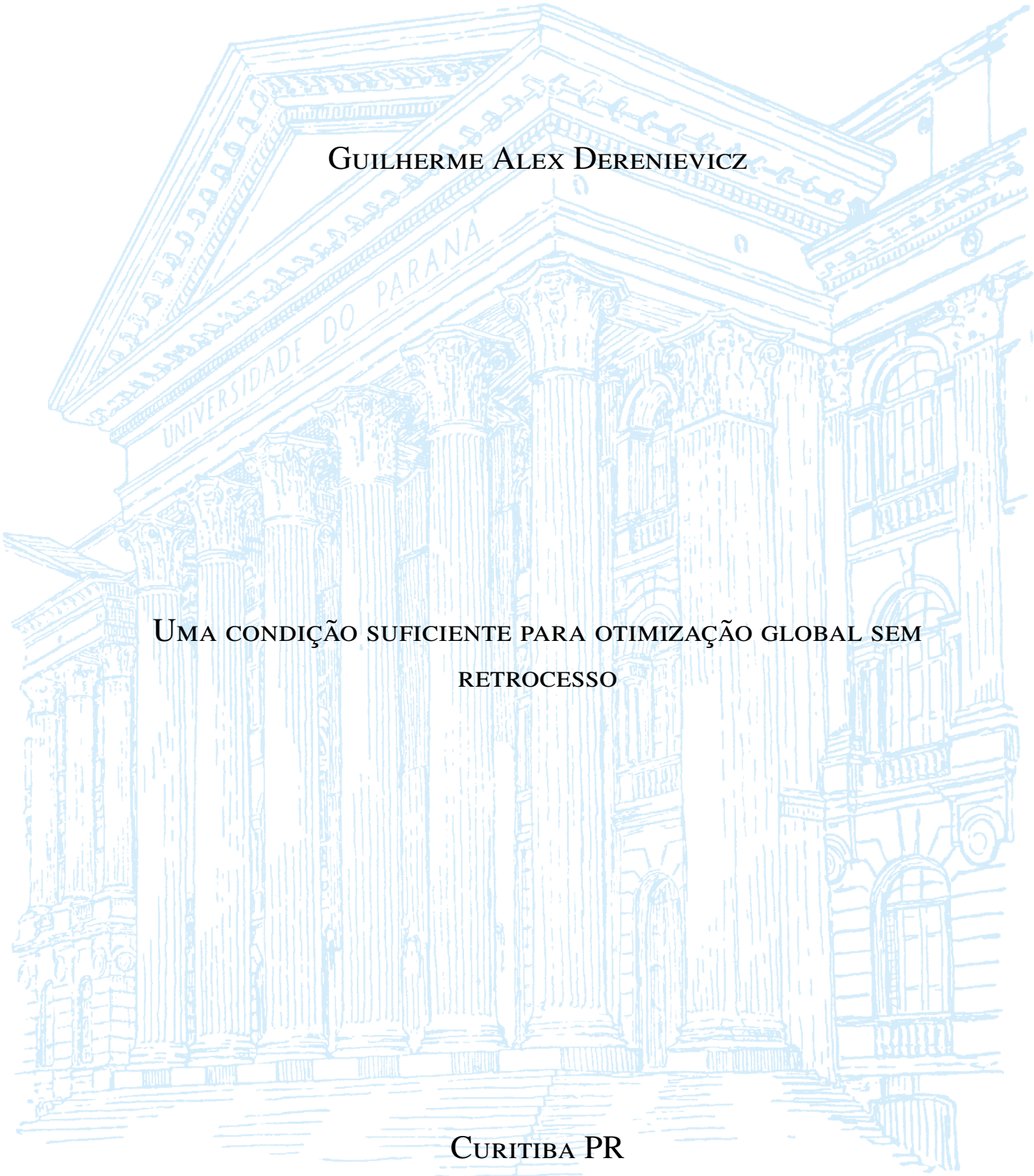
Universidade Federal do Paraná

Guilherme Alex Derenievicz



Uma condição suficiente para otimização global sem retrocesso

Curitiba PR
2018

Guilherme Alex Derenievicz

Uma condição suficiente para otimização global sem retrocesso

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: Fabiano Silva.

Curitiba PR
2018

FICHA CATALOGRÁFICA ELABORADA PELO SISTEMA DE BIBLIOTECAS/UFPR
BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

D431c Derenievicz, Guilherme Alex
Uma condição suficiente para otimização global sem retrocesso / Guilherme Alex Derenievicz. – Curitiba, 2018.
193 p. : il. color. ; 30 cm.

Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2018.

Orientador: Fabiano Silva.

1. Otimização global. 2. Consistência relacional. 3. Análise intervalar. 4. Decomposição epífita. 5. Hipergrafo. I. Universidade Federal do Paraná. II. Silva, Fabiano. III. Título.

CDD: 004.6

Bibliotecária: Romilda Santos - CRB-9/1214

MINISTÉRIO DA EDUCAÇÃO
SETOR CIÊNCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
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 **GUILHERME ALEX DERENIEVICZ** intitulada: **Uma condição suficiente para otimização global sem retrocesso**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 04 de Maio de 2018.

FABIANO SILVA
Presidente da Banca Examinadora

THIAGO ALVES DE QUEIROZ
Avaliador Externo

MARCOS ALEXANDRE CASTILHO
Avaliador Interno

RENATO JOSÉ DA SILVA CARMO
Avaliador Interno

GEOVANI NUNES GRAPIGLIA
Avaliador Externo

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

Aos meus pais Aleixo e Zuleide, meu irmão Gustavo e minha namorada Tatiana. À memória de Alexandre I. Direne, grande amigo e professor.

AGRADECIMENTOS

Agradeço primeiramente ao meu amigo e orientador Fabiano Silva, pela grande amizade, confiança e dedicação nestes sete anos em que me orientou. Sou grato pelos ensinamentos valiosos e pelas longas conversas que tornaram este trabalho realidade.

Agradeço imensamente aos meus pais, Aleixo e Zuleide, por todo o apoio, carinho e motivação; obrigado pelos valores que me transmitiram e pelo exemplo que sempre me deram. Agradeço também ao meu irmão Gustavo, pela leal amizade e pelos incontáveis momentos de reflexão, diversão e música. Agradeço do fundo do meu coração à minha namorada, amiga e companheira Tatiana, por seu amor, paciência e dedicação; obrigado por estar ao meu lado nos momentos mais difíceis e também nos mais felizes.

Agradeço a todos os amigos que me apoiaram e compreenderam minha ausência nos últimos meses, em especial aos que puderam acompanhar esta jornada mais de perto: Tiago, Benjamin, Leticia, Romano, Jorge, Katia, Jaqueline, Vinícius e Bianca. Agradeço ao carinho de toda a minha família, que é grande demais para ser listada aqui.

Meus agradecimentos a todos os professores e funcionários do Dinf, pela disposição e excelência em contribuir com a formação acadêmica. Em particular, agradeço aos professores Daniel Weingaertner, Luis Allan Kunzle e Luis Carlos E. de Bona pelas ótimas orientações em projetos que participei. Agradeço aos professores André L. Vignatti, Daniel Weingaertner, Geovani N. Grapiglia, Marcos A. Castilho, Murilo V. G. da Silva, Renato Carmo e Thiago A. de Queiroz pelas valiosas contribuições em meu trabalho.

Gostaria de agradecer aos meus colegas durante o período em que fui professor substituto na UFPR e a todos os alunos que me ensinaram muito mais do que eu mesmo pude ensinar.

A conclusão desta tese encerra um ciclo que teve inicio em 2008, quando comecei minha graduação na UFPR. Gostaria de agradecer, desta forma, a todos os meus colegas de turma. Em especial, ao Renan, Joice, José Ivan, Felipe Bolsi, Antônio, Larissa, Jaime, Yuri, Felipe Ickert e João Resende. Agradeço aos amigos que fiz no PET Computação e no C3SL e aos meus colegas de time na maratona de programação de 2013, Flávio Zavan, Rodrigo Gryzinski e Ricardo T. de Oliveira. Agradeço aos meus colegas do LIAMF, Carlos, Clariane, Danielle, Razer, Rodolfo, Willian, Bruno Ribas e Marcos Schreiner; e também aos colegas Edgar, Bete e Renato Melo.

Agradeço à professora Dinacir Rocha, que despertou minha paixão pelas ciências exatas e ao Fernando Cezanoski pelas várias viagens a Tijucas ouvindo Engenheiros do Hawaii.

Meus mais sinceros agradecimentos ao professor Alexandre I. Direne, pela amizade e ensinamentos que ficarão em mim para sempre. Com o prof. Direne aprendi que ser professor é muito mais que ensinar e orientar; é inspirar, motivar, cativar, discutir sobre tantos assuntos que uma sala de aula é pequena demais para conter. Descanse em paz, prof. Direne; seu legado é eterno!

Sobretudo agradeço a Deus, pela oportunidade me oferecida de vivenciar e aprender tanta coisa com amigos tão especiais.

One, remember to look up at the stars and not down at your feet.
Two, never give up work. Work gives you meaning and purpose and life is empty without it.
Three, if you are lucky enough to find love, remember it is there and don't throw it away.

