $|F|$. Logo, $|T_{F,\mathcal{G}}|$ é polinomial em $|F|$. Além disso, a árvore $T_{F,\mathcal{G}}$ contém as cláusulas de F como folhas, além de cláusulas de $\mathcal{G}$ como nodos internos. Logo, $T_{F,\mathcal{G}}$ é uma árvore de resolução que restringe polinomialmente F em $\mathcal{G}$. ■

É importante notar que o Teorema 4 indica que, se uma fórmula F pode ser restrita polinomialmente à outra mantida ACG pela Propagação Unitária, e se F é uma restrição polinomial de alguma de suas subfórmulas $F' \subset F$, então a subfórmula F' pode ser polinomialmente restrita para ser mantida ACG pela Propagação Unitária.

Além disso, se uma fórmula é mantida ACG pela Propagação Unitária, então sua satisfabilidade pode ser determinada em tempo polinomial:

Algoritmo 2: Satisfabilidade para fórmula mantida ACG pela Propagação Unitária

Entrada: Uma fórmula F
Saída: "SAT" se F é satisfatível, "UNSAT" caso contrário

```
1  F = PropagaçãoUnitária(F)
2  enquanto □ ∉ F e F ≠ ∅ faça
3      x = variável não valorada em F
4      F = Propagação(F, x)
5      F = PropagaçãoUnitária(F)
6  fim
7  se □ ∈ F então
8      retorna "UNSAT"
9  fim
10 se F = ∅ então
11     retorna "SAT"
12 fim
```

Teorema 5: Se a Propagação Unitária mantém a Arco Consistência Generalizada em uma fórmula F, então a satisfabilidade de F pode ser determinada em tempo polinomial.

Prova: Se uma fórmula F é mantida ACG pela Propagação Unitária, sua satisfabililidade pode ser determinada pelo procedimento descrito pelo Algoritmo 2.

Primeiramente, a Propagação Unitária é aplicada (linha 1). Então, é escolhida (qualquer) uma variável x ainda não valorada pela valoração atual (linha 3), e valora-se esta variável *gulosamente* com o valor 1 (verdadeiro) (linha 4). A Propagação Unitária é então aplicada novamente (linha 5). O processo é então repetido até que a fórmula contenha uma cláusula vazia ou até que todas as variáveis sejam valoradas, caso em que todas as cláusulas são removidas da fórmula (linha 2). Ao final do processo, se a cláusula vazia estiver contida em F, então F é insatisfatível (linhas 7 a 9). Caso contrário, F é satisfatível (linhas 10 a 12).

Como os procedimentos de propagações são polinomiais no tamanho da fórmula e como tais procedimentos são realizados um número polinomial de vezes, o algoritmo apresentado é polinomial em tempo no tamanho da fórmula dada.

Além disso, se a cláusula vazia □ não é encontrada durante o algoritmo (e, logo, se o algoritmo retorna "SAT"), então a valoração obtida ao final do mesmo satisfaz a fórmula dada, e, portanto, a fórmula é satisfatível. Em seguida, demonstra-se que, se a fórmula é satisfatível, então a cláusula vazia não é encontrada pelo algoritmo (e, logo, ele retorna "SAT").

Em todo ponto do algoritmo, seja $M_F(A)$ o conjunto de modelos da fórmula F que são extensões da valoração parcial atual A. É válido notar que, em particular, $M_F(\emptyset)$ é o conjunto de todos os modelos de F.

Para toda variável x tal que não existe um modelo de F que é uma extensão de A e que contém o literal x, a Propagação Unitária propagará $\neg x$, dado que tal procedimento mantém a fórmula ACG. Seja A então a valoração obtida após a aplicação da Propagação Unitária. Naturalmente, o conjunto de modelos $M_F(A)$ não é alterado após tal Propagação. Seja x uma variável não valorada pela Propagação Unitária. Certamente há pelo menos um modelo em $M_F(A)$ contendo o literal x, uma vez que, caso contrário, a Propagação Unitária teria valorado tal variável. Desta forma, valorar a variável x com o valor 1 (verdadeiro) não torna o conjunto de modelos $M_F(A)$ vazio (embora tal valoração pode, de fato, reduzir o tamanho de $M_F(A)$ removendo possíveis modelos contendo o literal $\neg x$).

Como o conjunto de modelos $M_F(A)$ não se torna vazio durante o processo, certamente um modelo para a fórmula original F será encontrado pelo procedimento. Logo, a cláusula vazia não será encontrada e, portanto, o algoritmo retornará "SAT". ■

O Teorema 5 afirma que o problema SAT é polinomial para a classe de instâncias que são mantidas ACG pela Propagação Unitária. De fato, pode-se demostrar, de forma análoga, que o problema é polinomial se a fórmula dada for mantida ACG por qualquer procedimento de tempo polinomial no tamanho da fórmula dada.

Como exemplo de execução do Algoritmo 2, considere a fórmula que codifica um *mux* $F = (\neg s \vee \neg a \vee y) \wedge (\neg s \vee a \vee \neg y) \wedge (s \vee \neg b \vee y) \wedge (s \vee b \vee \neg y) \wedge (\neg a \vee \neg b \vee y) \wedge (a \vee b \vee \neg y)$, como apresentada na seção 4.1. Como F não contém cláusulas unitárias, a Propagação Unitária inicialmente não modifica a fórmula. Em seguida, uma variável não valorada é escolhida. Considere a escolha da variável s. O literal positivo s é então propagado, resultando na fórmula $F = (\neg a \vee y) \wedge (a \vee \neg y) \wedge (\neg a \vee \neg b \vee y) \wedge (a \vee b \vee \neg y)$. A Propagação Unitária novamente não altera a fórmula, e outra variável é escolhida. Considere que a variável a é escolhida. O literal positivo a é então propagado, resultando na fórmula $F = (y) \wedge (\neg b \vee y)$. A Propagação Unitária então propaga o literal y, que torna a fórmula F vazia. O algoritmo então retorna "SAT". De fato, qualquer extensão da valoração atual $A = \{s, a, y\}$ satisfaz a fórmula dada.

Corolário 6: Sob $P \neq NP$, há codificações relativas para as quais não existem restrições polinomiais que as mantém ACG. A codificação relativa de *Prestwich* [Prestwich, 2003] é uma redução para SAT do problema do Ciclo Hamiltoniano, que está na classe de problemas NP-Completo [Karp, 1972]. Se a fórmula resultante desta codificação for polinomialmente restrita à outra mantida ACG pela Propagação Unitária, então, pelo Teorema 5, o problema poderia ser resolvido em tempo polinomial, o que resultaria no colapso $P = NP$.

O Corolário 6 indica que há codificações relativas que não podem ser polinomialmente restritas para serem mantidas ACG pela Propagação Unitária. Informalmente, o Teorema 1 e o Corolário 6 indicam situações "extremas" de codificações relativas: fórmulas que não contém cláusulas *at-least* e *at-most* são mantidas ACG pela Propagação Unitária, enquanto fórmulas contendo tais cláusulas não as são e, dependendo da maneira com que são utilizadas, não podem ser polinomialmente restritas para serem mantidas ACG pelo mesmo procedimento.

Entretanto, existem codificações relativas utilizadas para reduzir problemas polinomiais cujas fórmulas não são mantidas ACG pela Propagação Unitária. Como exemplo, a fórmula *hard* da codificação por árvore de Oliveira & Silva [Oliveira e Silva, 2014] para o problema da Árvore de Steiner é satisfatível se e somente se os vértices de um conjunto dado estão na mesma componente conexa do grafo dado. Determinar tal conectividade é polinomial, dado que o problema pode ser resolvido, por exemplo, com uma busca em profundidade no grafo dado.

Embora existem problemas polinomiais que podem ser reduzidos para SAT por codificações relativas, conjectura-se que a fórmula obtida por tais codificações também não pode ser polinomialmente restrita para ser mantida ACG pela Propagação Unitária.

Conjectura 7: Há fórmulas contendo apenas cláusulas *at-least* e *at-most* que não podem ser polinomialmente restritas a outras mantidas ACG pela Propagação Unitária.

Como motivação para a conjectura, é possível reduzir para SAT o problema que consiste em decidir se um grafo bipartido tem um emparelhamento perfeito usando apenas cláusulas *at-least* e *at-most*. Seja $\{A, B\}$ a partição dos vértices $V(G)$ do grafo dado. Para cada aresta $\{a, b\} \in E(G)$ com $a \in A, b \in B$, uma variável booleana $r_{a,b}$ é criada. A variável $r_{a,b}$ assume o valor 1 (verdadeiro) em um modelo da fórmula criada se e somente se a aresta $\{a, b\}$ está presente no emparelhamento perfeito.

Para cada vértice $a \in A$, é criada uma cláusula *at-least* ($\bigvee_{\{a,b\} \in E(G)} r_{a,b}$) e cláusulas *at-most* $\bigwedge_{\{a,b\},\{a,c\} \in E(G)} (\neg r_{a,b} \vee \neg r_{a,c})$. Estas cláusulas indicam que o vértice a deve estar emparelhado com exatamente um vértice em B. Para cada vértice $b \in B$, cláusulas análogas são criadas para indicar que tal vértice deve estar emparelhado com exatamente um vértice em A. A fórmula resultante desta redução é satisfatível se e somente se o grafo dado tem um emparelhamento perfeito.

Considere as instâncias do problema com o grafo bipartido completo $K_{n+1,n}$, para algum inteiro n. Esta instância é claramente insatisfatível, pois $|B| > |A|$. Decidir a resposta para esta instância é equivalente a refutar o a fórmula que codifica o *Princípio das Casas dos Pombos* com $n + 1$ pombos e n casas, denotada por PHP_n^{n+1} [Buss e Pitassi, 1998].

De acordo com Buss & Pitassi [Buss e Pitassi, 1998], não existe uma árvore de refutação polinomial que refuta PHP_n^{n+1}. Conjectura-se que, se a fórmula puder ser polinomialmente restrita à outra fórmula mantida ACG pela Propagação Unitária, então pode existir uma árvore de refutação polinomial que refuta PHP_n^{n+1}, o que seria um absurdo.

A Conjectura 7 indica que fórmulas contendo apenas cláusulas *at-least* e *at-most* podem não ser polinomialmente restritas a fórmulas mantidas ACG pela Propagação Unitária, no caso geral.

Em particular, a conjectura 7 indica que não existe uma codificação *relativa* polinomial mantida ACG pela Propagação Unitária para o problema do emparelhamento perfeito. Logo, este problema não poderia ser resolvido em tempo polinomial por um resolvedor SAT que utiliza a Propagação Unitária recebendo, como entrada, uma codificação relativa. Este fato pode não ser intuitivo, uma vez que o problema do emparelhamento perfeito é, de fato, polinomial (ele pode ser resolvido, por exemplo, com uma redução para o problema de Fluxo Máximo e pelo algoritmo de Ford-Fulkerson [Ford e Fulkerson,]).

O teorema abaixo indica que fórmulas obtidas pela codificação relativa contendo cláusulas *at-least* e *at-most* têm tamanho polinomial no tamanho das mesmas:

Teorema 8: O tamanho total das demais cláusulas presentes em codificações relativas é polinomial no tamanho total das cláusulas *at-least* e *at-most*.

Prova: Seja $|at|$ o tamanho total das cláusulas *at-least* e *at-most*.

Tal tamanho é polinomial em $|S|$, pois há $O(|S|)$ cláusulas *at-least* com no máximo $|S|$ literais cada, e no máximo $\sum_{a \in S}(|S|^2) \leq |S|^3$ cláusulas *at-most* com 2 literais cada. Logo, existe um inteiro positivo k tal que $|at| = O(|S|^k)$.

O tamanho total das demais cláusulas é, naturalmente, polinomial em $|S|$. De fato, o tamanho total da fórmula obtida por uma codificação relativa está em $O(|S|^3)$, devido principalmente ao tamanho das cláusulas de transitividade.

Desta forma, o tamanho total das demais cláusulas está em $O(|S|^3) = O((|S|^k)^{\frac{3}{k}}) = O(|at|^{\frac{3}{k}})$, que é polinomial em $|at|$. ■

O Teorema 8 afirma que, em codificações relativas, o tamanho de qualquer outra subfórmula da fórmula obtida é polinomial no tamanho da subfórmula contendo apenas as cláusulas *at-least* e *at-most*. Quando observado junto ao Teorema 4 e à Conjectura 7, o teorema indica que, se a fórmula for uma restrição polinomial da subfórmula que contém tais cláusulas, então ela não pode ser polinomialmente restrita à outra mantida ACG pela Propagação Unitária.

Além disso, conjectura-se também que a fórmula pode não ser mantida ACG por tal procedimento mesmo se a mesma não for uma restrição polinomial da subfórmula contendo tais cláusulas.

Conjectura 9: Codificações relativas contendo cláusulas *at-least* e *at-most* não podem ser polinomialmente restritas para serem mantidas ACG pela Propagação Unitária, no caso geral.

Desta forma, acredita-se que não é possível manter uma codificação relativa ACG pela Propagação Unitária no caso em que a codificação contém cláusulas *at-least* e *at-most*, no caso geral. Logo, embora um dado problema computacional de decisão possa ser resolvido em tempo polinomial, resolvê-lo utilizando uma redução para SAT através de uma codificação relativa mantida ACG pela Propagação Unitária pode levar tempo exponencial no tamanho de sua entrada.

Logo, pode não ser possível injetar informações, por exemplo, à fórmula *hard* da codificação por árvore de Oliveira & Silva [Oliveira e Silva, 2014] para o problema da Árvore de Steiner para manté-la ACG pela Propagação Unitária, mesmo dado que o problema codificado pela fórmula *hard*, que consiste em determinar a conectividade de vértices específicos no grafo dado, é polinomial.

4.3 Considerações finais

Como apresentado na seção anterior, pode não ser possível manter ACG pela Propagação Unitária uma fórmula obtida por uma codificação relativa, no caso geral. Entretanto, é possível injetar informações novas, na fórmula ou no resolvedor SAT, de tal forma que uma consistência *parcial* é mantida. Neste caso, o resolvedor pode propagar *alguns* (mas possivelmente não todos) literais que devem ser propagados para tornar a fórmula satisfatível. Desta forma, é possível podar *alguns* subespaços de busca que não contém uma solução para o problema SAT, reduzindo assim o espaço explorado pelo resolvedor. O capítulo 5 apresenta um método para tornar tal consistência parcial possível.

Capítulo 5

Consistência Parcial em Codificações Relativas

Como discutido no capítulo 4, conjectura-se não ser possível injetar informações de tamanho polinomial em uma codificação relativa para fazer a Propagação Unitária manter (inteiramente) a Arco Consistência Generalizada da fórmula, no caso geral. Desta forma, um resolvedor SAT pode não propagar todos os literais negativos necessários para satisfazer a fórmula dada. Entretanto, é possível fazer o resolvedor manter um nível parcial de consistência, e propagar *alguns*, possivelmente não todos, literais negativos necessários.

A próxima seção apresenta uma técnica para manter tal consistência parcial, enquanto a seguinte apresenta seus resultados experimentais.

5.1 Manutenção da Árvore de Dominantes

Esta seção apresenta a técnica de manutenção da árvore de dominantes do grafo induzido pela relação durante o procedimento de busca de um resolvedor SAT. Como discutido, a técnica tem como objetivo a manutenção de uma consistência parcial em fórmulas obtidas por codificações relativas. A técnica foi primeiramente apresentada por Oliveira & Silva [de Oliveira e Silva, 2015] como um procedimento de pré-processamento. Neste trabalho, a técnica é estendida para um procedimento interno ao resolvedor SAT, durante o processo de busca.