Stephen Hawking (1942 - 2018)

RESUMO

Um problema de satisfação de restrições (CSP, do inglês *constraint satisfaction problem*) consiste em encontrar uma atribuição de valores a um conjunto de variáveis que satisfaça uma rede de restrições. Técnicas de consistência local desempenham um papel central na resolução de CSPs, excluindo valores que certamente não constituem uma solução do problema. Muitos esforços vêm sendo aplicados na identificação de classes de CSPs relacionando a estrutura da rede (representada por um hipergrafo) com o nível de consistência local que garante uma solução livre de retrocesso, isto é, uma busca que encontra uma solução em um número polinomial de passos em relação ao tamanho da instância. Nesta tese, problemas de otimização global são representados por hipergrafos com um vértice raiz que representa a função objetivo a ser minimizada. Uma forma de decomposição de hipergrafos, chamada decomposição Epífita, é apresentada. Através da decomposição Epífita do hipergrafo de restrições, caracteriza-se uma classe de problemas de otimização onde a consistência de arco relacional direcionada garante uma solução livre de retrocesso. Alcançar consistência relacional exige a adição de novas restrições na rede, alterando a sua estrutura; por essa razão, um método de ramificação e poda intervalar para alcançar uma forma relaxada dessa consistência é proposto, encontrando uma aproximação do mínimo global de problemas de otimização. Um otimizador de código-fonte aberto que implementa esse método, chamado `OGRe`, é apresentado. A fim de generalizar o conceito de decomposição Epífita a todos os problemas de otimização, um parâmetro de largura de hipergrafos chamado largura epífita é introduzido. Como principal contribuição desta tese, mostra-se que problemas de otimização representados por hipergrafos com largura epífita k possuem decomposições k-Epífitas e são resolvidos sem retrocesso se alcançada k-consistência relacional direcionada forte.

Palavras-chave: otimização global, consistência relacional, análise intervalar, decomposição epífita, hipergrafo.

ABSTRACT

A constraint satisfaction problem (CSP) consists of finding an assignment of values to a set of variables that satisfy a constraint network. Local consistency techniques play a central role in solving CSPs, pruning values that surely do not constitute a solution of the problem. Many efforts have been applied to identify classes of CSPs by linking the constraint network structure (represented by a hypergraph) to the level of local consistency that guarantees a backtrack-free solution, i.e., a search that finds a solution in a polynomial number of steps with relation to the size of the instance. In this thesis, global optimization problems are represented by hypergraphs with a root vertex that represents the objective function to be minimized. A form of hypergraph decomposition is introduced, called Epiphytic decomposition. By the Epiphytic decomposition of constraint hypergraphs a class of optimization problems is characterized, for which directional relational arc-consistency ensures a backtrack-free solution. Achieving relational consistency requires the addition of new constraints on the network, changing its structure; for this reason, an interval branch and bound method to enforce a relaxed form of this consistency is proposed, thus finding an approximation for the global minimum of optimization problems. An open-source optimizer that implements this method, namely `OGRe`, is introduced. In order to generalize the Epiphytic decomposition concept to cover all optimization problems, a hypergraph width parameter is introduced, called epiphytic width. As the main contribution of this thesis, it is shown that optimization problems represented by hypergraphs with epiphytic width k have k-Epiphytic decompositions and are solved in a backtrack-free manner if achieved strong directional relational k-consistency.

Keywords: global optimization, relational consistency, interval analysis, epiphytic decomposition, hypergraph.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE DEFINIÇÕES

LISTA DE TEOREMAS

LISTA DE EXEMPLOS

SUMÁRIO

1 INTRODUÇÃO

Problemas de otimização aparecem em diversas áreas do conhecimento. Decidir um conjunto de ações que realizam uma determinada tarefa ou resolvem um problema da melhor forma possível é um objetivo comum da natureza humana e da sociedade como um todo. Muitos desses problemas podem ser representados em linguagem matemática, permitindo que ferramentas lógicas sejam utilizadas na busca por suas soluções.

O processo de representar problemas através de conceitos matemáticos, denominado modelagem, consiste em estabelecer variáveis e regras que relacionam essas variáveis, restringindo o conjunto de valores que cada uma delas pode assumir. Essa representação é, em geral, imperfeita, pois não contempla todas as características do sistema físico real. Dessa forma, escolher o conjunto de regras, ou restrições, que melhor representa o problema dentro de uma limitação imposta pelo maquinário disponível é, por si só, um problema difícil.

Uma forma de representar problemas de otimização é definir uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$, chamada função de custo ou função objetivo, que deve ser minimizada sob um conjunto de (in)equações da forma $g(\mathbf{x}) \leq 0$. No caso geral, otimização global é um problema $\mathcal{NP}$-difícil e, portanto, não se conhece um algoritmo de tempo polinomial no tamanho da instância para resolvê-lo.

Curiosamente, humanos "resolvem" diariamente problemas que são computacionalmente difíceis. Uma possível explicação para esse fato é que o raciocínio humano aproxima esses problemas por outros que são mais fáceis de resolver; ou ainda, divide o problema original em subproblemas mais fáceis cujas soluções são combinadas para aproximar a solução do problema original. De fato, muitos problemas computacionais difíceis possuem um subconjunto de instâncias que são fáceis de resolver. No caso da otimização global, por exemplo, quando a função objetivo e as equações do sistema são lineares, a instância pode ser resolvida em tempo polinomial (Khachiyan, 1980).

Nesse sentido, caracterizar subclasses de problemas que são fáceis de resolver, ou definir propriedades que garantam a solução de um problema difícil, como por exemplo, determinar uma relação entre a estrutura de cada instância e a quantidade de processamento suficiente para que se consiga resolvê-la de maneira eficiente, são questões de fundamental importância teórica e prática.

Nesta tese, essas relações são abordadas especificamente para o caso da otimização global, inspiradas em resultados semelhantes obtidos anteriormente na área de satisfação de restrições.

1.1 Contextualização

Um modelo matemático de importância indiscutível na representação de problemas de otimização é a programação linear, na qual define-se um conjunto de restrições sobre variáveis reais na forma de inequações de funções lineares que devem ser satisfeitas simultaneamente, além de uma função de custo linear que deve ser minimizada (Papadimitriou e Steiglitz, 1982; Bertsimas e Tsitsiklis, 1997). O problema de encontrar valores reais que satisfazem as condições desse modelo (que seriam, então, uma solução aproximada do problema real modelado) foi vastamente estudado nas últimas décadas, gerando milhares de artigos, livros, algoritmos e otimizadores, capazes de resolver instâncias de programação linear mesmo com um grande número de variáveis e restrições.

Formalmente, um problema de programação linear (LP, do inglês *linear programming*) é composto por uma função de custo $f : \mathbb{R}^n \mapsto \mathbb{R}$ e um conjunto de restrições S sobre variáveis reais $x_1, \ldots, x_n$.

$$\min \quad f(x_1, \ldots, x_n) \tag{1.1}$$

sujeito a

$$S = \begin{cases} g_i(x_1, \ldots, x_n) \leq 0 & \text{para } 1 \leq i \leq m \\ x_i \in D_i & \text{para } 1 \leq i \leq n \end{cases} \tag{1.2}$$

onde f e g_i são funções lineares, $D_i \subseteq \mathbb{R}$ é o conjunto domínio da variável x_i e $m, n \in \mathbb{N}^*$.

Definição 1.1 (Função linear). Uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$ é dita linear se, e somente se,

$$f(x_1 + y_1, \ldots, x_n + y_n) = f(x_1, \ldots, x_n) + f(y_1, \ldots, y_n) \tag{1.3}$$

$$f(ax_1, \ldots, ax_n) = af(x_1, \ldots, x_n) \tag{1.4}$$

Resolver uma instâncias de LP é encontrar uma atribuição de valores às variáveis $x_1, \ldots, x_n$ que satisfaça todas as restrições do sistema S e minimize a função objetivo f, isto é, tal que

$$f(x_1, \ldots, x_n) \leq f(x_1', \ldots, x_n') \tag{1.5}$$

para toda atribuição $(x_1', \ldots, x_n')$ que satisfaz S.

Em geral, problemas de otimização são classificados em contínuos ou discretos. Problemas contínuos são aqueles nos quais as variáveis assumem valores reais de um conjunto domínio infinito, sob restrições definidas por equações ou inequações. Problemas discretos, por

outro lado, definem um conjunto domínio discreto, finito ou infinito. Exemplos de problemas discretos são os combinatórios ou de programação linear inteira, no qual as variáveis devem assumir apenas valores dentro de um subconjunto de $\mathbb{Z}$. Programação linear inteira é um problema $\mathcal{NP}$-completo (Papadimitriou e Steiglitz, 1982).

Em particular, problemas de programação linear são contínuos. No entanto, como seu sistema de restrições define um poliedro convexo em $\mathbb{R}^n$, também apresentam características discretas. A saber, a solução de uma instância LP pode ser obtida através de uma busca no conjunto finito de pontos extremos do poliedro definido por suas restrições (Bertsimas e Tsitsiklis, 1997). Na prática, um dos métodos mais importantes para resolver LP, chamado Simplex, deriva desta observação (Dantzig, 1963).