Algumas definições são necessárias para o entendimento da técnica. Primeiramente, dado um grafo $G = (V(G), E(G))$ e um vértice $m \in V(G)$, para qualquer par de vértices $a, b \in V(G), a \neq b$, o vértice *a domina* o vértice *b* (em relação ao vértice *m*) se e somente se todo caminho do vértice *m* para o vértice *b* contém o vértice *a*.

Além disso, dado um grafo $G = (V(G), E(G))$ e um vértice $m \in V(G)$, para cada vértice $b \in V(G), b \neq m$, o *dominante imediato* do vértice *b*, denotado por $idom(b)$, é definido como o vértice *a* tal que *a* domina *b* e *não* existe um vértice *c* tal que *a* domina *c* e *c* domina *b*. O dominante imediato de cada vértice é único [Lengauer e Tarjan, 1979].

Por fim, a *Árvore de Dominantes* $D(G)$ de um grafo G (em relação a um vértice $m \in V(G)$) é uma árvore enraizada cujos nodos são os vértices de G, cuja raiz é o vértice m, e cujo $idom(b)$ é o pai do nodo b na árvore, para todo $b \in V(G), b \neq m$. É válido observar que um vértice $a \in V(G)$ domina um vértice $b \in V(G)$ em G se e somente se o vértice a é um ancestral do vértice b na árvore $D(G)$.

Seja $R \subseteq S \times S$ a relação binária codificada, e R^+ seu fecho transitivo. Considere que ambas as relações R e R^+ são codificadas, que R^+ é antireflexiva e antissimétrica, e

que $m \in S$ é um elemento mínimo de R. Desta forma, a fórmula obtida pela codificação relativa contém cláusulas de antireflexividade, antisimetria, transitividade, inclusão de conjunto e mínimo. Codificações conhecidas para problemas de interesse contém estas cláusulas [Prestwich, 2003, Velev e Gao, 2010].

Considere também que a fórmula obtida contém cláusulas *at-least* e *at-most*, e que, para cada elemento $a \in S$, os conjuntos $M_a \subseteq S$, que indica os elementos tal que a deve se relacionar a no máximo um deles, e $L_a \subseteq S$, que indica os elementos tal que a deve se relacionar a pelo menos um deles, são iguais, isto é $M_a = L_a$. Em outras palavras, as cláusulas codificam a restrição de que a deve se relacionar a exatamente um elemento de um dado conjunto de elementos $M_a = L_a$. A codificação relativa de Prestwich [Prestwich, 2003] para o problema do Ciclo Hamiltoniano, por exemplo, contém todos os tipos de cláusulas citados.

Seja $G = (V(G), E(G))$ o grafo definido por $V(G) = S$ e $E(G) = \{\{a, b\} \in \binom{S}{2} | b \in M_a\}$. Informalmente, o grafo G é o grafo induzido pelas restrições definidas pelas cláusulas *at-least* e *at-most*, de tal forma que, na relação codificada, um elemento a deve se relacionar a exatamente um elemento de sua vizinhança no grafo G. Por fim, em qualquer ponto do procedimento de busca do resolvedor SAT, seja A a valoração parcial atual sobre as variáveis da fórmula dada.

Seja $G'(A) = (V(G'(A)), E(G'(A)))$, com $V(G'(A)) = V(G)$ e $E(G'(A)) = \{(a, b) \in S \times S | \{a, b\} \in E(G) \wedge \neg r_{a,b} \notin A\}$, o grafo (direcionado) *parcial* induzido pela valoração parcial A. Informalmente, $G'(A)$ é o grafo direcionado contendo todos os vértices de G que contém um arco (a, b) para todo par ordenado de vértices originalmente conectados por uma aresta em G e cujas respectivas variáveis booleanas $r_{a,b}$ não são valoradas por A ou são valoradas com 1 (verdadeiro). Observa-se que $G'(\emptyset) = (S, \{(a, b) \in S \times S | \{a, b\} \in E(G)\})$, isto é, o grafo parcial inicial $G'(\emptyset)$ é o grafo G tal que cada aresta é substituída pelos seus dois arcos respectivos.

A figura 5.1 exemplifica (a) um grafo G, (b) o respectivo grafo $G'(\emptyset)$, e (c) sua árvore de dominantes.

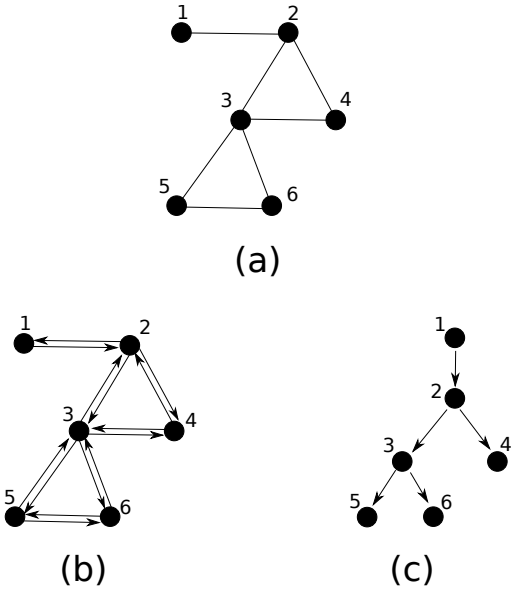

Figura 5.1: (a) Um grafo G; (b) O grafo $G'(\emptyset)$; (c) Sua árvore de dominantes em relação ao vértice 1

É possível observar que, se um vértice $a \in S$ domina um vértice $b \in S$ em $G'(A)$ (em relação ao elemento mínimo $m \in S$) para uma dada valoração parcial A, então o literal $\neg r^+_{b,a}$, assim como o literal $r^+_{a,b}$, deve necessariamente ser propagado para tornar a fórmula satisfatível.

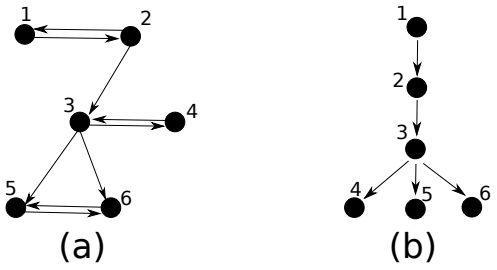

Figura 5.2: (a) O grafo $G'(A)$ com $\{\neg r_{3,2}, \neg r_{5,3}, \neg r_{6,3}\} \subset A$; (b) Sua árvore de dominantes em relação ao vértice 1

Logo, é possível, para qualquer valoração parcial A obtida durante o processo de busca do resolvedor SAT, propagar o literal $\neg r^+_{b,a}$ sempre que a domina b (com relação a m) no grafo $G'(A)$. Neste caso, o literal $r^+_{a,b}$, assim como o literal $\neg r_{b,a}$ (se existir), também pode ser propagado.

No grafo exemplificado pela figura 5.1 (b), o vértice 2 domina os vértices 3, 4, 5 e 6, assim como o vértice 3 domina os vértices 5 e 6. Logo, os literais $r^+_{2,3}, r^+_{2,4}, r^+_{2,5}, r^+_{2,6}, r^+_{3,5}, r^+_{3,6}$, assim como $\neg r^+_{3,2}, \neg r^+_{4,2}, \neg r^+_{5,2}, \neg r^+_{6,2}, \neg r^+_{5,3}, \neg r^+_{6,3}$, podem ser propagados. Note que, devido às cláusulas de inclusão de conjuntos, os literais $\neg r_{3,2}, \neg r_{5,3}, \neg r_{6,3}$ também são propagados. A figura 5.2 indica (a) o grafo parcial induzido pela valoração atual, e (b) sua árvore de dominantes.

A propagação de todos os literais citados acima ocorreria se a Propagação Unitária mantivesse ACG a fórmula obtida pela codificação. Entretanto, este procedimento pode não propagar tal literal, como exemplificado pela figura 4.2. Logo, fazer o resolvedor SAT propagar estes literais nestes casos pode aumentar o nível de consistência mantida pela Propagação Unitária. É válido observar, entretanto, que esta propagação não resulta na manutenção (total) da Arco Consistência Generalizada da fórmula.

A técnica apresentada anteriormente por Oliveira & Silva [de Oliveira e Silva, 2015] consiste em adicionar à fórmula as cláusulas unitárias $(r^+_{a,b})$ e $(\neg r^+_{b,a})$ para todo par de vértices $a, b \in S$ tal que a domina b (em relação ao mínimo m) no grafo $G'(\emptyset)$. Para determinar os pares de vértices a e b tal que a domina b no grafo dado, o uso do algoritmo de Lengauer-Tarjan [Lengauer e Tarjan, 1979] é sugerido.

Neste trabalho, o grafo parcial $G'(A)$ é mantido pelo resolvedor SAT e os literais com relação de dominância neste grafo são propagados, para toda valoração parcial A explorada pelo resolvedor durante seu processo de busca (e não apenas para a valoração inicial vazia e o grafo $G'(\emptyset)$).

Além disso, o grafo parcial $G'(A)$ é mantido como uma estrutura interna do resolvedor SAT: a consistência parcial obtida não é proveniente de uma restrição polinomial da fórmula obtida. A investigação da possibilidade da criação de tal restrição é discutida como trabalho futuro no capítulo 6.

Primeiramente, o grafo $G'(\emptyset)$ é construído durante a inicialização do resolvedor SAT. Então, sempre que o resolvedor decide ou infere um literal na forma $\neg r_{a,b}$ (com $a, b \in V(G)$), o arco (a, b) é removido do grafo parcial atual. Além disso, sempre que o resolvedor desfaz a valoração realizada para este literal durante seu *backtracking* (cronológico ou não), o arco (a, b) é reinserido no grafo parcial atual.

Para determinar se algum vértice a domina algum outro vértice b (com relação ao mínimo m) em $G'(A)$, a árvore de dominantes $D(G'(A))$ é mantida. Note que o grafo é mantido com uma sequência de remoções e reinserções de seus arcos. Por isso, o grafo parcial pode ser

descrito como um grafo *dinâmico*. Logo, a árvore de dominantes $D(G'(A))$ pode ser mantida durante a busca por algoritmos que mantém a árvore de dominantes de grafos dinâmicos.

Georgiadis et al. [Georgiadis et al., 2012] apresentam dois algoritmos para manter a árvore de dominantes para grafos dinâmicos: o algoritmo *Dynamic SNCA* (Dynsnca) e o algoritmo *Depth-based search* (DBS). O algoritmo Dynsnca é uma adaptação do algoritmo SNCA [Georgiadis et al., 2004] para grafos dinâmicos, enquanto o algoritmo DBS é um algoritmo baseado em uma Busca em Profundidade [Georgiadis et al., 2012]. Ambos os algoritmos suportam, de maneira eficiente, as operações de remoção e inserção de arcos de um grafo dado.

Neste trabalho, implementações de ambos os algoritmos foram inseridas no resolvedor SAT Glucose [Audemard e Simon, 2009]. Este resolvedor foi utilizado por ser aberto e ter, portanto, seu código-fonte disponível. O resolvedor Glucose, por sua vez, é baseado no resolvedor MiniSAT [Eén e Sörensson, 2004], cujas estruturas são conhecidas pela comunidade.

Assim, sempre que um literal é propagado pelo resolvedor SAT, o grafo parcial atual $G'(A)$ e sua árvore de dominantes $D(G'(A))$ são atualizados. Após, verifica-se, na árvore de dominantes $D(G'(A))$, se existe algum par de vértices a e b tal que a é ancestral de b em $D(G'(A))$ e tal que a variável $r^+_{a,b}$ não foi valorada ainda. Neste caso, propaga-se o literal $\neg r^+_{a,b}$, e o procedimento de busca prossegue normalmente.

Alguns detalhes de implementação, como estruturas de dados utilizadas, são apresentados no apêndice A. A seção a seguir apresenta os experimentos realizados.

5.2 Resultados Experimentais

Esta seção apresenta e discute os experimentos realizados e os resultados obtidos.

Como apresentado na seção anterior, o resolvedor SAT Glucose [Audemard e Simon, 2009] foi incrementado com implementações dos algoritmos de Georgiadis et al. [Georgiadis et al., 2012] para manter, durante a busca do resolvedor, a árvore de dominantes do grafo parcial. A técnica foi experimentada com fórmulas obtidas pela codificação relativa de Prestwich [Prestwich, 2003] para o problema do Ciclo Hamiltoniano.

Foram codificados grafos de quatro classes de instâncias do problema do Ciclo Hamiltoniano. Estas classes são usadas por Vandergriend [Vandegriend, 1998] para estudar e apresentar a região de mudança de fase do problema do Ciclo Hamiltoniano.

A primeira classe de grafos codificados consiste em grafos *Degree-bounded*. Um grafo *Degree-bounded* é um grafo aleatório cujo grau de cada vértice é menor ou igual a 3. O grafo é gerado aleatoriamente dado, como parâmetro, o grau médio de todos os vértices do grafo. Para experimentar a técnica com instâncias na região de mudança de fase, foram utilizados grafos *Degree-bounded* com 200 vértices com grau médio variando de 2.6 a 3. Para cada grau médio neste intervalo, 100 instâncias do problema foram geradas. A tabela 5.1 indica a satisfabilidade destas instancias. A coluna "SAT" indica o número de instâncias satisfatíveis para o grau médio dado, enquanto a coluna "UNSAT" indica o número de instâncias insatisfatíveis para o mesmo.

A segunda classe de grafos experimentada consiste em grafos $G_{n,m}$. Um grafo $G_{n,m}$ é um grafo aleatório de n vértices e m arestas. O grafo desta classe é gerado dado o número de vértices n e um *parâmetro de grau* k. O número de arestas m é igual a $\bar{d}n/2$, onde $\bar{d} = k \times (lg(n) + lg(lg(n)))$. Também segundo Vandergriend [Vandegriend, 1998], a mudança de fase, para esta classe de grafos, com $n = 100$, consiste em grafos com parâmetros de grau entre 0.5 e 3. Para cada parâmetro de grau neste intervalo, foram gerados 10 grafos. A tabela 5.2 indica a satisfabilidade dos mesmos.

A terceira classe de grafos experimentada consiste em grafos *Crossroad*. Dado um número k de *componentes*, um grafo *Crossroad* é o grafo formado, informalmente, pela união

Tabela 5.1: Satisfabilidade de instâncias de grafos *Degree-bounded*

grau médio	SAT	UNSAT
2.60	0	100
2.65	0	100
2.70	0	100
2.75	1	99
2.76	6	94
2.77	8	92
2.78	15	85
2.79	22	78
2.80	31	69
2.81	34	66
2.82	41	59
2.83	52	47
2.84	54	46
2.85	72	27
2.86	71	29
2.87	66	31
2.88	88	12
2.89	84	16
2.90	88	12
2.95	100	0
3.00	100	0

Tabela 5.2: Satisfabilidade de instâncias de grafos $G_{n,m}$

parâmetro de grau	SAT	UNSAT
0.50	0	10
0.60	0	10
0.70	0	10
0.80	0	10
0.90	1	9
0.95	3	7
1.00	3	7
1.05	3	7
1.10	5	4
1.15	8	2
1.20	4	6
1.25	7	3
1.30	7	3
1.40	9	1
1.50	10	0
1.60	9	1
1.70	10	0
1.80	10	0
1.90	10	0
2.00	10	0
2.10	10	0
2.20	10	0
2.30	10	0
2.40	10	0
2.50	10	0
2.60	10	0
2.70	10	0
2.80	10	0
2.90	10	0
3.00	10	0

de k grafos CR_S dispostos em um ciclo Hamiltoniano. Um grafo CR_S, por sua vez, é um grafo específico construído por Vandergriend [Vandegriend, 1998] para avaliar a dificuldade da resolução do problema. A figura 5.3 apresenta o grafo *Crossroad* com $k = 9$ componentes. Todas as instâncias desta classe são satisfatíveis, uma vez que há um Ciclo Hamiltoniano em todo grafo *Crossroad*, como é possível deduzir pela figura 5.3.

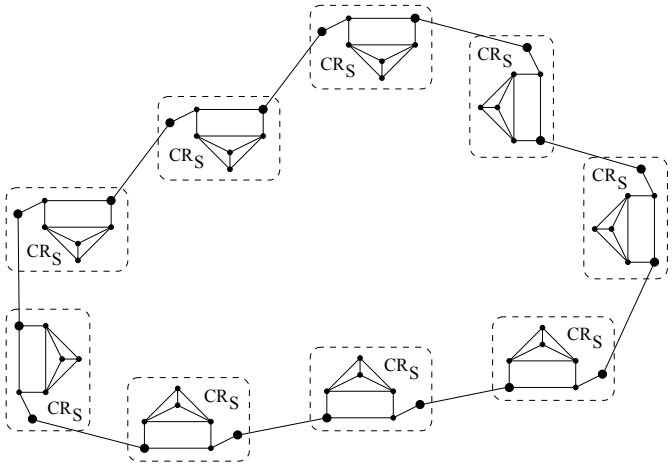

Figura 5.3: O grafo *Crossroad* com $k = 9$ componentes [Vandegriend, 1998]

Por fim, a quarta classe de grafos experimentados são grafos *Knight-Tour*. Dados inteiros n, m, r e c, o grafo *Knight-Tour n_m_r_c* é o grafo que representa os possíveis movimentos de um cavalo em um tabuleiro de xadrez, tal que o tabuleiro tem n linhas e m colunas, e, a cada movimento, o cavalo pode se deslocar, simultaneamente, por r linhas e c colunas ou vice-versa. Como exemplo, a figura 5.4 apresenta o grafo *Knight-Tour* 4_4_2_1, isto é, com $n = m = 4$, $r = 2$ e $c = 1$. Esta classe de grafos também é utilizada por Vandergriend [Vandegriend, 1998] para estudar a dificuldade do problema. A tabela 5.3 apresenta a satisfabilidade das instâncias utilizadas, onde "SAT" indica uma instância satisfatível, e "UNSAT" indica uma instância insatisfatível.

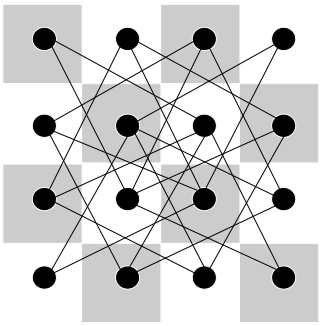

Figura 5.4: O grafo *Knight-Tour* 4_4_2_1

Todas as instâncias utilizadas neste trabalho podem ser obtidas em *http://www.inf.ufpr.br/rtoliveira/*.

Para medir a eficiência da técnica, foram observados: o número de literais propagados (inferidos ou decididos); o tempo de CPU total do resolvedor SAT; o número de decisões realizadas pelo resolvedor; e o número de conflitos encontrados pelo resolvedor durante sua busca.

Tabela 5.3: Satisfabilidade de instâncias de grafos *Knight-Tour*

Instância	Satisfabilidade
1_2_3_34	SAT
1_2_3_38	SAT
1_2_3_40	SAT
1_4_10_11	SAT
1_4_10_14	SAT
1_4_5_30	UNSAT
1_4_5_34	SAT
1_4_5_36	UNSAT
1_4_5_38	SAT
1_4_5_42	SAT
1_4_5_44	SAT
1_4_5_46	SAT
1_4_5_52	SAT
1_4_5_54	SAT
1_4_8_10	UNSAT
1_4_9_12	SAT
1_4_9_16	SAT
1_4_9_18	SAT
1_4_9_20	SAT
1_6_7_50	UNSAT
1_6_9_38	SAT
1_6_9_40	SAT
2_3_10_12	UNSAT
2_3_9_20	SAT
2_5_10_16	UNSAT
2_5_10_17	UNSAT
2_5_11_18	SAT
2_5_11_20	SAT
2_5_7_28	UNSAT
2_5_7_32	UNSAT
2_5_7_36	SAT
2_5_7_38	SAT
2_5_9_30	UNSAT
3_4_7_36	SAT

Os experimentos foram realizados em uma máquina *AMD Opteron(tm) Processor 6136 @ 2.4 Ghz, 120 Gb RAM, Linux 4.6.4*. As tabelas 5.4, 5.5, 5.6 e 5.7 e figuras 5.5, 5.6, 5.7 e 5.8 apresentam os resultados obtidos para os grafos *Degree-bounded*. As tabelas 5.8, 5.9, 5.10 e 5.11 e figuras 5.9, 5.10, 5.11 e 5.12, por sua vez, apresentam os resultados para grafos $G_{n,m}$. As tabelas 5.12, 5.13, 5.14 e 5.15 e figuras 5.13, 5.14, 5.15 e 5.16 apresentam os resultados obtidos para grafos *Crossroad*. Por fim, as tabelas 5.16, 5.17, 5.18 e 5.19 e figuras 5.17, 5.18, 5.19 e 5.20 apresentam os resultados para grafos *Knight-tour*.

Tabela 5.4: Número de propagações para grafos *Degree-bounded*

grau médio	Dynsnca	DBS	Nenhum
2.60	1150.88	1162.20	**509.38**
2.65	959.96	959.96	**944.78**
2.70	5497.41	5497.41	**1815.65**
2.75	16264.40	**16263.21**	16402.93
2.76	34225.11	31707.67	**31346.64**
2.77	**34816.38**	53788.42	57500.36
2.78	**197381.57**	198172.01	206349.08
2.79	117014.99	125707.40	**107445.50**
2.80	**74621.83**	82296.92	89513.66
2.81	**85921.27**	91310.53	96768.25
2.82	110836.16	**109472.58**	405827.91
2.83	138127.18	**130221.75**	138440.03
2.84	1256724.36	1232768.93	**1231956.71**
2.85	**208325.19**	230858.31	244145.37
2.86	214432.20	**206057.16**	237232.74
2.87	206034.86	**195486.70**	209517.63
2.88	312522.79	**307655.09**	307874.82
2.89	356311.37	360382.14	**280024.28**
2.90	375116.38	408131.03	**316066.38**
2.95	804788.25	726385.60	**469730.22**
3.00	1483598.48	1404542.13	**546617.65**

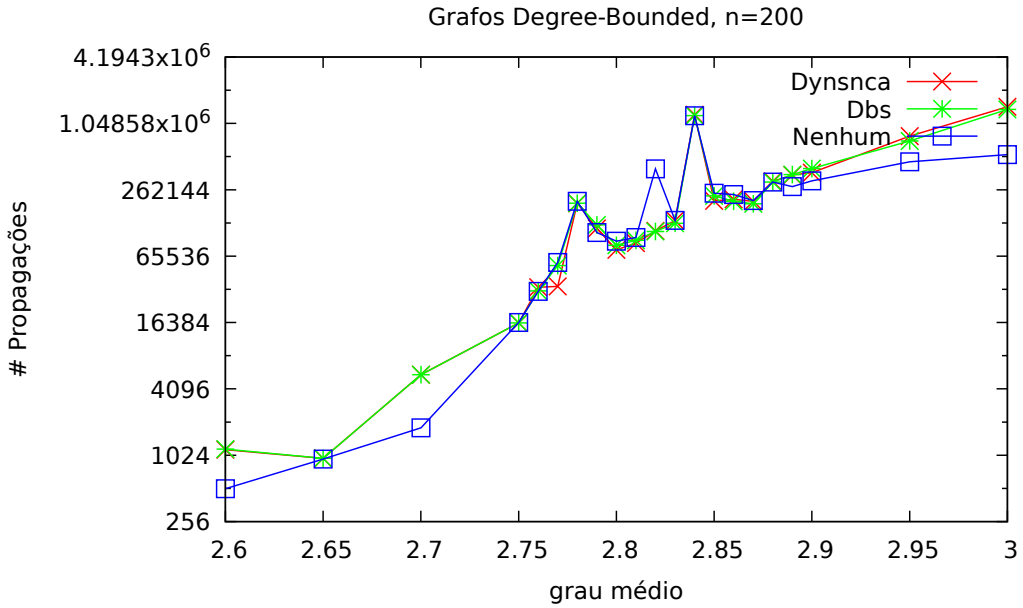

Figura 5.5: Número de propagações para grafos *Degree-bounded*

Tabela 5.5: Número de decisões para grafos *Degree-bounded*

grau médio	Dynsnca	DBS	Nenhum
2.60	**153.90**	160.21	171.84
2.65	**252.01**	**252.01**	294.87
2.70	**413.49**	**413.49**	451.16
2.75	**947.46**	**947.46**	969.41
2.76	927.44	**923.11**	981.38
2.77	**1054.68**	1080.07	1154.04
2.78	1266.72	**1259.39**	1342.95
2.79	**1441.64**	1464.12	1557.73
2.80	**1499.53**	1531.51	1613.99
2.81	**1475.94**	1483.00	1507.35
2.82	1798.80	**1796.54**	2211.28
2.83	1980.38	**1978.64**	2086.90
2.84	2781.24	**2729.36**	2867.11
2.85	**2504.95**	2518.49	2568.60
2.86	2443.30	**2415.96**	2656.81
2.87	2382.45	**2366.91**	2468.38
2.88	2922.08	**2917.65**	3018.26
2.89	**2865.31**	2876.22	2891.15
2.90	**2796.83**	2846.85	2893.95
2.95	3861.07	3834.45	**3614.51**
3.00	4803.38	4687.29	**3957.95**

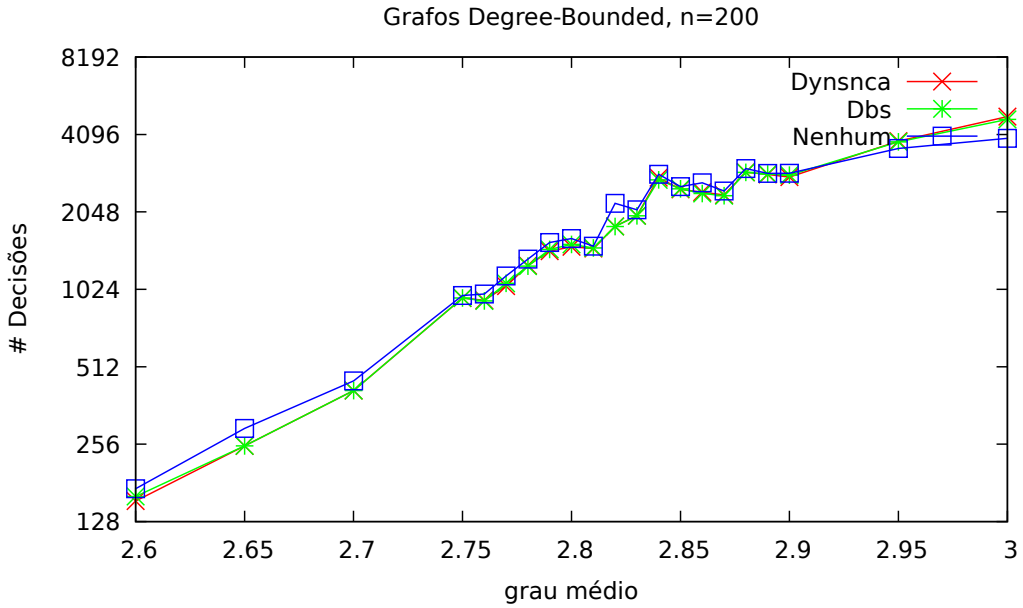

Figura 5.6: Número de decisões para grafos *Degree-bounded*

Tabela 5.6: Número de conflitos para grafos *Degree-bounded*

grau médio	Dynsnca	DBS	Nenhum
2.60	2.94	3.01	**2.63**
2.65	3.45	3.45	**3.33**
2.70	8.40	8.40	**4.86**
2.75	25.05	25.05	**24.56**
2.76	48.16	**44.59**	46.12
2.77	**50.81**	69.71	72.38
2.78	152.90	**151.04**	158.72
2.79	122.74	141.27	**122.51**
2.80	**89.98**	100.64	110.54
2.81	**107.45**	113.90	119.53
2.82	133.42	**130.98**	420.75
2.83	155.83	**149.56**	165.10
2.84	794.49	**759.45**	774.48
2.85	**228.92**	242.97	263.98
2.86	238.10	**222.70**	272.42
2.87	234.55	**217.17**	225.91
2.88	323.21	**319.58**	324.88
2.89	365.72	373.17	**298.54**
2.90	367.68	399.30	**324.14**
2.95	713.74	667.70	**452.84**
3.00	1226.33	1148.20	**509.37**

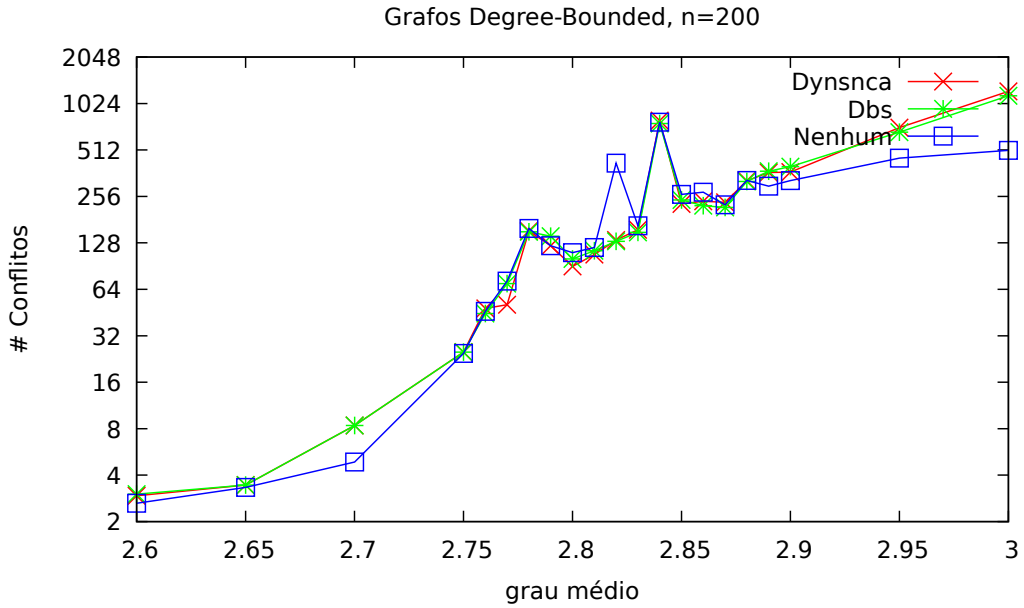

Figura 5.7: Número de conflitos para grafos *Degree-bounded*

Tabela 5.7: Tempo de CPU para grafos *Degree-bounded*

grau médio	Dynsnca	DBS	Nenhum
2.60	9.70	9.69	**9.65**
2.65	9.73	9.73	**9.69**
2.70	9.77	9.78	**9.75**
2.75	10.00	10.00	**9.94**
2.76	10.21	10.23	**10.11**
2.77	**10.23**	10.52	10.43
2.78	12.52	12.50	**12.25**
2.79	11.47	11.54	**11.09**
2.80	**10.82**	10.96	10.88
2.81	**10.98**	11.03	10.99
2.82	**11.35**	11.36	15.06
2.83	11.68	11.64	**11.56**
2.84	26.72	25.98	**25.74**
2.85	**12.81**	13.19	13.06
2.86	12.83	**12.70**	12.82
2.87	12.77	12.66	**12.51**
2.88	14.30	14.23	**13.92**
2.89	14.65	14.79	**13.38**
2.90	14.85	15.35	**13.65**
2.95	19.92	19.16	**15.60**
3.00	27.33	26.21	**16.51**