Embora haja um vasto maquinário especializado em resolver LP, existem problemas que não são bem representados por funções lineares, de forma que a solução do modelo matemático não reflete uma solução real do problema. A necessidade de representação de características não lineares exigiu a definição de um modelo de representação mais poderoso que a programação linear, ao mesmo tempo que demandou, nas últimas quatro décadas, uma extensa pesquisa com o objetivo de resolver este novo modelo, adaptando técnicas já conhecidas do contexto linear e propondo novos algoritmos de resolução. Assim, no problema da programação não linear (NLP, do inglês *nonlinear programming*), a função de custo e aquelas que compõem as restrições do sistema podem não ser lineares (Bertsekas, 1999).

Exemplo 1.1. Instância *sample* do *benchmark* COCONUT (Shcherbina et al., 2003b):

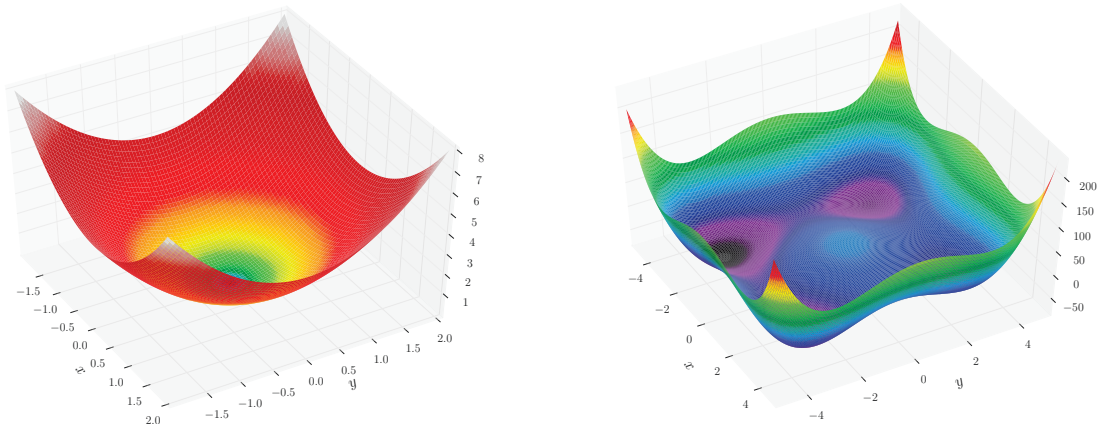
$$\min \quad x_1 + x_2 + x_3 + x_4 \tag{1.6}$$

sujeito a

$$\begin{aligned} & 4/x_1 + 2{,}25/x_2 + 1/x_3 + 0{,}25/x_4 \leq 0{,}0401 \\ & 0{,}16/x_1 + 0{,}36/x_2 + 0{,}64/x_3 + 0{,}64/x_4 \leq 0{,}010085 \\ & x_1 \in [100, 400000] \\ & x_2 \in [100, 300000] \\ & x_3 \in [100, 200000] \\ & x_4 \in [100, 100000] \end{aligned} \tag{1.7}$$

△

O grande esforço em desenvolver algoritmos para NLP resultou em uma gama de otimizadores que encontram a solução ótima de instâncias, em geral, classificadas por características específicas. A principal dessas classes é definida por instâncias convexas ou côncavas (ver Figura 1.1), nas quais é possível obter a solução ótima através da análise do vetor gradiente da função objetivo.



(a) (b)

Figura 1.1: Exemplo de função convexa (a), na qual existe um único mínimo local, e não convexa (b), com 4 mínimos locais. Figuras por Ortiz (2012b) e Ortiz (2012c).

Para o Exemplo 1.1, o otimizador BARON (Sahinidis, 1996), que utiliza-se de informações do vetor gradiente, encontra a seguinte solução:

$$x_1 = 193{,}4479462950$$
$$x_2 = 179{,}4363559960$$
$$x_3 = 185{,}1184412970$$
$$x_4 = 168{,}6339551800$$

resultando no mínimo global 726,6366987670 para a função de custo. É importante notar que as duas primeiras restrições do sistema não são satisfeitas de forma exata, mas com uma precisão de 5 casas decimais, referentes a erros de representação numérica e arredondamentos.

Métodos que utilizam-se de informações de convexidade e diferenciabilidade da função objetivo (através de vetores gradientes, por exemplo), provenientes da comunidade de programação matemática e pesquisa operacional, são vastamente aplicados a problemas de otimização global. Exemplos desses métodos são o Lagrangeano Aumentado (Hestenes, 1969) e o Pontos Interiores (Potra e Wright, 1997), nos quais, a partir de uma atribuição de valores iniciais, obtém-se novas atribuições que convergem para uma solução da instância.

Embora os métodos da comunidade de programação matemática constituam o principal ferramental disponível para abordar problemas de otimização global, em instâncias NLP não convexas e com múltiplos mínimos locais tais métodos não garantem a otimalidade global da solução encontrada, pois podem convergir para um ótimo **local** da função objetivo.

Definição 1.2 (Ótimo local). Dada uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$, diz-se que $f(x_1, \ldots, x_n)$ é um ótimo local de f se $f(x_1, \ldots, x_n) \leq f(x_1', \ldots, x_n')$, para todo $x_i' \in \mathbb{R}$ tal que $|x_i - x_i'| < \delta$, para algum $\delta > 0$ e para todo $1 \leq i \leq n$.

Por outro lado, métodos exatos (ou completos), como ramificação e poda, busca com retrocesso, métodos intervalares (Hansen e Walster, 2004) ou, mais especificamente no cenário discreto ou convexo, métodos combinatórios, busca exaustiva e métodos de plano de corte (Papadimitriou e Steiglitz, 1982), conseguem garantir a otimalidade global da solução encontrada, pois percorrem todo o espaço de busca.

Com o avanço de técnicas de inferência aplicadas à análise intervalar, métodos de ramificação e poda vêm sendo aplicados na resolução de problemas de otimização global de forma exata (Hansen, 1980; Ratschek e Rokne, 1988; Kearfott, 1992; Van Hentenryck, 1997; Hansen e Walster, 2004; Lebbah et al., 2007; Kampen, 2010; Bhurjee e Panda, 2012), embora, na prática, sua eficiência ainda seja inferior aos métodos da comunidade de programação matemática (Neumaier et al., 2005; Araya et al., 2014).

1.2 Motivação

Na área de programação por restrições, campo de pesquisa que emergiu na década de 1970 combinando técnicas de inteligência artificial, programação lógica e pesquisa operacional, um problema de satisfação de restrições (CSP, do inglês *constraint satisfaction problem*) consiste em encontrar uma atribuição de valores a um conjunto de variáveis que satisfaça um conjunto de restrições. No caso mais geral, não se conhece um algoritmo de tempo polinomial no tamanho da instância para resolver CSP. Formalmente, CSP é um problema $\mathcal{NP}$-difícil (Dechter, 2003).

Nas últimas décadas, diversas técnicas foram propostas a fim de melhorar o tempo de caso médio na resolução de CSPs. Em geral, essas técnicas constituem duas classes principais de algoritmos: busca e inferência. No primeiro caso, destacam-se os algoritmos de retrocesso, que consistem em valorar variáveis em uma determinada ordem até que se encontre uma solução da instância ou conclua-se que a valoração, mesmo que parcial, não pode caracterizar uma solução; neste caso, a valoração da última variável é desfeita e outro valor lhe é atribuído. No segundo caso, inferência caracteriza-se por reduzir a instância ao adicionar informações que estão implícitas em sua própria representação. Por exemplo, a partir da afirmação "todo homem é mortal" e da sentença "Sócrates é homem" é possível inferir que "Sócrates é mortal". Ou ainda, a partir da inequação $x_1 < x_2$ e dos domínios $x_1 \in \{1, 2, 3\}$ e $x_2 \in \{1, 2, 3\}$ pode-se inferir que $x_1 = 3$ é uma valoração parcial inválida (também dita inconsistente), bem como a valoração parcial $x_2 = 1$. Este tipo de inferência é chamado de consistência de arco, proposta por Waltz (1975) para abordar o problema de interpretar figuras bidimensionais de poliedros.

Em geral, para encontrar a solução de uma instância CSP, busca e inferência costumam ser combinadas. Por outro lado, ao propor o primeiro algoritmo de consistência, Waltz notou que algumas instâncias de problemas eram eficientemente resolvidas apenas com inferência, sem a necessidade de retrocesso. Mackworth e Freuder (1985) analisaram a complexidade da consistência de arco e concluíram que instâncias acíclicas são resolvidas alcançando-se esse nível de consistência. Freuder (1982) já havia identificado uma condição suficiente para que instâncias

de CSPs fossem resolvidas sem retrocesso; resumidamente, Freuder estabelecera uma relação entre a estrutura da instância, usualmente representada por um grafo, e o nível de consistência necessário para resolvê-la.