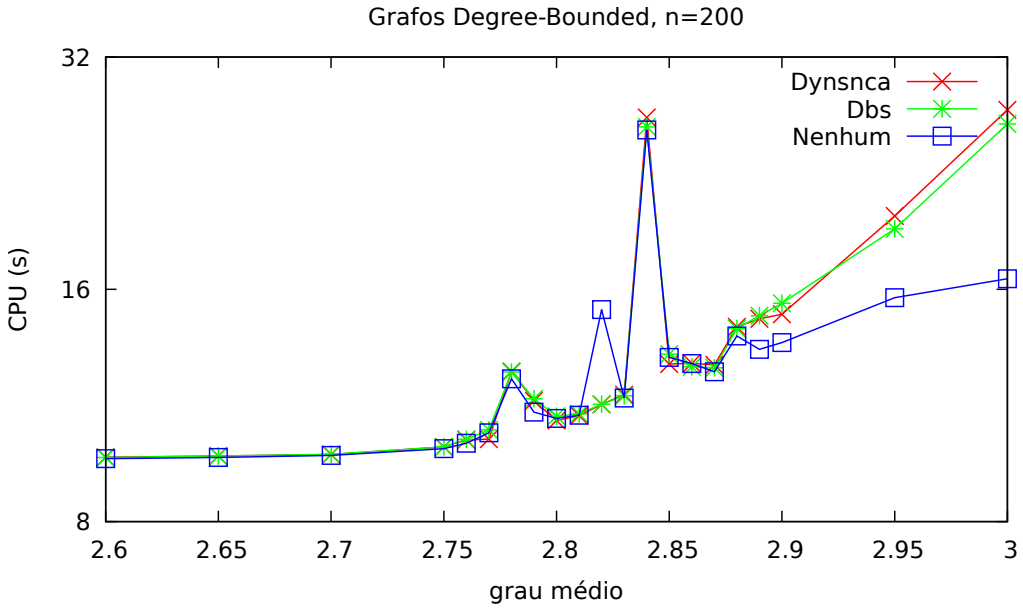

Figura 5.8: Tempo de CPU para grafos *Degree-bounded*

Tabela 5.8: Número de propagações para grafos $G_{n,m}$

parâmetro de grau	Dynsnca	DBS	Nenhum
0.50	3.75	3.75	**3.40**
0.60	2.44	2.44	**2.40**
0.70	**37.89**	**37.89**	55.30
0.80	**86.20**	**86.20**	**86.20**
0.90	82549.50	68329.11	**27968.90**
0.95	308521.67	**164919.10**	202815.20
1.00	378568.60	359563.33	**251745.90**
1.05	344177.60	**257562.50**	348931.60
1.10	761657.20	900545.44	**580361.56**
1.15	4398253.00	2553255.70	**2275115.80**
1.20	542620.80	**497760.90**	536704.60
1.25	3442085.30	2982991.80	**2752703.50**
1.30	3049992.10	**2591657.70**	2863670.30
1.40	**5374647.00**	7808214.80	7601278.70
1.50	14281100.90	**11306143.30**	13311434.70
1.60	**15242302.50**	15369905.80	17439966.30
1.70	**12258155.60**	13824748.10	17685544.20
1.80	11207649.70	8585115.60	**8426926.40**
1.90	**7119734.10**	13909272.20	15618342.20
2.00	**9708037.20**	13101227.80	11290547.00
2.10	10479541.80	**10297866.80**	12535993.40
2.20	**9993041.10**	11319146.60	15001090.50
2.30	12857886.30	12290424.00	**11658676.30**
2.40	8715603.80	**6541206.20**	7696268.80
2.50	7160859.30	**4707068.10**	5853189.40
2.60	10912570.00	10996829.80	**10722137.10**
2.70	8594680.30	11958190.00	**7943781.50**
2.80	10126787.90	**8508550.80**	10225635.90
2.90	**6022141.60**	7430291.40	10864311.40
3.00	7081231.10	10167515.70	**5586924.90**

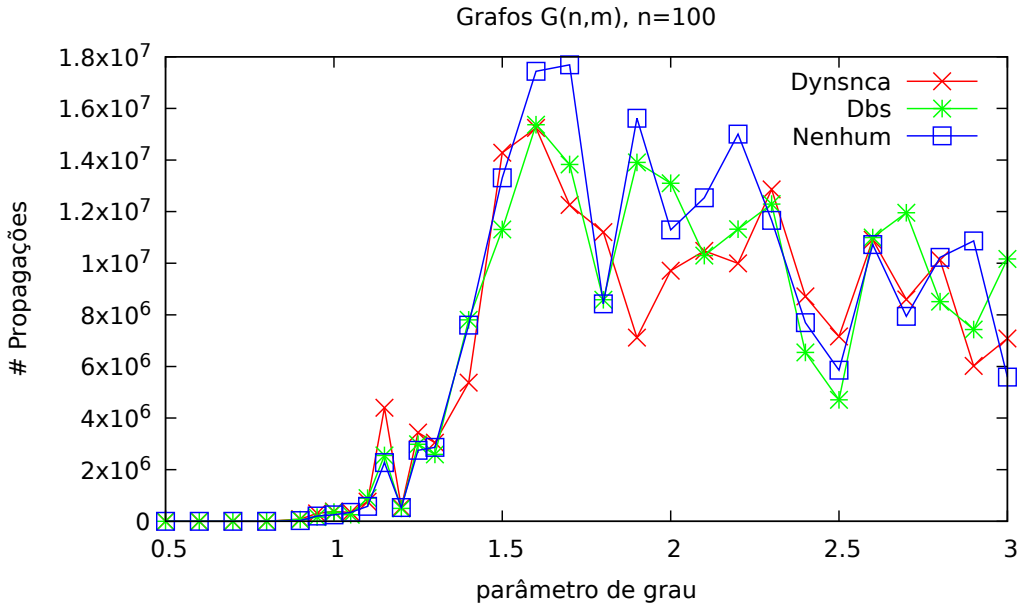

Figura 5.9: Número de propagações para grafos $G_{n,m}$

Tabela 5.9: Número de decisões para grafos $G_{n,m}$

parâmetro de grau	Dynsnca	DBS	Nenhum
0.50	**0.00**	**0.00**	**0.00**
0.60	**0.00**	**0.00**	**0.00**
0.70	**0.00**	**0.00**	**0.00**
0.80	**0.00**	**0.00**	**0.00**
0.90	560.80	501.89	**320.50**
0.95	1894.33	**1301.30**	1535.40
1.00	2465.00	2525.89	**2105.00**
1.05	2240.30	**1822.50**	2217.90
1.10	4644.70	5822.78	**4320.00**
1.15	24547.50	15782.00	**15157.20**
1.20	4045.60	**3672.80**	3676.60
1.25	21896.70	21645.00	**20322.60**
1.30	21547.00	**18825.60**	24530.00
1.40	**42081.40**	65208.70	64880.30
1.50	128304.50	**100699.80**	126857.10
1.60	160987.20	**157772.90**	182038.10
1.70	**138596.80**	151997.70	184717.60
1.80	139674.40	107420.30	**100991.60**
1.90	**96085.60**	160428.50	187843.40
2.00	**120047.50**	156728.40	132420.40
2.10	135271.00	**127481.20**	169782.10
2.20	**136423.90**	161624.80	205318.50
2.30	180888.50	189488.60	**161177.50**
2.40	124115.10	**89376.90**	105217.50
2.50	101243.80	**68767.50**	78827.40
2.60	152975.10	145486.50	**137160.10**
2.70	102598.80	118904.00	**96392.00**
2.80	149996.70	**114929.80**	139317.00
2.90	**96122.80**	117570.50	158478.50
3.00	106213.90	134536.00	**73257.20**

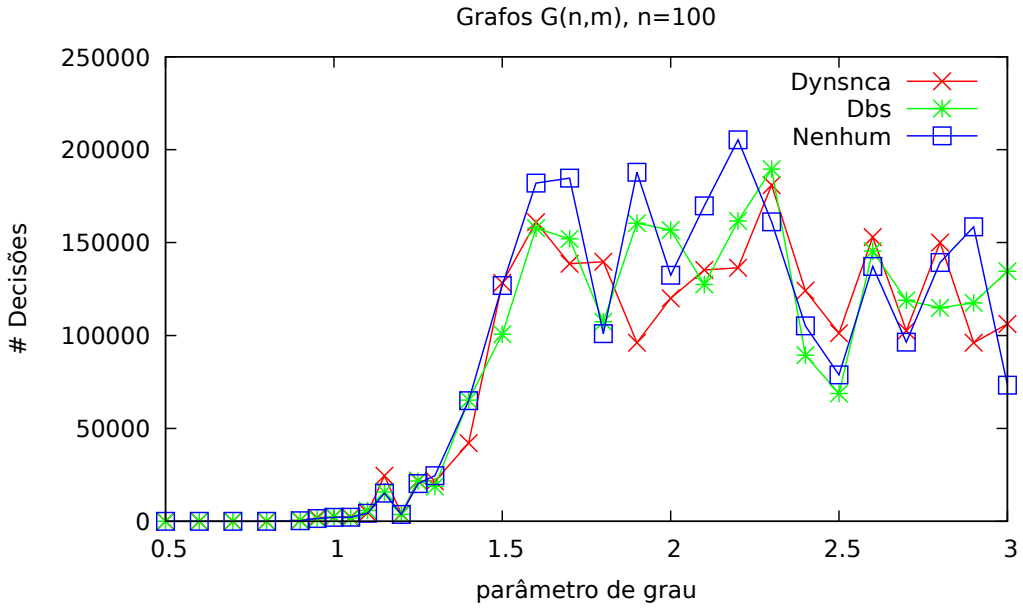

Figura 5.10: Número de decisões para grafos $G_{n,m}$

Tabela 5.10: Número de conflitos para grafos $G_{n,m}$

parâmetro de grau	Dynsnca	DBS	Nenhum
0.50	**0.00**	**0.00**	**0.00**
0.60	**0.00**	**0.00**	**0.00**
0.70	**0.00**	**0.00**	**0.00**
0.80	**0.00**	**0.00**	**0.00**
0.90	163.60	131.44	**68.90**
0.95	636.89	**341.40**	413.70
1.00	766.00	733.44	**521.70**
1.05	674.40	**531.20**	725.20
1.10	1608.10	1968.00	**1205.11**
1.15	9676.40	5852.30	**5195.80**
1.20	1228.10	**1090.30**	1224.70
1.25	7922.00	7190.70	**6752.70**
1.30	7599.40	**6833.90**	7410.90
1.40	**13931.00**	21325.30	21561.30
1.50	42968.10	**32581.60**	40927.40
1.60	50008.50	**49453.60**	57250.70
1.70	**41884.80**	46828.50	58922.80
1.80	39654.60	30371.30	**29375.50**
1.90	**24781.30**	50613.90	57585.20
2.00	**37678.80**	51576.00	42559.90
2.10	**39311.60**	39557.40	49102.30
2.20	**38892.50**	43092.90	59250.40
2.30	49168.30	47805.80	**44741.00**
2.40	33674.00	**25544.20**	30122.50
2.50	27281.30	**18774.40**	22860.30
2.60	43616.70	42746.40	**42043.60**
2.70	37344.70	50618.90	**33447.30**
2.80	42441.20	**35453.10**	44067.40
2.90	**24543.20**	30507.70	45998.90
3.00	29453.10	42185.30	**22629.40**

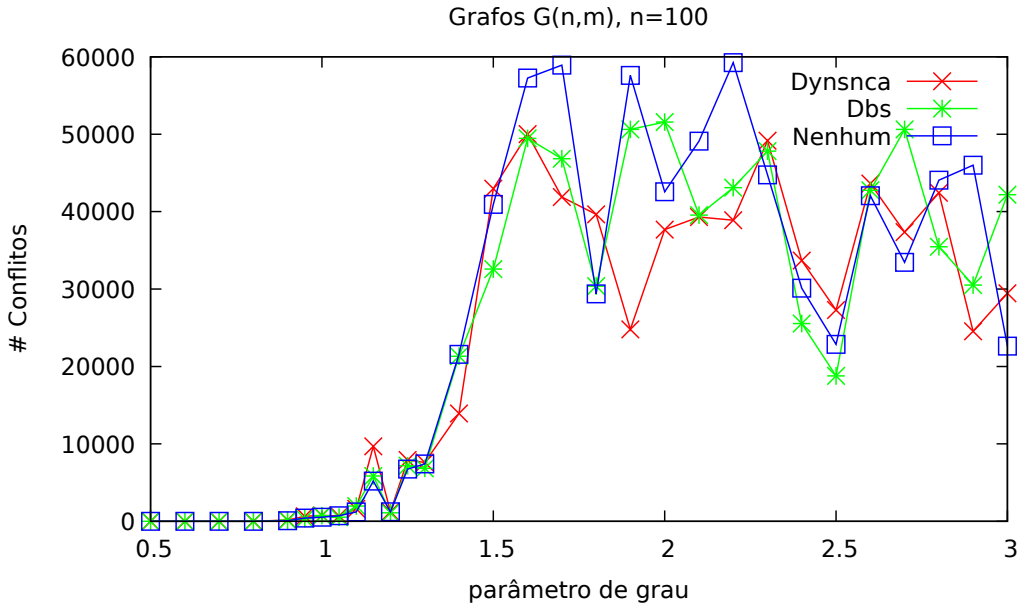

Figura 5.11: Número de conflitos para grafos $G_{n,m}$

Tabela 5.11: Tempo de CPU para grafos $G_{n,m}$

parâmetro de grau	Dynsnca	DBS	Nenhum
0.50	0.05	0.05	**0.04**
0.60	0.05	0.06	**0.04**
0.70	0.05	0.05	**0.04**
0.80	0.05	0.05	**0.04**
0.90	0.49	0.43	**0.20**
0.95	1.60	0.97	**0.85**
1.00	1.81	1.80	**0.99**
1.05	1.70	1.28	**1.17**
1.10	3.36	4.08	**2.07**
1.15	17.07	10.29	**6.13**
1.20	2.38	2.30	**1.52**
1.25	12.70	11.46	**6.92**
1.30	10.79	9.81	**7.00**
1.40	18.85	26.92	**16.36**
1.50	46.28	39.13	**25.50**
1.60	48.66	53.70	**28.83**
1.70	38.87	47.21	**28.58**
1.80	35.25	30.65	**12.84**
1.90	22.33	50.00	**21.40**
2.00	30.28	48.00	**14.46**
2.10	27.74	32.79	**13.81**
2.20	27.11	38.49	**15.50**
2.30	35.88	44.33	**11.64**
2.40	24.87	24.16	**7.78**
2.50	21.06	17.80	**6.24**
2.60	32.95	43.20	**10.46**
2.70	28.16	57.10	**7.66**
2.80	31.57	36.70	**9.73**
2.90	20.92	34.37	**10.15**
3.00	24.58	50.35	**5.53**

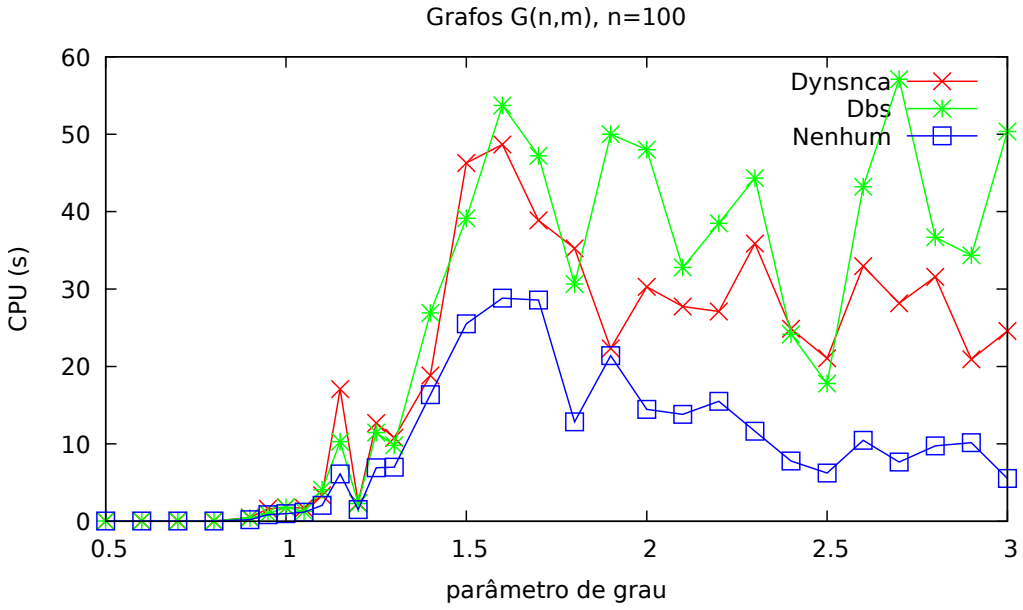

Figura 5.12: Tempo de CPU para grafos $G_{n,m}$

Tabela 5.12: Número de propagações para grafos *Crossroad*

Componentes (K)	Dynsnca	DBS	Nenhum
7	**2856.00**	**2856.00**	4650.00
8	**2202.00**	**2202.00**	6919.00
9	**7907.00**	9487.00	8990.00
10	25942.00	16388.00	**14915.00**
11	20020.00	**17988.00**	24133.00
12	84164.00	63160.00	**38231.00**
13	**19790.00**	19908.00	34747.00
14	**17937.00**	54540.00	30151.00
15	**26346.00**	**26346.00**	70381.00
16	**18296.00**	**18296.00**	37129.00
17	**17125.00**	**17125.00**	44582.00
18	**18806.00**	**18806.00**	45223.00
19	**21232.00**	22126.00	67856.00
20	152098.00	103444.00	**67371.00**

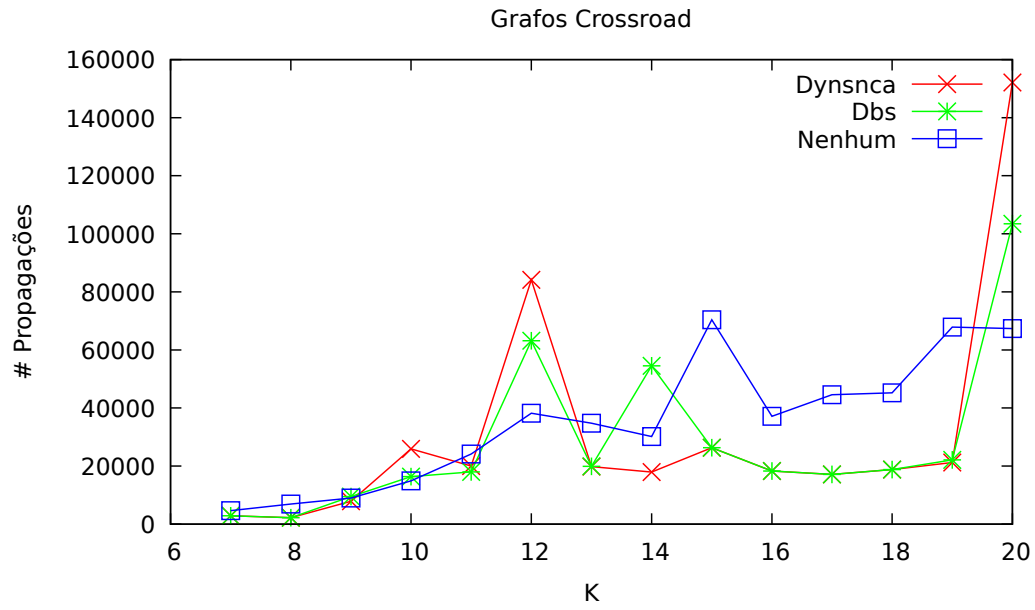

Figura 5.13: Número de propagações para grafos *Crossroad*

Tabela 5.13: Número de decisões para grafos *Crossroad*

Componentes (K)	Dynsnca	DBS	Nenhum
7	**116.00**	**116.00**	233.00
8	**52.00**	**52.00**	471.00
9	**171.00**	199.00	327.00
10	**210.00**	213.00	406.00
11	**319.00**	362.00	679.00
12	503.00	**468.00**	1081.00
13	**370.00**	421.00	877.00
14	**496.00**	999.00	1335.00
15	**327.00**	**327.00**	1645.00
16	**427.00**	**427.00**	1165.00
17	**276.00**	**276.00**	1092.00
18	**606.00**	**606.00**	1448.00
19	382.00	**379.00**	2325.00
20	2154.00	**2078.00**	2819.00

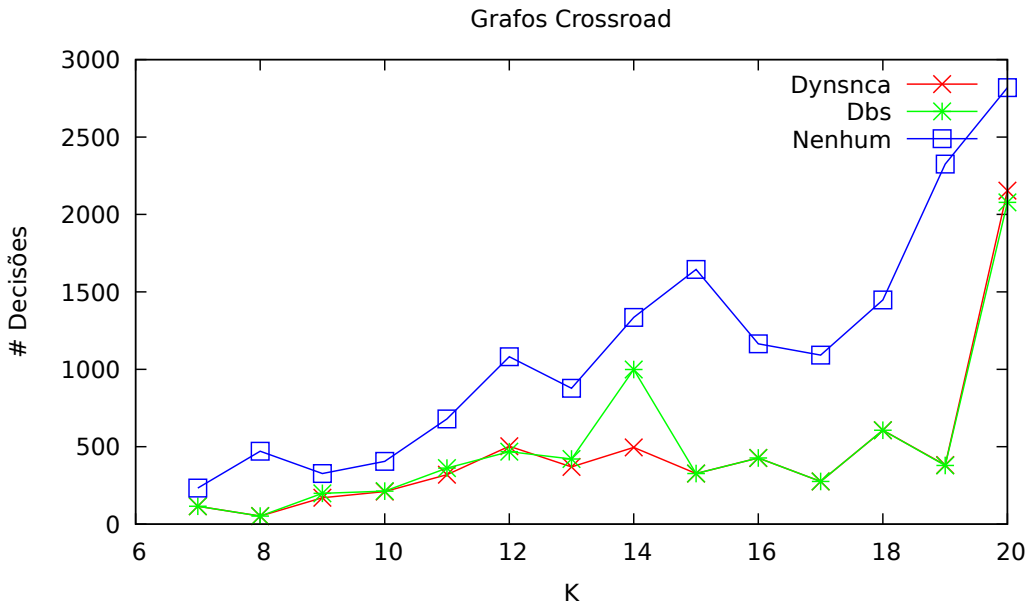

Figura 5.14: Número de decisões para grafos *Crossroad*

Tabela 5.14: Número de conflitos para grafos *Crossroad*

Componentes (K)	Dynsnca	DBS	Nenhum
7	**15.00**	**15.00**	38.00
8	**9.00**	**9.00**	53.00
9	**30.00**	48.00	61.00
10	129.00	66.00	**62.00**
11	**82.00**	93.00	84.00
12	346.00	310.00	**121.00**
13	73.00	**69.00**	158.00
14	**35.00**	169.00	91.00
15	**141.00**	**141.00**	203.00
16	**50.00**	**50.00**	97.00
17	**34.00**	**34.00**	102.00
18	**43.00**	**43.00**	129.00
19	**35.00**	**35.00**	184.00
20	332.00	273.00	**140.00**

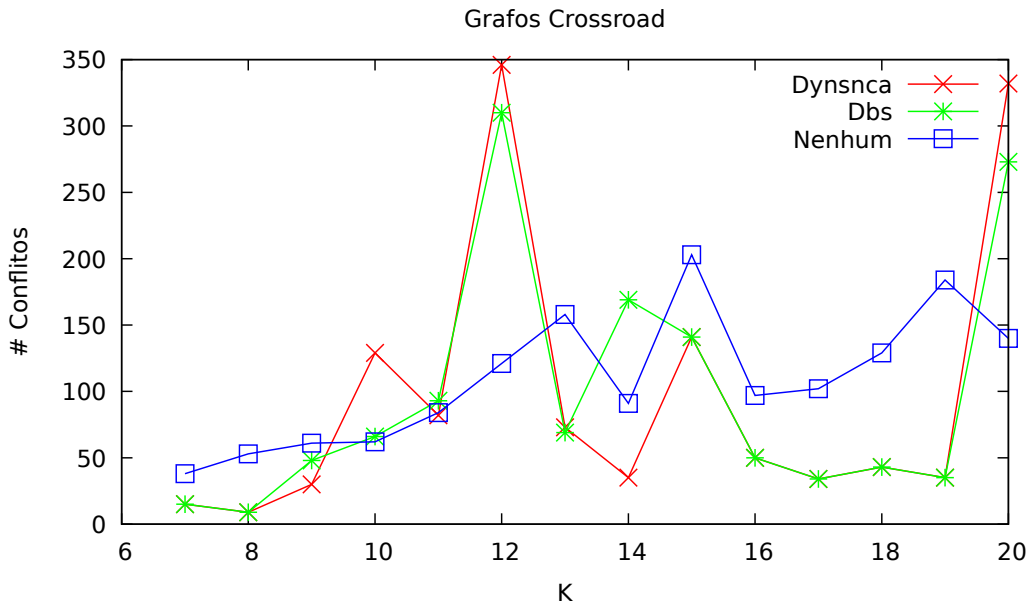

Figura 5.15: Número de conflitos para grafos *Crossroad*

Tabela 5.15: Tempo de CPU para grafos *Crossroad*

Componentes (K)	Dynsnca	DBS	Nenhum
7	0.07	0.05	**0.05**
8	0.09	0.09	**0.09**
9	0.14	0.15	**0.14**
10	0.26	0.26	**0.23**
11	0.34	0.36	**0.30**
12	0.83	0.78	**0.48**
13	0.66	0.65	**0.58**
14	**0.77**	1.05	0.77
15	1.05	1.04	**1.00**
16	**1.13**	1.14	1.16
17	1.40	**1.40**	1.43
18	**1.73**	1.73	1.78
19	2.17	**2.10**	2.28
20	4.15	3.94	**2.65**

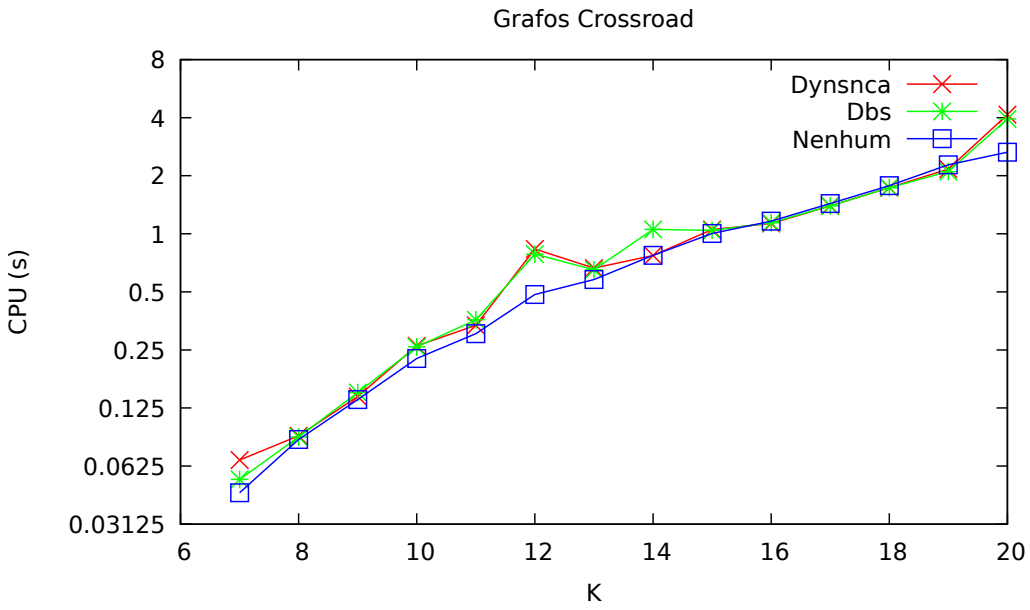

Figura 5.16: Tempo de CPU para grafos *Crossroad*

Tabela 5.16: Número de propagações para grafos *Knight-Tour*

Instância	Dynsnca	DBS	Nenhum
1_2_3_34	1308737.00	897901.00	**805686.00**
1_2_3_38	**326110.00**	**326110.00**	7078223.00
1_2_3_40	1675302.00	3520516.00	**75769.00**
1_4_10_11	247958.00	**246377.00**	851414.00
1_4_10_14	**424899.00**	**424899.00**	617653.00
1_4_5_30	1434262.00	**547898.00**	909972.00
1_4_5_34	1227294.00	**758706.00**	35727238.00
1_4_5_36	**3512718.00**	3802439.00	5318230.00
1_4_5_38	**1000195.00**	1390623.00	1835597.00
1_4_5_42	2508884.00	2943648.00	**917395.00**
1_4_5_44	8953950.00	**1339339.00**	8565189.00
1_4_5_46	22563204.00	**10662018.00**	67158949.00
1_4_5_54	**21876373.00**	41076260.00	133797639.00
1_4_9_12	87620.00	87620.00	**72050.00**
1_4_9_16	1010907.00	1010907.00	**640256.00**
1_4_9_18	1174604.00	1174604.00	**637639.00**
1_4_9_20	**867743.00**	**867743.00**	10834678.00
1_6_9_38	11998329.00	**10987972.00**	49119546.00
1_6_9_40	**6104910.00**	37966460.00	124250315.00
2_3_10_12	**1087270.00**	1153925.00	1127089.00
2_3_9_20	406372.00	**128430.00**	130893.00
2_5_10_16	6209.00	**6124.00**	6747.00
2_5_10_17	57636.00	57636.00	**44412.00**
2_5_11_18	**323691.00**	1181518.00	513966.00
2_5_11_20	**666501.00**	**666501.00**	28828259.00
2_5_7_28	**501603.00**	543122.00	648957.00
2_5_7_32	17818427.00	8363145.00	**7153049.00**
2_5_7_36	**606665.00**	3811923.00	1346295.00
2_5_7_38	**286236.00**	**286236.00**	678806.00
2_5_9_30	24124.00	24124.00	**23169.00**
3_4_7_36	1705913.00	4703918.00	**1266850.00**