O trabalho de Freuder inspirou outros pesquisadores a encontrar relações entre estrutura e nível de consistência. Jégou (1993) estendeu os resultados anteriores a redes de restrições não binárias. Van Beek e Dechter estabeleceram o nível de consistência relacional que garante uma solução livre de retrocesso em instâncias com tamanho de domínios e tamanho de restrições restritos (van Beek e Dechter, 1997), bem como quanto à convexidade das restrições (van Beek e Dechter, 1995). Sam-Haroud e Faltings (1996) também estabeleceram uma relação entre consistência relacional e convexidade de restrições em problemas contínuos. Embora CSP seja um problema $\mathcal{NP}$-difícil, tais investigações puderam identificar classes de instâncias solúveis em tempo polinomial, como problemas acíclicos, satisfazibilidade proposicional de cláusulas de tamanho 2 e fórmulas de Horn (Dechter, 2003).

Em geral, a relação entre estrutura e consistência estabelece uma condição suficiente para que uma instância CSP seja resolvida sem a necessidade de retrocesso. Além disso, estes conceitos são naturalmente estendidos a problemas de otimização (como NLP, por exemplo). No entanto, até onde esta pesquisa pôde alcançar, não se conhece um trabalho que identifique uma condição suficiente para otimização global sem retrocesso de maneira semelhante aos trabalhos supracitados.

1.3 Objetivos da tese

O objetivo desta tese é identificar uma condição suficiente para otimização global sem retrocesso, sob a qual problemas de otimização são resolvidos em tempo polinomial. Naturalmente, alcançar essa condição em uma instância genérica é certamente um problema $\mathcal{NP}$-difícil.

Adicionalmente, deseja-se caracterizar classes de problemas para o quais é possível alcançar essa condição eficientemente, bem como introduzir um parâmetro de complexidade que identifique o nível de pré-processamento que alcança a condição de solução polinomial em cada instância do problema.

1.4 Contribuições da tese

Esta seção está dividida em dois tópicos: contribuições principais, listadas em ordem de apresentação, e contribuições secundárias. Uma descrição detalhada das contribuições será apresentada na conclusão desta tese (Seção 6.1). Esta pesquisa gerou a seguinte publicação:

- Derenievicz, G. A., Silva, F. (2018). Epiphytic Trees: Relational Consistency Applied to Global Optimization Problems. Em *Proceedings of the 15th International Conference*

on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2018. *Lecture Notes in Computer Science*. 10848.

Contribuições principais:

- Na Seção 4.2, classifica-se problemas de otimização em três classes primárias, conforme o nível de consistência que garante solução livre de retrocesso. Prova-se que instâncias acíclicas são resolvidas aplicando-se consistência de arco generalizada, que é a extensão de um resultado previamente estabelecido no contexto de satisfação de restrições. Além disso, mostra-se que instâncias ordenadas de forma que cada restrição contenha pelo menos uma variável que não aparece nas restrições antecessoras são resolvidas com uma combinação de consistência de arco generalizada direcionada e consistência de arco relacional direcionada;

- Na Seção 4.3, uma decomposição de hipergrafos chamada decomposição Epífita é definida. Ao representar problemas de otimização por hipergrafos, essa decomposição garante a existência de ordenação de restrições conforme descrito no item anterior;

- Na Seção 4.3.2, propõe-se um algoritmo de tempo linear para verificar se um hipergrafo arbitrário possui uma decomposição Epífita. Este algoritmo também é uma extensão de resultados semelhantes no campo da satisfação de restrições;

- Na Seção 4.4, propõe-se um método de ramificação e poda que aproxima a consistência de arco relacional direcionada, caracterizando um método de otimização global aproximado. Este método é uma variante dos algoritmos intervalares usualmente aplicados para resolver problemas de otimização global de forma exata, propondo novas heurísticas para a escolha de variáveis a serem ramificadas;

- Na Seção 4.5, estende-se a condição suficiente para otimização sem retrocesso a todas as instâncias de problemas de otimização global, através do conceito de decomposição k-Epífita;

- Na Seção 4.6, compara-se a decomposição k-Epífita com outros parâmetros de caracterização de classes de hipergrafos e o método de otimização proposto com outras abordagens da literatura;

- No Capítulo 5, apresenta-se a implementação preliminar de um otimizador de código-fonte aberto denominado `OGRe` (Otimização Global Relaxada), fundamentado nos resultados teóricos obtidos nesta pesquisa. O objetivo primário deste otimizador é avaliar experimentalmente a classe de instâncias que possuem uma decomposição Epífita, através do algoritmo que aproxima a consistência de arco relacional direcionada. Este otimizador está disponível em Derenievicz (2018), sob a licença GNU GPL.

Contribuições secundárias:

- Os Capítulos 2 e 3 apresentam uma revisão sobre problemas de satisfação de restrições e análise intervalar. Em particular, no Capítulo 3, várias técnicas de consistência consagradas na literatura são apresentadas de maneira didática e linear. Esta organização pode diferir da apresentada em diversos livros de inteligência artificial, porém, objetiva uma melhor assimilação do conteúdo conforme a experiência do autor durante a pesquisa. Em geral, há notável escassez de literatura em português revisando técnicas de consistência, sendo esta tese mais uma possível fonte de consulta. Em particular, um pequeno capítulo do livro Norvig e Russell (2017) é dedicado a problemas de satisfação de restrições, em cuja tradução baseiam-se as principais traduções de termos utilizadas nesta tese.

1.5 Organização da tese

Esta tese está organizada da seguinte maneira:

- No Capítulo 2, são apresentadas definições e conceitos básicos de hipergrafos, problemas de satisfação de restrições e análise intervalar;
- No Capítulo 3, elenca-se técnicas de processamento de restrições, destacando-se as técnicas de consistência para problemas finitos e numéricos. Também apresenta-se a aplicação de métodos intervalares na resolução de problemas numéricos, inclusive otimização global, e faz-se uma revisão acerca de soluções sem retrocesso;
- No Capítulo 4, define-se decomposição Epífita de hipergrafos e prova-se uma condição suficiente para otimização global sem retrocesso. Um algoritmo de consistência relacional aproximada é proposto. Por fim, estende-se os resultados a todas as instâncias de problemas de otimização;
- No Capítulo 5, apresenta-se o otimizador `OGRe` como implementação do método proposto, bem como resultados preliminares obtidos;
- No Capítulo 6, conclui-se a tese revisando as contribuições e limitações da pesquisa, indicando algumas questões em aberto e possíveis trabalhos futuros.

2 DEFINIÇÕES

Neste capítulo, introduz-se definições e conceitos fundamentais do campo de satisfação de restrições. Em particular, como forma de representação e processamento de restrições numéricas, apresenta-se a análise intervalar e as principais operações aritméticas envolvendo intervalos. As definições e notação aqui utilizadas seguem, de forma geral, o modelo usualmente utilizado na literatura, sobretudo nos livros de Berge (1985), Dechter (2003), Rossi et al. (2006), Hansen e Walster (2004) e Moore et al. (2009).

2.1 Conjuntos, grafos e hipergrafos

Nesta seção, são apresentadas algumas definições e notação sobre conjuntos, grafos e hipergrafos; conceitos fundamentais para o restante desta tese.

Definição 2.1 (**Conjunto**)**.** Um conjunto A é uma coleção de objetos distintos. A cardinalidade $|A|$ de um conjunto é o seu número de elementos. Se $|A| \leq n$, para algum $n \in \mathbb{N}$, então A é dito um conjunto finito; do contrário, A é dito infinito. O conjunto que não possui elementos é chamado de conjunto vazio e é denotado por $\emptyset$, onde $|\emptyset| = 0$. Se exite uma relação de ordem total entre os elementos de A, então A é dito um conjunto ordenado. Denota-se que x é um elemento de A por $x \in A$ ou $A \ni x$. Analogamente, $x \notin A$ e $A \not\ni x$ denotam que x não pertence a A.

Dados dois conjuntos A e B quaisquer, define-se:

- $A \subseteq B$ (A é um subconjunto de B) se, e somente se, $\forall x \in A : x \in B$;
- $A \supseteq B$ (A é um superconjunto de B) se, e somente se, $B \subseteq A$;
- $A = B$ se, e somente se, $A \subseteq B$ e $A \supseteq B$;
- $A \subset B$ (A é um subconjunto próprio de B) se, e somente se, $A \subseteq B$ e $A \neq B$;
- $A \supset B$ (A é um superconjunto próprio de B) se, e somente se, $B \subset A$;
- $A \cap B = \{x \mid x \in A \text{ e } x \in B\}$;
- $A \cup B = \{x \mid x \in A \text{ ou } x \in B\}$;
- $A \setminus B = \{x \mid x \in A \text{ e } x \notin B\}$;

- A e B são ditos disjuntos se, e somente se, $A \cap B = \emptyset$.

Definição 2.2 (**Grafo**)**.** Um grafo é um par $G = (V, E)$, onde V é um conjunto finito de vértices e $E \subseteq \{S \subseteq V \mid |S| = 2\}$ é um conjunto finito de arestas. O grau de um vértice v é o número de arestas de G que contêm v, isto é, $|\{A \in E \mid v \in A\}|$.

Um grafo é usualmente desenhado como um conjunto de pontos que representam seus vértices, e estes são ligados sempre que existir uma aresta correspondente no grafo.

Definição 2.3 (**Subgrafo**)**.** O grafo $G' = (V', E')$ é um subgrafo de $G = (V, E)$ se, e somente se, $V' \subseteq V$ e $E' \subseteq E$.

Definição 2.4 (**Caminho, ciclo e árvore**)**.** Dado um grafo $G = (V, E)$, um caminho entre os vértices v_1 e v_n em G é uma sequência $(v_1, \ldots, v_n)$ de vértices distintos (exceto possivelmente por v_1 e v_n) tal que $\{v_1, \ldots, v_n\} \subseteq V$ e $\{v_i, v_{i+1}\} \in E$, para todo $1 \leq i < n$. Um ciclo é um caminho $(v_1, \ldots, v_n)$ tal que $v_1 = v_n$. Um grafo G é dito cíclico se possui pelo menos um ciclo; do contrário, G é dito acíclico. Um grafo G é conexo se existe pelo menos um caminho entre qualquer par de vértices em G. Uma árvore é um grafo conexo e acíclico.

Definição 2.5 (**Hipegrafo**)**.** Um hipergrafo é um par $\mathcal{H} = (V, E)$, onde V é um conjunto finito de vértices e $E \subseteq \{S \subseteq V \mid S \neq \emptyset\}$ é um conjunto finito de arestas. Se toda aresta $A \in E$ tem cardinalidade k, então $\mathcal{H}$ é dito um hipergrafo k-uniforme.

Hipergrafo é uma extensão natural do conceito de grafo. De fato, um hipergrafo 2-uniforme é um grafo. O grau de um vértice v no hipergrafo $\mathcal{H}$, de maneira análoga à definição de grau de vértices em grafos, é o número de arestas de $\mathcal{H}$ que contêm v. Um hipergrafo também é desenhado utilizando-se um conjunto de pontos para representar seus vértices, enquanto suas arestas podem ser representadas por linhas ou diagramas de Venn, como mostra a Figura 2.1. Nesta tese, arestas são sempre representadas por linhas.

Definição 2.6 (**Hipergrafo parcial**)**.** O hipergrafo $H' = (V', E')$ é um hipergrafo parcial de $\mathcal{H} = (V, E)$ se, e somente se, $E' \subseteq E$ e $V' = \bigcup_{A \in E'} A$. Dada uma aresta $A \in E$, denota-se por $\mathcal{H} - A$ o hipergrafo parcial de $\mathcal{H}$ com arestas $E \setminus \{A\}$.

Definição 2.7 (**Grafo primal**)**.** Dado um hipergrafo $\mathcal{H} = (V, E)$, define-se o grafo primal $\mathcal{H}^{primal} = (V^{primal}, E^{primal})$ tal que $V^{primal} = V$ e $E^{primal} = \{\{v_i, v_j\} \mid v_i, v_j \in A, v_i \neq v_j \text{ e } A \in E\}$.

Definição 2.8 (**Grafo dual**)**.** Dado um hipergrafo $\mathcal{H} = (V, E)$, define-se o grafo dual $\mathcal{H}^{dual} = (V^{dual}, E^{dual})$ tal que $V^{dual} = E$ e $E^{dual} = \{\{A_i, A_j\} \mid A_i, A_j \in E, A_i \neq A_j \text{ e } A_i \cap A_j \neq \emptyset\}$.

Informalmente, no grafo primal de um hipergrafo todo par de vértices que se relacionam é conectado por uma aresta. Por outro lado, no grafo dual cada aresta é interpretada como um

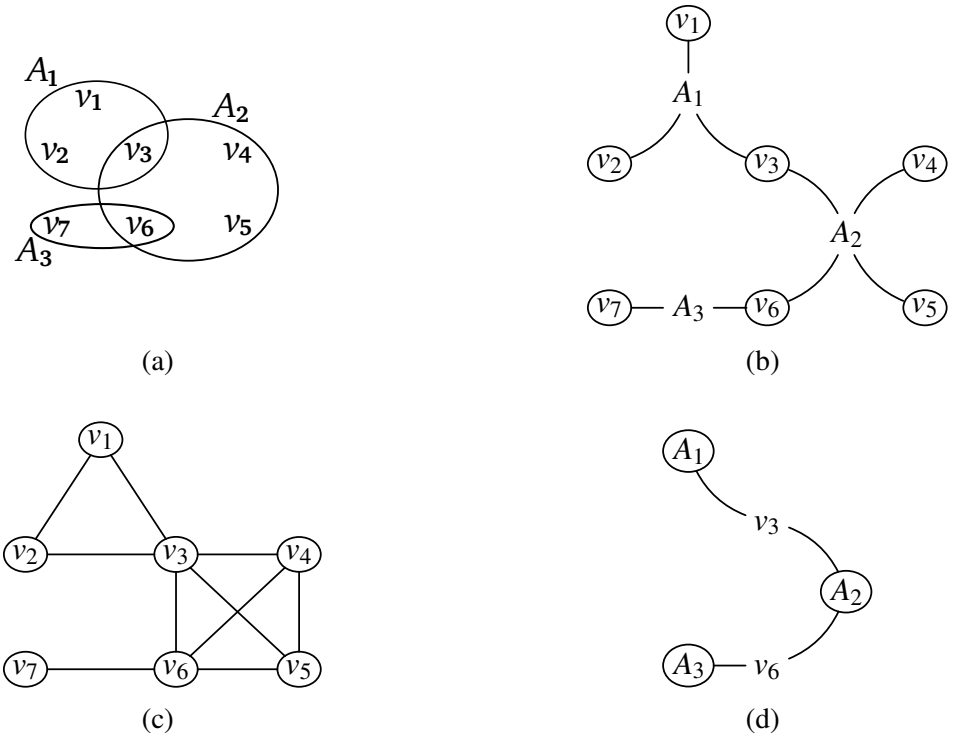


Figura 2.1: Hipergrafo $\mathcal{H} = (\{v_1, \ldots, v_7\}, \{\{v_1, v_2, v_3\}, \{v_3, v_4, v_5, v_6\}, \{v_6, v_7\}\})$ representado por (a) um diagrama de Venn e (b) um conjunto de pontos conectados; (c) o seu grafo primal e (d) o seu grafo dual.

vértice e estes são conectados sempre que as respectivas arestas compartilharem vértices no hipergrafo original. A Figura 2.1 apresenta o grafo primal (c) e o grafo dual (d) do hipergrafo (a).

O conceito de ciclo não é naturalmente extensível a hipergrafos, de forma que é possível definir várias noções de ciclicidade. Entre as definições mais importantes, cita-se o conceito de Berge-ciclo, apresentado em Berge (1985), e os conceitos de γ-ciclo, β-ciclo e α-ciclo (Brouwer e Kolen, 1980; Beeri et al., 1981, 1983; Fagin, 1983), propostos originalmente no cenário de banco de dados relacionais, mas também aplicados na teoria de hipergrafos e de satisfação de restrições. Estas definições podem ser ordenadas de acordo com a sua generalização, da seguinte forma:

$$\text{Berge-acíclico} \preceq \gamma\text{-acíclico} \preceq \beta\text{-acíclico} \preceq \alpha\text{-acíclico}$$

Isto é, todo hipergrafo Berge-acíclico também é γ-acíclico, e assim por diante. Nesta tese, são utilizados os conceitos de Berge-acíclico e α-acíclico.

Definição 2.9 (Berge-caminho e Berge-ciclo). Dado um hipergrafo $\mathcal{H} = (V, E)$, um Berge-caminho entre as arestas A_1 e A_n em $\mathcal{H}$ é uma sequência alternada $(A_1, v_1, A_2, v_2, \ldots, A_{n-1}, v_{n-1}, A_n)$ de arestas e vértices distintos (exceto possivelmente por A_1 e A_n) tal que $n \geq 3$ e $v_i \in (A_i \cap A_{i+1})$, para todo $1 \leq i < n$. Um Berge-ciclo é um Berge-caminho $(A_1, v_1, A_2, v_2, \ldots, A_{n-1}, v_{n-1}, A_n)$ tal que $A_1 = A_n$. Um hipergrafo $\mathcal{H}$ é dito Berge-cíclico se possui pelo menos um Berge-ciclo; do contrário, $\mathcal{H}$ é dito Berge-acíclico.