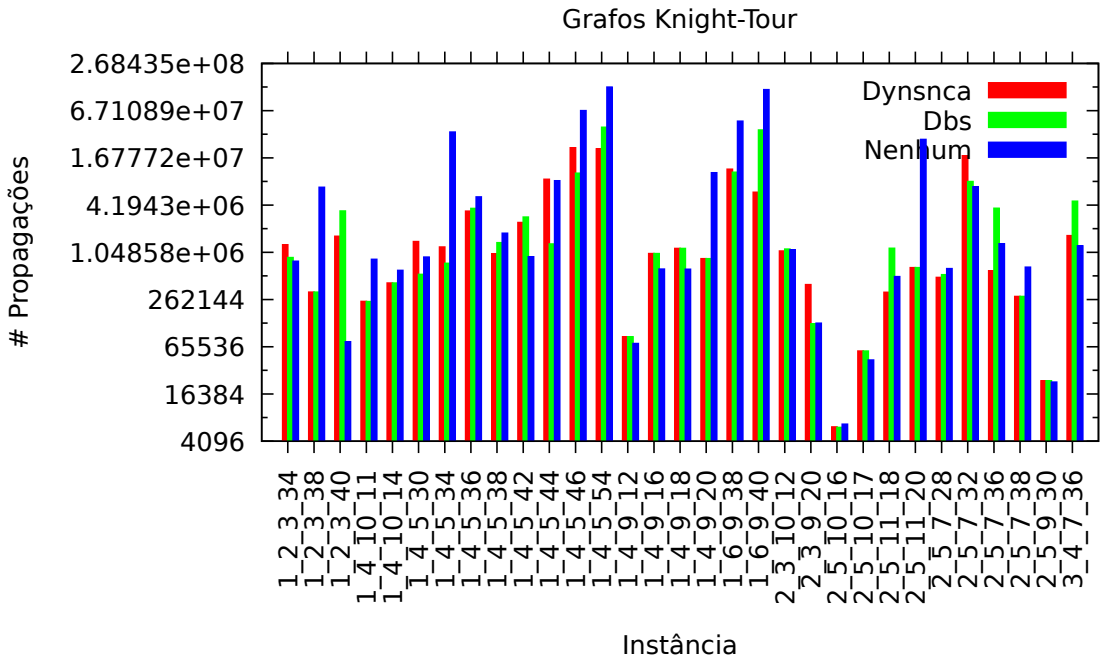

Figura 5.17: Número de propagações para grafos *Knight-Tour*

Tabela 5.17: Número de decisões para grafos *Knight-Tour*

Instância	Dynsnca	DBS	Nenhum
1_2_3_34	7437.00	5293.00	**4661.00**
1_2_3_38	**3325.00**	**3325.00**	28564.00
1_2_3_40	9513.00	12213.00	**2886.00**
1_4_10_11	**2369.00**	**2369.00**	4554.00
1_4_10_14	**4645.00**	**4645.00**	4790.00
1_4_5_30	10280.00	**7567.00**	12945.00
1_4_5_34	5963.00	**5265.00**	48091.00
1_4_5_36	**38039.00**	43042.00	56338.00
1_4_5_38	**6865.00**	7403.00	7550.00
1_4_5_42	9909.00	10685.00	**6502.00**
1_4_5_44	23621.00	**7863.00**	28118.00
1_4_5_46	50477.00	**27466.00**	123627.00
1_4_5_54	**30276.00**	44936.00	252465.00
1_4_9_12	1595.00	1595.00	**1250.00**
1_4_9_16	**3181.00**	**3181.00**	3229.00
1_4_9_18	6017.00	6017.00	**4128.00**
1_4_9_20	**6846.00**	**6846.00**	37625.00
1_6_9_38	**13366.00**	21457.00	87672.00
1_6_9_40	**15555.00**	49692.00	213270.00
2_3_10_12	**9986.00**	10598.00	11001.00
2_3_9_20	6456.00	**5469.00**	6110.00
2_5_10_16	1892.00	1892.00	**1443.00**
2_5_10_17	2680.00	2680.00	**2395.00**
2_5_11_18	**4000.00**	5498.00	5452.00
2_5_11_20	**4354.00**	**4354.00**	53285.00
2_5_7_28	**5419.00**	5488.00	7185.00
2_5_7_32	**52091.00**	52793.00	59818.00
2_5_7_36	**6328.00**	7534.00	9554.00
2_5_7_38	11438.00	11438.00	**10571.00**
2_5_9_30	**7171.00**	**7171.00**	**7171.00**
3_4_7_36	21477.00	23507.00	**18464.00**

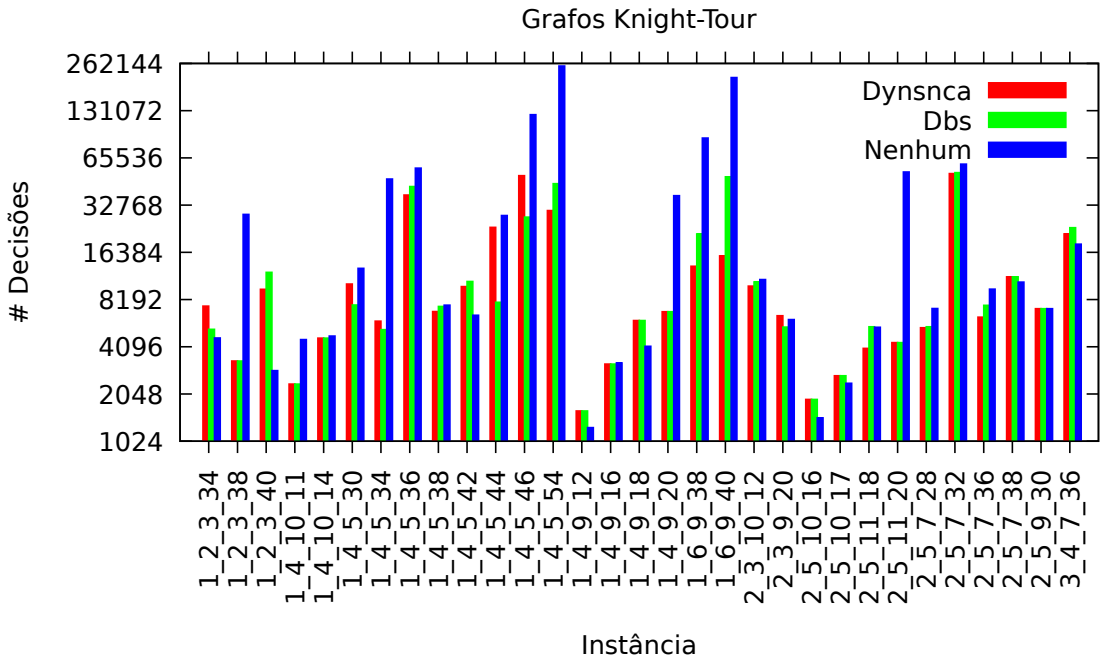

Figura 5.18: Número de decisões para grafos *Knight-Tour*

Tabela 5.18: Número de conflitos para grafos *Knight-Tour*

Instância	Dynsnca	DBS	Nenhum
1_2_3_34	3687.00	2242.00	**2017.00**
1_2_3_38	**718.00**	**718.00**	17007.00
1_2_3_40	4805.00	6931.00	**204.00**
1_4_10_11	**730.00**	732.00	2184.00
1_4_10_14	**706.00**	**706.00**	1168.00
1_4_5_30	6984.00	**4669.00**	9041.00
1_4_5_34	1584.00	**1314.00**	28735.00
1_4_5_36	**27617.00**	31381.00	41639.00
1_4_5_38	**1792.00**	2154.00	2578.00
1_4_5_42	2019.00	2308.00	**672.00**
1_4_5_44	8870.00	**1264.00**	8783.00
1_4_5_46	19640.00	**8621.00**	57758.00
1_4_5_54	**8180.00**	18750.00	98400.00
1_4_9_12	402.00	402.00	**154.00**
1_4_9_16	1282.00	1282.00	**985.00**
1_4_9_18	1494.00	1494.00	**737.00**
1_4_9_20	**769.00**	**769.00**	10995.00
1_6_9_38	**2102.00**	3184.00	20393.00
1_6_9_40	**1212.00**	8673.00	49430.00
2_3_10_12	**6318.00**	6837.00	7032.00
2_3_9_20	369.00	**122.00**	183.00
2_5_10_16	16.00	**15.00**	40.00
2_5_10_17	345.00	345.00	**338.00**
2_5_11_18	**489.00**	1042.00	620.00
2_5_11_20	**404.00**	**404.00**	19509.00
2_5_7_28	**1594.00**	1651.00	2520.00
2_5_7_32	**37126.00**	39068.00	42118.00
2_5_7_36	**292.00**	1301.00	906.00
2_5_7_38	**158.00**	**158.00**	319.00
2_5_9_30	**64.00**	**64.00**	**64.00**
3_4_7_36	**356.00**	1355.00	582.00

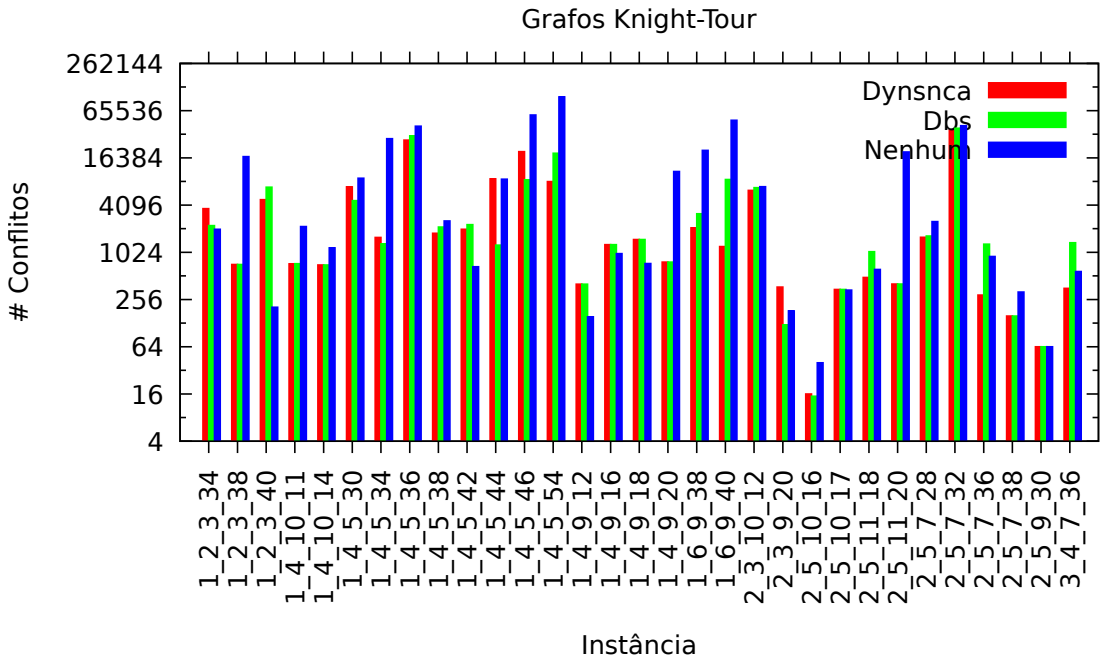

Figura 5.19: Número de conflitos para grafos *Knight-Tour*

Tabela 5.19: Tempo de CPU para grafos *Knight-Tour*

Instância	Dynsnca	DBS	Nenhum
1_2_3_34	8.90	7.05	**4.54**
1_2_3_38	**3.84**	3.90	24.70
1_2_3_40	12.24	22.28	**3.23**
1_4_10_11	**4.59**	4.61	6.45
1_4_10_14	10.59	**10.10**	10.39
1_4_5_30	23.15	14.35	**10.13**
1_4_5_34	18.92	**14.85**	347.33
1_4_5_36	44.66	55.18	**42.66**
1_4_5_38	**31.25**	35.62	38.30
1_4_5_42	69.86	74.77	**38.56**
1_4_5_44	169.70	**53.89**	159.81
1_4_5_46	392.77	**217.29**	922.81
1_4_5_54	**175.79**	257.27	1072.61
1_4_9_12	1.28	1.27	**1.06**
1_4_9_16	7.32	7.45	**4.76**
1_4_9_18	9.73	9.85	**7.15**
1_4_9_20	12.80	**12.55**	55.93
1_6_9_38	**132.54**	143.62	620.17
1_6_9_40	**77.90**	413.02	1627.11
2_3_10_12	6.21	7.49	**2.99**
2_3_9_20	10.17	7.22	**7.15**
2_5_10_16	4.06	4.08	**4.01**
2_5_10_17	5.30	5.48	**5.08**
2_5_11_18	**11.91**	16.75	12.96
2_5_11_20	**18.88**	19.50	216.48
2_5_7_28	12.43	13.66	**11.14**
2_5_7_32	120.17	92.98	**33.88**
2_5_7_36	**9.40**	28.94	18.98
2_5_7_38	6.72	**6.52**	11.04
2_5_9_30	4.09	3.92	**3.77**
3_4_7_36	15.62	41.85	**15.61**

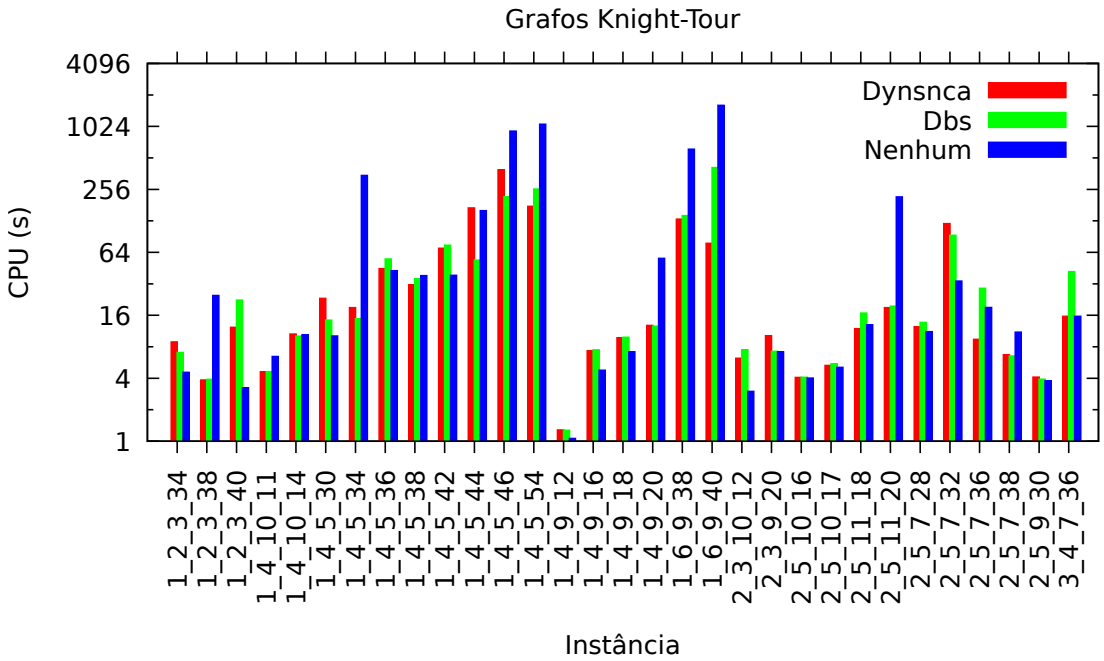

Figura 5.20: Tempo de CPU para grafos *Knight-Tour*

Tabela 5.20: p-valores obtidos pelo teste de Friedman

	CPU	Decisões	Conflitos	Propagações
Degree-bounded	**0.0302**	**0.00000002**	0.2258	0.2215
$G_{n,m}$	**0.00002**	0.558	0.3679	0.7788
Crossroad	**0.009**	**0.0009**	0.5866	0.5866
Knight-Tour	0.3679	**0.04069**	**0.02135**	0.5134

Para cada classe de instâncias experimentadas, é possível observar algumas características preliminares dos resultados obtidos. Para grafos *Degree-bounded*, é possível notar, primeiramente, a alta correlação entre o número de propagações realizadas e de conflitos encontrados pelo resolvedor para todas as técnicas experimentadas, como observado pelos gráficos nas figuras 5.5 e 5.7. Esta correlação também pode ser notada para grafos $G_{n,m}$, como observado pelo gráficos nas figuras 5.9 e 5.11. Esta relação pode indicar que ambas as métricas poderiam ser utilizadas para mensurar a exploração do espaço de busca do resolvedor.