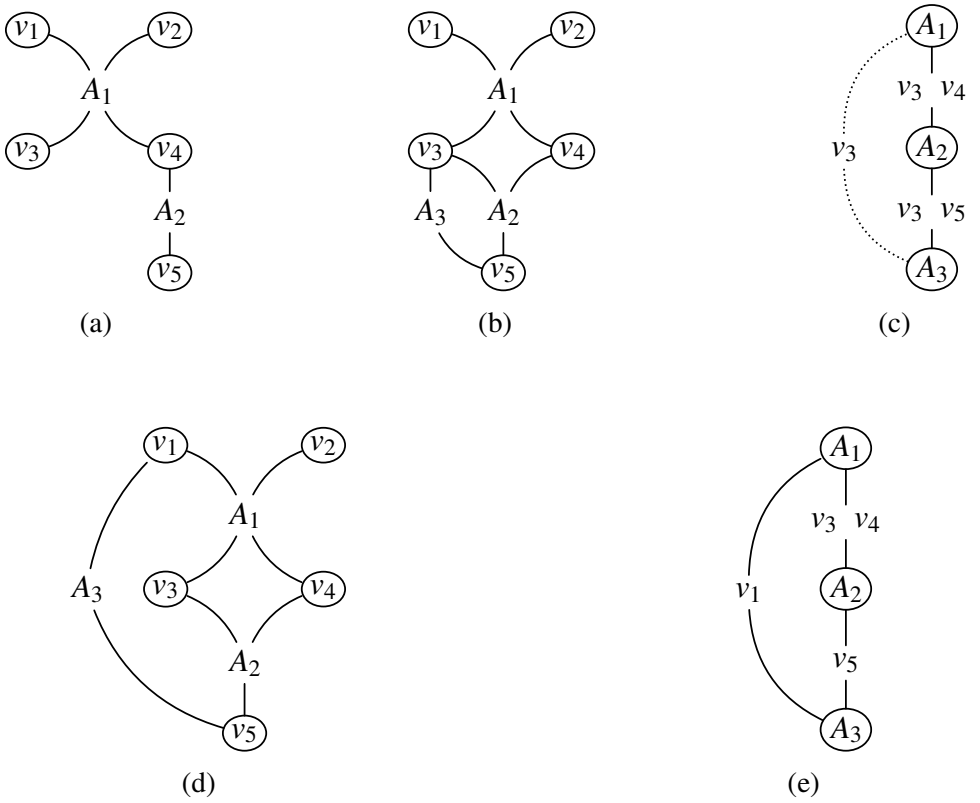


Figura 2.2: Exemplo de hipergrafo (a) Berge-acíclico, e (b) e (d) Berge-cíclicos; por exemplo, $(A_1, v_3, A_2, v_4, A_1)$ é um Berge-ciclo em ambos os hipergrafos. Enquanto o hipergrafo (b) é α-acíclico, pois possui uma árvore de junção (c), desconsiderando a aresta pontilhada que é redundante, o hipergrafo (d) é α-cíclico, pois não possui uma árvore de junção, conforme pode ser visto pelo seu grafo dual (e).

Definição 2.10 (**Árvore de junção**)**.** Um hipergrafo $\mathcal{H} = (V, E)$, com grafo dual $\mathcal{H}^{dual} = (V^{dual}, E^{dual})$, possui uma árvore de junção se, e somente se, existe uma árvore $T = (V_T, E_T)$ tal que $V_T = V^{dual}$, $E_T \subseteq E^{dual}$ e para todo par de vértices distintos $v_i, v_j \in V_T$, tais que $v_i \cap v_j \neq \emptyset$, existe um caminho $(v_i = v_1, \ldots, v_n = v_j)$ em T tal que $v_i \cap v_j \subseteq v_k$, para todo $1 \leq k \leq n$. Um hipergrafo $\mathcal{H}$ é dito α-acíclico se possui uma árvore de junção; do contrário, $\mathcal{H}$ é dito α-cíclico.

Dado um hipergrafo $\mathcal{H}$, uma árvore de junção pode ser obtida removendo-se de $\mathcal{H}^{dual}$ as arestas **redundantes**. Aqui, uma aresta é considerada redundante se ela conecta dois vértices duais A_i e A_j e existe um outro caminho entre A_i e A_j no qual todo vértice desse caminho contém $A_i \cap A_j$ (lembrando que os vértices de $\mathcal{H}^{dual}$ são as arestas de $\mathcal{H}$). Se o grafo obtido ao remover-se as arestas redundantes é uma árvore, então esta é uma árvore de junção de $\mathcal{H}$.

Exemplo 2.1. A Figura 2.2 apresenta exemplos de hipergrafo Berge-acíclico, α-acíclico e α-cíclico, com os respectivos grafos duais. △

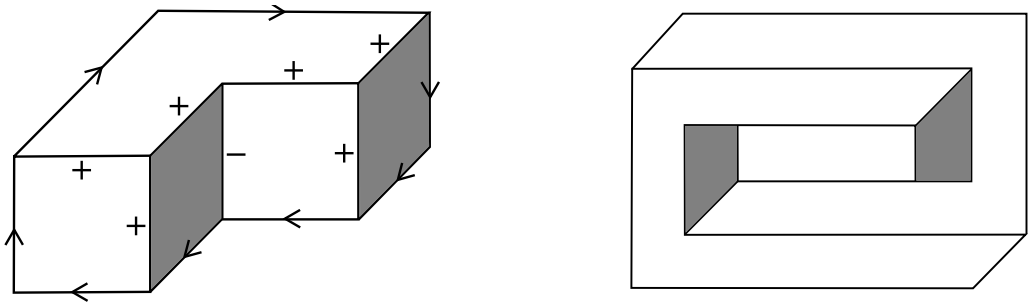
2.2 Problemas de Satisfação de Restrições

Restrições existem naturalmente em diversas atividades do dia a dia e áreas do conhecimento humano. A razão entre o perímetro de uma circunferência e seu diâmetro é π; a soma dos ângulos de um triângulo é sempre 180°; uma semana tem 7 dias; em 100 gramas de arroz tem-se 28 gramas de carboidratos e 2,7 gramas de proteínas. De modo geral, restrições são parcelas de conhecimento que reduzem o espaço de possibilidades no mundo real. Adicionalmente, resolver problemas com restrições também é uma atividade comum da natureza humana. Como obter uma alimentação balanceada? Como organizar uma viagem com os recursos disponíveis? Em qual data marcar um jantar para os amigos de forma que todos possam participar? De fato, enquanto alguns desses problemas são resolvidos diariamente sem muito custo, outros podem exigir um grande esforço computacional (muitos jantares são esquecidos simplesmente por não se conseguir estabelecer uma data adequada).

Diversas áreas das ciências modelam problemas utilizando restrições, denominados problemas de satisfação de restrições (CSPs). Informalmente, um CSP consiste em encontrar uma atribuição de valores a um conjunto de variáveis que satisfaça um conjunto de restrições. Técnicas de processamento de restrições são então aplicadas a esses problemas, classificadas em dois grupos principais: busca e inferência. O processo de modelar um problema do mundo real como CSP e aplicar sobre este técnicas de processamento de restrições é chamado de programação por restrições.

A programação por restrições emergiu de três diferentes áreas: matemática, pesquisa operacional e inteligência artificial. Em particular, problemas com restrições são estabelecidos na matemática há milhares de anos. Os babilônios já tinham um método para resolver equações quadráticas (uma equação é uma restrição sobre variáveis reais) em aproximadamente 2000 a.C.; o matemático indiano Brahmagrupta estudava a resolução de equações sobre variáveis inteiras no século VII; o astrônomo e matemático persa al-Khwarizmi, cujo nome originou o termo "algoritmo", apresentou um método para resolver equações lineares no século IX; e, mais recentemente, métodos de resolução de inequações e equações lineares foram estudados, respectivamente, por Fourier, em 1827, e por Gauss, em 1829.

Na área de inteligência artificial, a programação por restrições originou-se no sistema SKETCHPAD (Sutherland, 1963), que resolvia determinadas restrições geométricas em diagramas. Mas foi com o trabalho de Waltz (1972, 1975) que técnicas de inferência, também chamada de propagação de restrições, foram formalizadas. Waltz introduziu o primeiro algoritmo de propagação de restrições conhecido (mais tarde nomeado consistência de arco, por Mackworth (1977a)) aplicado ao problema de interpretar poliedros em figura bidimensionais. Simplificadamente, nesse problema cada aresta do poliedro pode ser rotulado como ocluso ($<$), convexo ($+$) ou côncavo ($-$); existem muitas combinações de rótulos, mas apenas algumas têm um significado tridimensional coerente, isto é, interpretam a imagem como um objeto real possível. A Figura 2.3 apresenta uma rotulagem possível de um objeto tridimensional e um objeto que não pode ser



(a) (b)

Figura 2.3: Uma rotulagem (a) encontrada pelo algoritmo de Waltz e uma figura (b) que não possui rotulagem válida, pois não representa um objeto tridimensional real.

rotulado, pois não é um objeto real. A ideia central do método de Waltz era fixar restrições entre os rótulos de arestas vizinhas e propagar essas restrições por todo o objeto, eliminando o conjunto de rótulos possíveis em cada aresta e reduzindo, assim, o espaço combinatório de rotulagens válidas. Ele evidenciou, empiricamente, que para alguns poliedros esse algoritmo é suficiente para resolver o problema sem a necessidade de uma busca combinatória. Montanari (1974) estendeu o algoritmo de Waltz e definiu, pela primeira vez, o termo "rede de restrições", identificando CSPs como uma classe geral e destacando a característica estrutural desses problemas, comumente associados a grafos e hipergrafos.

2.2.1 Alguns exemplos

A programação por restrições vem sendo aplicada com sucesso em diversas áreas da engenharia, ciência da computação e inteligência artificial. Entre as principais aplicações destacam-se:

- computação gráfica;
- sistemas de banco de dados;
- pesquisa operacional;
- projetos de circuitos;
- planejamento;
- alocação de recursos;
- sequenciamento de DNA;
- processamento de linguagem natural.

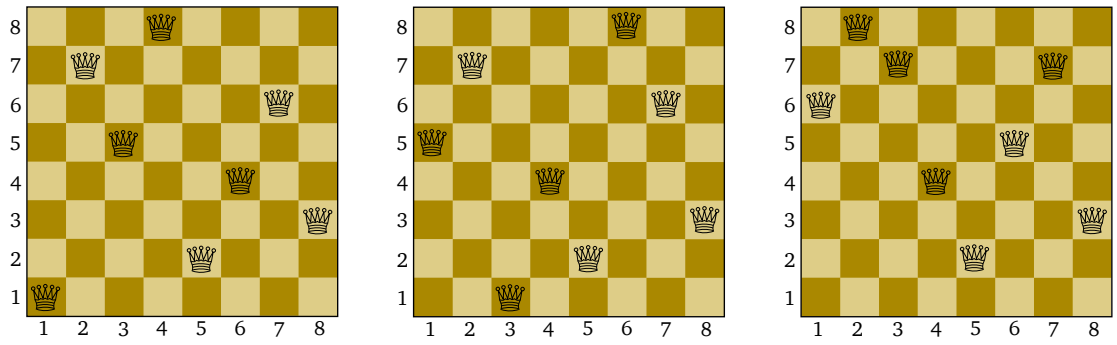


Figura 2.4: Três disposições de peças no problema das 8 rainhas; enquanto as duas primeiras configurações são soluções do problema, no terceiro tabuleiro todas as rainhas estão sendo atacadas por alguma outra.

Há ainda alguns problemas clássicos que podem ser citados por sua importância teórica, como o problema das 8 rainhas, palavras-cruzadas, satisfazibilidade proposicional e sistemas de equações.

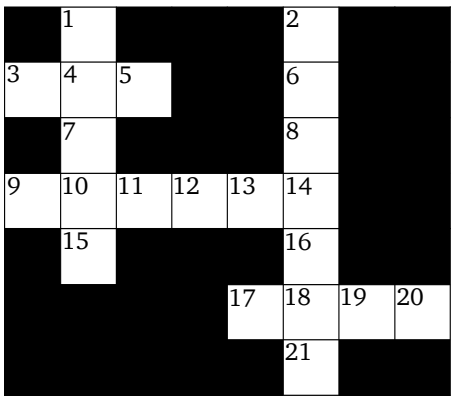
Problema das 8 rainhas

O problema das 8 rainhas consiste em dispor 8 rainhas em um tabuleiro de xadrez de forma que nenhuma peça possa ser capturada por outra. Uma rainha pode capturar outra peça se esta estiver em qualquer posição da mesma linha, coluna ou diagonal. Como cada rainha x_j deve ser colocada em uma coluna diferente, esse problema pode ser representado por 8 variáveis, uma para cada rainha em uma coluna do tabuleiro, e cada variável pode assumir um valor i do conjunto $\{1, 2, \ldots, 8\}$, representando que a rainha x_j está sendo colocada na linha i da coluna j. Além disso, existe um conjunto de restrições entre cada par de rainhas $\{x_j, x_k\}$, impedindo que estas sejam colocadas na mesma linha ou diagonal. A Figura 2.4 mostra três disposições distintas de rainhas no tabuleiro; enquanto as duas primeiras são soluções do problema, a terceira não satisfaz as restrições. Esse problema pode ser generalizado para n rainhas em um tabuleiro $n \times n$.

Palavras-cruzadas

Quantas pessoas resolvem problemas de satisfação de restrições diariamente sem notar que o estão fazendo? Palavras-cruzadas, sudokus, e tantos outros passatempos de lógica são ótimos exemplos de problemas com restrições.

Um quebra-cabeça de palavras-cruzadas consiste em preencher um tabuleiro com palavras de um conjunto pré-estabelecido, como exemplificado na Figura 2.5. Esse problema pode ser modelado da seguinte forma: cada quadrado em branco é uma variável x_i que pode



HAN
JAR
POE
KYLO
LEIA
LUKE
DOOKU
LANDO
VADER
WINDU
ANAKIN
GREEDO
KENOBI
AMIDALA
SIDIOUS
TYRANUS

Figura 2.5: Uma instância do problema de palavras-cruzadas. Cada sequência contígua de quadrados (vertical ou horizontal) deve ser preenchida com alguma das palavras do conjunto ao lado. Nem todas as palavras desse conjunto devem ser utilizadas.

assumir como valor uma letra qualquer do alfabeto. No entanto, existem restrições entre quadrados de sequências contíguas na mesma linha ou coluna, de forma que estes devem ser preenchidos, simultaneamente, com letras de palavras que estão no conjunto dado. No exemplo da Figura 2.5, existe uma restrição sobre as variáveis x_3, x_4 e x_5, forçando que estas assumam alguma das seguintes combinações de letras: (H, A, N), (J, A, R) ou (P, O, E).

Satisfazibilidade proposicional

No problema da satisfazibilidade proposicional, deseja-se encontrar uma atribuição de valores-verdade a variáveis proposicionais (variáveis que podem assumir apenas o valor `verdadeiro` ou `falso`) de forma que uma dada fórmula proposicional seja satisfeita, isto é, resulte no valor-verdade `verdadeiro`.

Uma fórmula proposicional é constituída de conjunções ($\wedge$) de cláusulas, que são disjunções ($\vee$) de literais, onde um literal, por sua vez, é uma variável proposicional x_i ou a sua negação $\neg x_i$. Os conectivos $\wedge$, $\vee$ e $\neg$ são definidos conforme a Tabela 2.1, onde x_1 e x_2 são duas variáveis proposicionais. Um exemplo de fórmula proposicional é

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee x_3) \tag{2.1}$$

Neste problema, cada variável proposicional é, naturalmente, uma variável do problema de satisfação de restrições, e cada cláusula define uma restrição sobre um subconjunto dessas variáveis. Na fórmula (2.1), a primeira cláusula restringe as atribuições permitidas às variáveis x_1, x_2 e x_3, de forma que ao menos a uma delas seja atribuída o valor-verdade `verdadeiro`. Uma solução dessa instância é x_1 = `verdadeiro`, x_2 = `falso` e x_3 = `verdadeiro`.

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$	$\neg x_1$
falso	falso	falso	falso	verdadeiro
falso	verdadeiro	falso	verdadeiro	verdadeiro
verdadeiro	falso	falso	verdadeiro	falso
verdadeiro	verdadeiro	verdadeiro	verdadeiro	falso

Tabela 2.1: Conectivos lógicos de conjunção ($\wedge$), disjunção ($\vee$) e negação ($\neg$).

Sistemas de equações

Um sistema de equações consiste de um conjunto finito de m equações sobre um conjunto de n variáveis. Uma solução para um sistema de equações é uma atribuição de valores reais às variáveis satisfazendo simultaneamente todas as m equações.

$$\begin{cases} 3{,}2x_1 + 5x_2 - x_3/2 = 5{,}07 \\ 1{,}5x_1 - x_2 - x_3 = 0{,}775 \\ 5x_1 + 2{,}5x_2 + x_3 = 8{,}375 \end{cases} \tag{2.2}$$

Este problema é, evidentemente, um problema de satisfação de restrições, onde cada restrição é escrita na forma de uma equação. No exemplo da equação (2.2), a solução do sistema é dada por $x_1 = 1{,}35$, $x_2 = 0{,}25$ e $x_3 = 1$. Sistemas de equações pertencem a uma classe de problemas chamada de problemas de satisfação de restrições numéricas, pois, diferentemente dos problemas exemplificados anteriormente, permite que as variáveis possam assumir qualquer valor dentre um conjunto contínuo de números reais.

2.2.2 Definição

Nesta seção, o problema de satisfação de restrições, bem como seus conceitos fundamentais, são definidos formalmente.

Definição 2.11 (**Variável e domínio**)**.** Uma variável x representa um valor de uma coleção de objetos D, chamada domínio. Isto é, o domínio D de uma variável x é um conjunto que lista todos os valores possíveis que essa variável pode assumir; ou ainda, todos os valores que podem ser atribuídos a x.

O domínio de uma variável pode ser discreto, contínuo, finito ou infinito, e expresso via enumeração ou definido de acordo com alguma propriedade computável. Por exemplo, $D_1 = \{\texttt{verdadeiro}, \texttt{falso}\}$ e $D_2 = \{1, 2, \ldots, 10\}$ são domínios discretos e finitos, $D_3 = \{n \in \mathbb{N} \mid \text{n é par}\}$ é um domínio discreto e infinito e $D_4 = \{x \in \mathbb{R} \mid 0 \leq x < 1\}$ é um domínio contínuo. Em particular, um domínio também pode ser a união de domínios contínuos, como $D_5 = \{x \mid 0 \leq x < 1 \text{ ou } 2 \leq x < 3\}$.

Definição 2.12 (**Valoração**)**.** Dada uma variável x com domínio D, uma valoração $\langle x = a \rangle$ é uma atribuição do valor $a \in D$ à variável x, de forma que x passa a representar apenas o valor a e nenhum outro de seu domínio.