A análise das tabelas 5.4 e 5.7 pode indicar que o número de propagações e o tempo total de CPU são menores sem a manutenção parcial de consistência para grafos *Degree-bounded* no início e no final da região da mudança de fase, enquanto tais medidas são maiores com a manutenção parcial de consistência em tais regiões. Isto pode indicar que a manutenção parcial pode ter mais impacto na resolução de instâncias dentro da região da mudança de fase do problema. Entretanto, de acordo com a tabela 5.5, o número de decisões foi reduzido para a maioria das instâncias, como esperado.

Para instâncias de grafos $G_{n,m}$, a tabela 5.11 indica que o tempo de CPU levado pelo resolvedor SAT foi menor sem a manutenção da consistência parcial para todas as instâncias experimentadas, indicando uma possível sobrecarga gerada pelo algoritmo implementado no resolvedor. As tabelas 5.8, 5.9 e 5.10, entretanto, não permitem concluir uma melhora ou piora do desempenho do resolvedor com a utilização da técnica experimentada.

Contrariamente, para instâncias de grafos *Crossroad*, é possível observar diferenças de desempenho do resolvedor com a manutenção parcial ou não da consistência da fórmula obtida. Como pode ser observado pelas tabelas 5.12 e 5.14, a manutenção parcial da consistência reduziu o número de propagações realizadas e de conflitos encontrados pelo resolvedor para a maioria das instâncias experimentadas. De acordo com a tabela 5.13, a manutenção parcial da consistência também reduziu o número de decisões realizadas pelo resolvedor para todas as instâncias experimentadas. A tabela 5.15 indica que os tempos de CPU foram menores sem a manutenção parcial da consistência para a maioria das instâncias experimentadas, embora tais tempos podem ser considerados insignificantes devido a seus baixos valores.

Por fim, não é possível, através da análise das tabelas 5.16, 5.17, 5.18 e 5.19, observar tais diferenças para instâncias de grafos *Knight-tour*.

Entretanto, pra determinar para quais casos foi observada diferença estatística significante nos resultados experimentais obtidos, o teste de Friedman [Demsar, 2006, Iman e Davenport, 1980] foi utilizado. A tabela 5.20 apresenta os p-valores obtidos pelo teste. Valores em negrito indicam os casos em que o p-valor foi inferior a 0.05 e, portanto, foi observada diferença estatística.

A análise da tabela 5.20 permite observar que houve diferença estatística significativa para o tempo de CPU levado pelo resolvedor SAT para resolver instâncias de grafos *Degree-bounded*, $G_{n,m}$ e *Crossroad*. Também houve diferença estatística para o número de decisões

Tabela 5.21: Pós-teste de Nemenyi, Degree-bounded × CPU

	Dynsnca	DBS	Avg. Rank
Dynsnca	-	-	2.03
DBS	0.485	-	2.29
Nenhum	0.295	**0.023**	1.68

Tabela 5.22: Pós-teste de Nemenyi, $G_{n,m}$ × CPU

	Dynsnca	DBS	Avg. Rank
Dynsnca	-	-	2.5
DBS	1	-	2.5
Nenhum	**0.00021**	**0.00021**	1

Tabela 5.23: Pós-teste de Nemenyi, Crossroad × CPU

	Dynsnca	DBS	Avg. Rank
Dynsnca	-	-	2.5
DBS	0.966	-	2.38
Nenhum	**0.016**	**0.033**	1.12

Tabela 5.24: Pós-teste de Nemenyi, Degree-bounded × Decisões

	Dynsnca	DBS	Avg. Rank
Dynsnca	-	-	1.84
DBS	0.97	-	1.89
Nenhum	**0.000001**	**0.000004**	2.71

feitas pelo resolvedor para grafos *Degree-bounded*, *Knight-Tour* e *Crossroad*, assim como para o numero de conflitos para grafos *Knight-Tour*. Não observa-se diferença estatística para o número de propagações realizadas, embora este fato pode ser um reflexo da diferença estatística no número de decisões, como discutido a seguir.

As tabelas 5.21, 5.22, 5.23, 5.24, 5.25, 5.26 e 5.27 apresentam o resultado do pós-teste de Nemenyi [Nemenyi, 1963] para os casos em que observou-se diferença estatística.

Ao analisar as tabelas 5.24, 5.25 e 5.26, é possível observar que a manutenção da árvore de dominantes do grafo parcial reduziu, com sucesso, o número de decisões realizadas pelo resolvedor SAT para determinar a satisfabilidade da fórmula dada, para grafos *Degree-bounded*, *Knight-Tour* e *Crossroad*. Este fato pode ser um reflexo do nível parcial de consistência mantida pela técnica. Observa-se que decisões são realizadas apenas após a aplicação da Propagação Unitária, e, logo, tais decisões são realizadas apenas quando o resolvedor não pode mais inferir o valor verdade de alguma(s) variável(is) através deste procedimento. Quando o nível de

Tabela 5.25: Pós-teste de Nemenyi, Knight-tour × Decisões

	Dynsnca	DBS	Avg. Rank
Dynsnca	-	-	1.94
DBS	0.647	-	2.16
Nenhum	**0.049**	0.31	2.35

Tabela 5.26: Pós-teste de Nemenyi, Crossroad × Decisões

	Dynsnca	DBS	Avg. Rank
Dynsnca	-	-	1.38
DBS	0.5768	-	1.88
Nenhum	**0.0014**	**0.0332**	3

Tabela 5.27: Pós-teste de Nemenyi, Knight-tour × Conflitos

	Dynsnca	DBS	Avg. Rank
Dynsnca	-	-	1.87
DBS	0.567	-	2.13
Nenhum	**0.025**	0.251	2.39

consistência da fórmula é aumentado, o valor verdade de algumas variáveis que seriam decididas é implicado, o que reduz o número de decisões realizadas durante o processo.

A tabela 5.27 também indica que, como um todo, o número de conflitos foi reduzido com o uso do algoritmo Dynsnca para instâncias com grafos *Knight-Tour*. Este fato também pode ser um reflexo da consistência parcial mantida, uma vez que a propagação de literais realizada pela análise da árvore de dominantes pode cortar subespaços de busca inviáveis.

Entretanto, as tabelas 5.21, 5.22 e 5.23 indicam que o tempo total de CPU levado pelo resolvedor SAT para resolver instâncias com grafos *Degree-bounded*, $G_{n,m}$ e *Crossroad* é significantemente maior quando a técnica apresentada é utilizada. Como apontado anteriormente, isto pode indicar que, embora a técnica reduz o número de decisões ou conflitos encontrados durante a busca, o custo computacional criado ao introduzir algoritmos nativos dentro do resolvedor SAT pode não compensar o tempo total de CPU levado para o término do processo. É possível esperar que este tempo seja reduzido se, ao invés de embutir algoritmos dentro resolvedor, a fórmula obtida seja polinomialmente restrita com uma codificação "booleana" da árvore de dominantes. O estudo desta possibilidade é apresentado no capítulo 6 como um possível trabalho futuro.

Por fim, a tabela 5.20 não apresenta diferença estatística significante para o número de propagações realizadas pelo resolvedor. Entretanto, este fato pode ser causado pelos possivelmente diferentes subespaços de busca explorados pelo resolvedor SAT, como sugerido pelas reduções do número de decisões e conflitos encontrados pelo mesmo. Também há indícios de que os subespaços de busca explorados pelo resolvedor utilizando o algoritmo DBS podem ser diferentes dos subespaços explorados pelo mesmo, utilizando o algoritmo Dynsnca, o que pode sugerir que

os procedimentos adicionados ao resolvedor podem ter influenciado algumas heurísticas internas utilizadas pelo mesmo.

5.3 Considerações finais

Este capítulo apresenta a técnica utilizada neste trabalho para criar um nível parcial de consistência mantida por um resolvedor SAT em codificações relativas. A técnica consiste na implementação de algoritmos de manutenção da árvore de dominantes para grafos dinâmicos dentro do resolvedor SAT.

Como apresentado na seção anterior, a técnica reduziu com sucesso o número de decisões e conflitos encontrados pelo resolvedor SAT para classes de instâncias na mudança de fase do problema original. Entretanto, o tempo total de CPU levado pelo resolvedor não foi reduzido. Técnicas alternativas de implementação do método são discutidas como trabalho futuro no capítulo 6.

Capítulo 6

Conclusão e Trabalhos Futuros

Como apresentado no capítulo 1, este trabalho tem como principais contribuições o estudo sobre a manutenção da Arco Consistência Generalizada em codificações SAT relativas, e a manutenção de um nível parcial de consistência através da manutenção da árvore de dominantes do grafo induzido durante a busca. Além disso, são também apresentadas codificações relativas propostas pelos autores, um *framework* genérico para codificações relativas, um algoritmo polinomial para SAT para instâncias mantidas ACG, e a alteração de um resolvedor SAT no estado-da-arte em seu código fonte.

Como apresentado no capítulo 4, fórmulas obtidas por codificações relativas que não contém cláusulas *at-least* e *at-most* são mantidas ACG pela Propagação Unitária. Além disso, há fórmulas que contém estas cláusulas que não são mantidas ACG por este procedimento, e conjectura-se que tais fórmulas podem não ser polinomialmente restritas a outras mantidas ACG, no caso geral.

Os seguintes artigos publicados pelos autores fazem parte, direta ou indiretamente, deste trabalho:

- [de Oliveira et al., 2012] Ricardo Tavares de Oliveira, Fabiano Silva, Bruno Ribas, e Marcos Castilho. *On Modeling Connectedness in Reductions from Graph Problems to Extended Satisfiability*. Ibero-American Conference on Artificial Intelligence (IBERAMIA), 2012. Este trabalho apresenta codificações para o problema de conectividade através de operadores de cardinalidade, que são aplicadas ao problema da Árvore de Steiner;

- [Oliveira e Silva, 2014] Ricardo Tavares de Oliveira e Fabiano Silva. *SAT and MaxSAT Encodings for Trees Applied to the Steiner Tree Problem*. Brazilian Conference on Intelligent Systems (BRACIS), 2014. Este trabalho apresenta diversas codificações para o problema da Árvore de Steiner, incluindo a codificação relativa por árvore;

- [de Oliveira e Silva, 2015] Ricardo Tavares de Oliveira and Fabiano Silva. *On a Relative MaxSAT Encoding for the Steiner Tree Problem in Graphs*. Mexican International Conference on Artificial Intelligence (MICAI), 2015. Este trabalho apresenta aprimoramentos à codificação por árvore para o problema da Árvore de Steiner, em particular o uso da técnica de Bryant & Velev [Bryant e Velev, 2002] e da árvore de dominantes do grafo dado, como um pré-processamento.

Além disso, as principais contribuições deste trabalho são apresentadas no artigo *On the Generalized Arc Consistency on Relative SAT Encodings* de Oliveira & Silva, submetido em Novembro de 2016 à revista *Discrete Applied Mathematics*, atualmente sob análise.

O fato de que a conjunção de duas subfórmulas mantidas ACG pode não ser mantida ACG pode induzir ao estudo da dificuldade computacional necessária para unir duas fórmulas mantidas ACG em uma única fórmula mantida ACG, possivelmente maior, dependendo da estrutura de ambas as subfórmulas. Como motivação para este estudo, determinar a satisfabilidade da conjunção de fórmulas booleanas $2-SAT$ e fórmulas booleanas *Horn* (cujas satisfabilidade podem ser determinadas em tempo polinomial) tem custo teórico de $O(2^{0.5284n})$, onde n é o número total de variáveis na fórmula [Porschen e Speckenmeyer, 2007]. Embora exponencial, o custo é menor que $O(2^n)$, o que pode sugerir que a conjunção de fórmulas mantidas ACG pode ser resolvida, embora exponencialmente, de maneira mais eficiente que as aplicadas comumente pelos resolvedores SAT atuais. Este estudo é apresentado como possível trabalho futuro.

Além disso, como apresentado no capítulo 5, a manutenção da árvore de dominantes durante o processo de busca de um resolvedor SAT, utilizada para manter um nível parcial de consistência da fórmula, de fato reduz o número decisões feitas por um resolvedor para várias instâncias experimentadas. Entretanto, o tempo total de CPU levado pelo resolvedor para resolver estas instâncias também sofreu um aumento com o uso da técnica. Também é válido observar que a árvore de dominantes é mantida dentro do resolvedor SAT, e que a fórmula obtida pela codificação não é polinomialmente restrita a outra contendo as cláusulas unitárias com os literais propagados pela técnica. Por estes motivos sugere-se, também como trabalho futuro, o estudo da existência de uma restrição polinomial das fórmulas obtidas para outras fórmulas de tal forma que a Propagação Unitária propague um literal sempre que um dado vértice domina outro na respectiva árvore de dominantes.

Referências Bibliográficas

[Abío et al., 2012] Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E. e Mayer-Eichberger, V. (2012). A new look at bdds for pseudo-boolean constraints. *Journal of Artificial Intelligence Research*, 45(1):443–480.

[Ansótegui et al., 2009] Ansótegui, C., Bonet, M. L. e Levy, J. (2009). Solving (weighted) partial maxsat through satisfiability testing. volume 5584, páginas 427–440. Springer-Verlag, Springer-Verlag.

[Aspvall et al., 1979] Aspvall, B., Plass, M. F. e Tarjan, R. E. (1979). A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121 – 123.

[Audemard e Simon, 2009] Audemard, G. e Simon, L. (2009). Predicting learnt clauses quality in modern sat solvers. Em *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, páginas 399–404, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Bailleux e Boufkhad, 2003] Bailleux, O. e Boufkhad, Y. (2003). Efficient cnf encoding of boolean cardinality constraints. Em Rossi, F., editor, *Principles and Practice of Constraint Programming – CP 2003*, volume 2833 de *Lecture Notes in Computer Science*, páginas 108–122. Springer Berlin Heidelberg.

[Bailleux et al., 2009] Bailleux, O., Boufkhad, Y. e Roussel, O. (2009). New encodings of pseudo-boolean constraints into cnf. Em Kullmann, O., editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 de *Lecture Notes in Computer Science*, páginas 181–194. Springer Berlin Heidelberg.

[Bessière et al., 2002] Bessière, C., Meseguer, P., Freuder, E. C. e Larrosa, J. (2002). On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141(1–2):205 – 224.

[Biere, 2013] Biere, A. (2013). Lingeling, plingeling and treengeling entering the sat competition 2013. Em *Proceedings of SAT Competition 2013*, volume B-2013-1. University of Helsinki.

[Björk, 2009] Björk, M. (2009). Successful sat encoding techniques. *JSAT Addendum, Julho*.

[Bryant e Velev, 2002] Bryant, R. E. e Velev, M. N. (2002). Boolean satisfiability with transitivity constraints. *ACM Trans. Comput. Logic*, 3(4):604–627.

[Buro e Büning, 1993] Buro, M. e Büning, H. K. (1993). Report on a sat competition. *Bulletin of the European Association for Theoretical Computer Science*, 49:143–151.

[Buss e Pitassi, 1998] Buss, S. R. e Pitassi, T. (1998). Resolution and the weak pigeonhole principle. Em *Selected Papers from the11th International Workshop on Computer Science Logic*, CSL '97, páginas 149–156, London, UK, UK. Springer-Verlag.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. Em *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, páginas 151–158, New York, NY, USA. ACM.

[Davis et al., 1962] Davis, M., Logemann, G. e Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5:394–397.

[De Kleer, 1989] De Kleer, J. (1989). A comparison of atms and csp techniques. Em *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, páginas 290–296, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[de Oliveira, 2013] de Oliveira, R. T. (2013). Reduções de problemas em grafos com soluções conexas para (max)sat e adaptação de um resolvedor sat e maxsat não clausal para as instâncias obtidas. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, Brasil.

[de Oliveira e Silva, 2015] de Oliveira, R. T. e Silva, F. (2015). On a relative maxsat encoding for the steiner tree problem in graphs. Em Pichardo Lagunas, O., Herrera Alcántara, O. e Arroyo Figueroa, G., editores, *Advances in Artificial Intelligence and Its Applications: 14th Mexican International Conference on Artificial Intelligence, MICAI 2015, Cuernavaca, Morelos, Mexico, October 25-31, 2015, Proceedings, Part II*, páginas 422–434. Springer International Publishing.

[de Oliveira et al., 2012] de Oliveira, R. T., Silva, F., Ribas, B. e Castilho, M. (2012). On modeling connectedness in reductions from graph problems to extended satisfiability. Em Pavón, J., Duque-Méndez, N. e Fuentes-Fernández, R., editores, *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637 de *Lecture Notes in Computer Science*, páginas 381–391. Springer Berlin Heidelberg.

[Demsar, 2006] Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

[Di Rosa et al., 2010] Di Rosa, E., Giunchiglia, E. e Maratea, M. (2010). Solving satisfiability problems with preferences. *Constraints*, 15:485–515.

[Dowling e Gallier, 1984] Dowling, W. F. e Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267 – 284.

[Eén e Sörensson, 2004] Eén, N. e Sörensson, N. (2004). An extensible sat-solver. Em Giunchiglia, E. e Tacchella, A., editores, *Theory and Applications of Satisfiability Testing*, volume 2919 de *Lecture Notes in Computer Science*, páginas 502–518. Springer Berlin Heidelberg.

[Ford e Fulkerson,] Ford, L. R. e Fulkerson, D. R. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8:399–404.

[Freeman, 1995] Freeman, J. W. (1995). *Improvements To Propositional Satisfiability Search Algorithms*. Tese de doutorado, University of Pennsylvania.

[Fu e Malik, 2006] Fu, Z. e Malik, S. (2006). On solving the partial max-sat problem. Em *In International Conference on Theory and Applications of Satisfiability Testing (SAT'06), LNCS 4121*, páginas 252–265. Springer.

[Gelder, 2008] Gelder, A. V. (2008). Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156(2):230 – 243. Computational Methods for Graph Coloring and its Generalizations.

[Gent, 2002] Gent, I. P. (2002). Arc consistency in sat. Em *in Proceedings of ECAI 2002*, páginas 121–125. IOS Press.

[Georgiadis et al., 2012] Georgiadis, L., Italiano, G., Laura, L. e Santaroni, F. (2012). An experimental study of dynamic dominators. Em Epstein, L. e Ferragina, P., editores, *Algorithms – ESA 2012*, volume 7501 de *Lecture Notes in Computer Science*, páginas 491–502. Springer Berlin Heidelberg.

[Georgiadis et al., 2004] Georgiadis, L., Werneck, R. f., Tarjan, R. e., Triantafyllis, S. e August, D. i. (2004). *finding dominators in practice*, páginas 677–688. springer berlin heidelberg, berlin, heidelberg.

[Heras e Larrosa, 2006] Heras, F. e Larrosa, J. (2006). New inference rules for efficient max-sat solving. *Proceedings of the 21st national conference on Artificial intelligence*, 1:68–73. AAAI Press.

[Heras et al., 2008] Heras, F., Larrosa, J. e Oliveras, A. (2008). Minimaxsat: An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research*, 31:1–32.

[Hertel et al., 2007] Hertel, A., Hertel, P. e Urquhart, A. (2007). Formalizing dangerous sat encodings. Em *Proceedings of the 10th international conference on Theory and applications of satisfiability testing*, SAT'07, páginas 159–172. Springer-Verlag.

[Huang, 2007] Huang, J. (2007). The effect of restarts on the efficiency of clause learning. Em *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, páginas 2318–2323, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Iman e Davenport, 1980] Iman, R. L. e Davenport, J. M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in Statistics - Theory and Methods*, 9(6):571–595.

[Iwama e Miyazaki, 1994] Iwama, K. e Miyazaki, S. (1994). Sat-variable complexity of hard combinatorial problems. Em *in Proceedings of the World Computer Congress of the IFIP*, páginas 253–258. Elsevier Science B.V.

[Jeroslow e Wang, 1990] Jeroslow, R. G. e Wang, J. (1990). Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, (1):167–187.

[Karp, 1972] Karp, R. (1972). Reducibility among combinatorial problems. Em Miller, R., Thatcher, J. e Bohlinger, J., editores, *Complexity of Computer Computations*, The IBM Research Symposia Series, páginas 85–103. Springer US.

[Kasif, 1990] Kasif, S. (1990). On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artif. Intell.*, 45(3):275–286.

[Kautz et al., 1996] Kautz, H., Selman, B. e Jiang, Y. (1996). A general stochastic approach to solving problems with hard and soft constraints. Em *The Satisfiability Problem: Theory and Applications*, páginas 573–586. American Mathematical Society.

[Klieber e Kwon, 2007] Klieber, W. e Kwon, G. (2007). Efficient cnf encoding for selecting 1 from n objects. Em *International Workshop on Constraints in Formal Verification*.

[Larrosa et al., 2008] Larrosa, J., Heras, F. e de Givry, S. (2008). A logical approach to efficient max-sat solving. *ARTIFICIAL INTELLIGENCE*, 172:204–233. ACM.

[Lengauer e Tarjan, 1979] Lengauer, T. e Tarjan, R. E. (1979). A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141.

[Lenhardt, 2010] Lenhardt, R. (2010). Proof of concept: Fast solutions to np-problems by using SAT and integer programming solvers. *CoRR*, abs/1011.5447.

[Li e Quan, 2010] Li, C. M. e Quan, Z. (2010). An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. Em *AAAI*, volume 10, páginas 128–133.

[Mackworth, 1977] Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8(1):99 – 118.

[Marques-Silva e Sakallah, 1996] Marques-Silva, J. P. e Sakallah, K. A. (1996). Grasp - a new search algorithm for satisfiability. Em *in Proceedings of the International Conference on Computer-Aided Design*, páginas 220–227.

[Moskewicz et al., 2001] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L. e Malik, S. (2001). Chaff: Engineering an efficient sat solver. Em *ANNUAL ACM IEEE DESIGN AUTOMATION CONFERENCE*, páginas 530–535. ACM.

[Muller e Preparata, 1975] Muller, D. E. e Preparata, F. (1975). Bounds to complexities of networks for sorting and for switching. *Journal of the ACM*, 22:195 – 201.

[Nemenyi, 1963] Nemenyi, P. (1963). *Distribution-free Multiple Comparisons*. Princeton University.

[Oliveira e Silva, 2014] Oliveira, R. T. d. e Silva, F. (2014). Sat and maxsat encodings for trees applied to the steiner tree problem. Em *Brazilian Conference on Intelligent Systems (BRACIS), 2014*, páginas 192–197.

[Plaisted e Greenbaum, 1986] Plaisted, D. A. e Greenbaum, S. (1986). A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293 – 304.

[Porschen e Speckenmeyer, 2007] Porschen, S. e Speckenmeyer, E. (2007). Satisfiability of mixed horn formulas. *Discrete Applied Mathematics*, 155(11):1408 – 1419.

[Prestwich, 2003] Prestwich, S. (2003). Sat problems with chains of dependent variables. *Discrete Applied Mathematics*, 130(2):329–350.

[Prestwich, 2009] Prestwich, S. (2009). Cnf encodings. Em *Handbook of Satisfiability*, páginas 75–93.

[Samer e Veith, 2009] Samer, M. e Veith, H. (2009). Encoding treewidth into sat. Em Kullmann, O., editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 de *Lecture Notes in Computer Science*, páginas 45–50. Springer Berlin Heidelberg.

[Selman et al., 1992] Selman, B., Levesque, H. e Mitchell, D. (1992). A new method for solving hard satisfiability problems. *National Conference on Articifial Intelligence*, páginas 440–446. AAAI Press.

[Sinz, 2005] Sinz, C. (2005). Towards an optimal cnf encoding of boolean cardinality constraints. Em *In Proc. of the 11th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2005)*, páginas 827–831.

[Smyth et al., 2003] Smyth, K., Hoos, H. H. e Stützle, T. (2003). Iterated robust tabu search for max-sat. *In Proc. of the 16th Conf. of the Canadian Society for Computational Studies of Intelligence*, 2671:129–144. Springer.

[Tseitin, 1968] Tseitin, G. S. (1968). On the complexity of derivation in propositional calculus. *Structures in Constructive Mathematics and Mathematical Logic*, 2:115–125.

[Vandegriend, 1998] Vandegriend, B. (1998). Finding hamiltonian cycles: Algorithms, graphs and performance. Dissertação de Mestrado, University of Alberta, Edmonton, Alberta.

[Velev, 2007] Velev, M. N. (2007). Exploiting hierarchy and structure to efficiently solve graph coloring as sat. Em *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, páginas 135–142.

[Velev e Gao, 2009] Velev, M. N. e Gao, P. (2009). Efficient sat techniques for absolute encoding of permutation problems: Application to hamiltonian cycles. Em *Symposium on Abstraction, Reformulation, and Approximation (SARA)*. AAAI.

[Velev e Gao, 2010] Velev, M. N. e Gao, P. (2010). Design of parallel portfolios for sat-based solving of hamiltonian cycle problems. Em *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2010), Fort Lauderdale, Florida, USA, January 6-8, 2010*.

[Wallace e Freuder, 1996] Wallace, R. e Freuder, E. C. (1996). Comparative studies of constraint satisfaction and davis-putnam algorithms for maximum satisfiability problems. *Cliques, Coloring and Satisfiability*, páginas 587–615. American Mathematical Society.

[Walsh, 2000] Walsh, T. (2000). Sat v csp. Em *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP-00)*, páginas 441–456. Springer-Verlag.

[Yannakakis, 1981] Yannakakis, M. (1981). Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79.

[Zhang, 1997] Zhang, H. (1997). Sato: An efficient propositional prover. Em *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, páginas 272–275.

Apêndice A

Implementação realizada no resolvedor SAT Glucose

Este apêndice apresenta alguns detalhes breves da implementação dos algoritmos para manutenção da árvore de dominantes no resolvedor SAT Glucose [Audemard e Simon, 2009], citada no capítulo 5.

O resolvedor Glucose [Audemard e Simon, 2009] é baseado no resolvedor MiniSAT [Eén e Sörensson, 2004]. Desta forma, o resolvedor Glucose tem, como procedimentos internos, as mesmas funções e estruturas de dados utilizados pelo MiniSAT. O código-fonte do resolvedor MiniSAT, por sua vez, é aberto, e suas funções e estruturas são apresentadas em [Eén e Sörensson, 2004].

Primeiramente, o resolvedor recebe como entrada um aquivo representando a fórmula booleana dada, em CNF, no formato DIMACS. A primeira linha deste arquivo contém a descrição `p cnf <n> <m>`, onde `<n>` indica n, o número de variáveis, e `<m>` indica m, o número de cláusulas. As variáveis são numeradas de 1 a n. As próximas m linhas descrevem uma cláusula cada. Cada cláusula é descrita por uma lista de inteiros, separados por um espaço, terminada em 0. Um inteiro positivo indica um literal positivo na cláusula, enquanto um inteiro negativo indica um literal negativo na mesma. O valor absoluto do inteiro indica sua variável.

Além do arquivo em formato CNF, o resolvedor foi alterado para também receber um segundo arquivo, chamado *arquivo de mapa*. O arquivo de mapa consiste em uma descrição da *semântica* de cada variável na fórmula. Para codificações relativas, a semântica de uma variável indica se a mesma representa uma relação de R ou de R^+ (seu fecho transitivo, representado no arquivo por T), e também seus pares de elementos. Por exemplo, se a variável 1 consiste na variável $r_{1,2}$ e a variável 2 consiste na variável $r^+_{2,3}$, então o arquivo de mapa contém as seguintes linhas:

```
1 R 1 2
2 T 2 3
```

Com este arquivo, é possível associar cada variável da fórmula de entrada a sua aresta do grafo original dado correspondente. O arquivo é construído pelo mesmo procedimento que realiza a redução do problema original para SAT.

Durante a inicialização do resolvedor, as informações do arquivo de mapa são carregadas em vetores e matrizes, de tal forma que consultar a semântica de uma variável tem custo $O(1)$, assim como determinar qual variável tem uma semântica determinada.

Também durante a inicialização do resolvedor, o grafo parcial inicial $G'(\emptyset)$ é construído. O grafo, por sua vez, é representado internamente por suas listas de adjacências, de tal forma que

a (re)inserção de um arco tem custo $O(1)$, enquanto a remoção de um arco pode ter custo linear nos graus de seus vértices. A árvore de dominantes, por sua vez, é representada por um vetor de $|V(G)|$ posições contendo os dominantes imediatos de cada vértice do grafo. Estas estruturas são as utilizadas pelos algoritmos de Georgiadis et al. [Georgiadis et al., 2012].

A função `uncheckedEnqueue(Lit p, CRef from)` do resolvedor MiniSAT é responsável por enfileirar um literal p a ser propagado, como apresentado nas linhas 2 e 8 do algoritmo 1 no capítulo 2. Ao ser chamada, verifica-se se a semântica de p corresponde a um literal na forma $\neg r_{a,b}$ e, em caso positivo, o arco (a, b) é removido do grafo parcial atual e a árvore de dominantes, se necessário, é atualizada pelos algoritmos implementados. Uma *flag* global indica se houve alguma mudança na árvore neste passo.

A função `propagate()`, por sua vez, é responsável por propagar os literais enfileirados anteriormente (linhas 4 e 6 do algoritmo 1). *Após* a propagação de todos os literais obtidos pela Propagação Unitária, e caso a *flag* indique mudanças recentes na árvore de dominantes, uma busca na árvore é realizada. A busca consiste em, para cada vértice b, percorrer seu caminho até a raiz da árvore e, logo, enumerar seus ancestrais na mesma. Para cada ancestral a, verifica-se se a variável de semântica $r^{+}_{a,b}$ está valorada, e, em caso negativo, tal literal é propagado, utilizando para tal as funções internas do resolvedor. A busca na árvore, no pior caso, pode ter custo quadrático no número de seus vértices.

Caso um conflito seja encontrado com os literais propagados desta maneira (e não pela Propagação Unitária) na função `search(nof_conflicts)`, o último nível de decisão é desfeito, uma cláusula contendo a negação dos literais decididos é aprendida, e um retrocesso cronológico é realizado.

Por fim, a função `cancelUntil(level)` é responsável pelos retrocessos (cronológicos e não cronológicos) do resolvedor. Quando um nível de decisão é desfeito por esta função, todas as variáveis valoradas naquele nível são desvaloradas. Para cada literal desvalorado, verifica-se se sua semântica corresponde a um literal na forma $\neg r_{a,b}$. Em caso positivo, o arco (a, b) é reinserido no grafo parcial atual, e a árvore de dominantes é novamente atualizada pelos algoritmos implementados, se necessário.