Definição 2.13 (**Restrição**)**.** Uma restrição R_C sobre um conjunto de variáveis $C = \{x_1, \ldots, x_k\}$ é uma relação $R_C \subseteq D_1 \times \cdots \times D_k$ que denota as atribuições permitidas às variáveis de C simultaneamente, onde $D_1, \ldots, D_k$ são os domínios das variáveis $x_1, \ldots, x_k$, respectivamente. O conjunto C é chamado de escopo da restrição.

Definição 2.14 (**Satisfazibilidade**)**.** Uma restrição $R_C \subseteq D_1 \times \cdots \times D_k$, tal que $C = \{x_1, \ldots, x_k\}$, é dita satisfeita pela valoração $\{\langle x_1 = a_1 \rangle, \ldots, \langle x_k = a_k \rangle\}$ se, e somente se, $(a_1, \ldots, a_k) \in R_C$.

Restrições podem ser definidas via enumeração explícita de tuplas permitidas ou por alguma expressão computável que defina essas tuplas de forma implícita (nesse caso, alguma **linguagem de restrições** é utilizada). Por exemplo, a restrição $R_C = \{(\texttt{falso}, \texttt{falso}), (\texttt{verdadeiro}, \texttt{falso}), (\texttt{verdadeiro}, \texttt{verdadeiro})\}$, sobre o conjunto de variáveis $C = \{x_1, x_2\}$, define que apenas as seguintes atribuições são válidas:

$$\{\langle x_1 = \texttt{falso} \rangle, \langle x_2 = \texttt{falso} \rangle\}$$
$$\{\langle x_1 = \texttt{verdadeiro} \rangle, \langle x_2 = \texttt{falso} \rangle\}$$
$$\{\langle x_1 = \texttt{verdadeiro} \rangle, \langle x_2 = \texttt{verdadeiro} \rangle\}$$

Alternativamente, R_C pode ser expressa utilizando lógica proposicional: $R_C \equiv (x_1 \vee \neg x_2)$.

Em particular, uma restrição numérica sobre variáveis reais é escrita da forma

$$f(x_1, \ldots, x_n) = b \tag{2.3}$$

onde $f : \mathbb{R}^n \mapsto \mathbb{R}$ é uma função real e $b \in \mathbb{R}$ é constante.

Definição 2.15 (**Rede de restrições**)**.** Uma rede de restrições $\mathcal{R}$ (ou simplesmente uma rede) é uma tupla $(\mathcal{X}, \mathcal{D}, C)$, onde $\mathcal{X} = \{x_1, \ldots, x_n\}$ é um conjunto de variáveis com domínios $\mathcal{D} = \{D_1, \ldots, D_n\}$ e $C = \{R_{C_1}, \ldots, R_{C_m}\}$ é um conjunto de restrições sobre as variáveis em $\mathcal{X}$.

Definição 2.16 (**Aridade**)**.** A aridade de uma restrição R_C é a cardinalidade do seu escopo, isto é, $|C|$. Uma rede de restrições $\mathcal{R}$ é dita k-ária se todas as suas restrições têm aridade no máximo k.

É importante observar que uma variável x é dita binária se o seu domínio D contém apenas dois elementos, isto é, $|D| = 2$. Por outro lado, uma restrição R_C é dita binária se o seu escopo contém apenas duas variáveis (não necessariamente binárias), isto é, $|C| = 2$.

Usualmente, considera-se que uma rede de restrições está sempre **normalizada**, isto é, que não há duas ou mais restrições sobre um mesmo conjunto de variáveis; ou ainda, que

todos os escopos de restrições são distintos entre si. Uma rede pode ser normalizada combinando restrições com escopos iguais em uma nova restrição. Por exemplo, as restrições $R_{C_1} \equiv x_1 \leq x_2$ e $R_{C_2} \equiv x_1 \neq x_2$, onde $C_1 = C_2 = \{x_1, x_2\}$, podem ser combinadas em uma única restrição $R_{\{x_1,x_2\}} \equiv x_1 < x_2$.

Definição 2.17 (**Valoração parcial, total e consistente**)**.** Dada uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, uma valoração $I = \{\langle x_1 = a_1\rangle, \ldots, \langle x_k = a_k\rangle\}$ é dita parcial se $k < |\mathcal{X}|$, do contrário, I é dita total. Além disso, I é dita consistente com $\mathcal{R}$ (ou com um subconjunto de restrições $S \subseteq C$) se, e somente se, toda restrição $R_C \in C$ (ou $R_C \in S$), tal que $C \subseteq \{x_1, \ldots, x_k\}$, é satisfeita por I.

Definição 2.18 (**Problema de satisfação de restrições (CSP)**)**.** Um problema de satisfação de restrições (ou simplesmente CSP, do inglês *constraint satisfaction problem*) é definido por uma rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, onde D_i é finito, para todo $D_i \in \mathcal{D}$. A solução de um CSP é uma valoração total de $\mathcal{X}$ consistente com $\mathcal{R}$.

Exemplo 2.2. O problema de palavras-cruzadas, introduzido na Seção 2.2.1, é um exemplo de CSP. Em particular, a instância da Figura 2.5 pode ser representada por uma rede de restrições $\mathcal{R}_{PC} = (\mathcal{X}, \mathcal{D}, C)$, tal que

- $\mathcal{X} = \{x_1, \ldots, x_{21}\}$, onde x_i representa o quadrado i do tabuleiro.
- $\mathcal{D} = \{D_1, \ldots, D_{21}\}$, onde $D_i = \{A, \ldots, Z\}$ é o conjunto de letras maiúsculas do alfabeto.
- $C = \{R_{C_1}, \ldots, R_{C_5}\}$ é um conjunto de restrições sobre cada sequência contígua de quadrados no tabuleiro, isto é,

$$C_1 = \{x_1, x_4, x_7, x_{10}, x_{15}\} \qquad C_4 = \{x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}\}$$
$$C_2 = \{x_2, x_6, x_8, x_{14}, x_{16}, x_{18}\} \qquad C_5 = \{x_{17}, x_{18}, x_{19}, x_{20}\}$$
$$C_3 = \{x_3, x_4, x_5\}$$

  E cada restrição $R_{C_i} \in C$ enumera as possíveis combinações de letras às variáveis de C_i, de acordo com o conjunto de palavras disponíveis:

$$R_{C_1} = \{(\text{D, O, O, K, U}), (\text{L, A, N, D, O}), (\text{V, A, D, E, R}), (\text{W, I, N, D, U})\}$$
$$R_{C_2} = \{(\text{A, M, I, D, A, L, A}), (\text{S, I, D, I, O, U, S}), (\text{T, Y, R, A, N, U, S})\}$$
$$R_{C_3} = \{(\text{H, A, N}), (\text{J, A, R}), (\text{P, O, E})\}$$
$$R_{C_4} = \{(\text{A, N, A, K, I, N}), (\text{G, R, E, E, D, O}), (\text{K, E, N, O, B, I})\}$$
$$R_{C_5} = \{(\text{K, Y, L, O}), (\text{L, E, I, A}), (\text{L, U, K, E})\}$$

Seja $I = \{\langle x_1 = \text{V}\rangle, \langle x_3 = \text{P}\rangle, \langle x_4 = \text{O}\rangle, \langle x_5 = \text{E}\rangle, \langle x_7 = \text{D}\rangle, \langle x_{10} = \text{E}\rangle, \langle x_{15} = \text{R}\rangle\}$ uma valoração parcial desta rede. Enquanto a restrição R_{C_3} é satisfeita por I, o mesmo não ocorre com R_{C_1}, pois

a palavra VODER não está no conjunto de palavras disponíveis. Logo, I não é uma valoração consistente com $\mathcal{R}$. △

Definição 2.19 (**Problema de satisfação de restrições numéricas (NCSP)**)**.** Um problema de satisfação de restrições numéricas (ou simplesmente NCSP, do inglês *numerical constraint satisfaction problem*) é definido por uma rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, onde D_i é um conjunto de números reais possivelmente infinito, para todo $D_i \in \mathcal{D}$. A solução de um NCSP é uma valoração total de $\mathcal{X}$ consistente com $\mathcal{R}$.

Na literatura, NCSP é usualmente definido com domínios de variáveis sendo conjuntos contínuos de números reais, isto é, intervalos. Aqui, optou-se por manter o caso geral, onde união de intervalos (multi-intervalos) são permitidos. Por exemplo, $D = \{-3.2, -1\} \cup \{x \mid 1 \leq x \leq 4 \text{ ou } x > 10\}$ é, portanto, um domínio válido em um NCSP. Na prática, as restrições de um NCSP são da forma $f(x_1, \ldots, x_n) = b$, onde $f : \mathbb{R}^n \mapsto \mathbb{R}$ e $b \in \mathbb{R}$. Enquanto domínios finitos podem ser expressos via enumeração de valores (e, de fato, muitos algoritmos para CSPs têm complexidade em função da cardinalidade dos domínios), para domínios contínuos uma forma de representação alternativa se faz necessária. A análise intervalar, que será introduzida na Seção 2.3, provê um arcabouço de processamento de domínios contínuos e restrições numéricas de maneira eficiente.

Definição 2.20 (**Problema de otimização global (NCOP)**)**.** Um problema de otimização global (ou simplesmente NCOP, do inglês *numerical constrained global optimization problem*) é definido por uma rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, onde D_i é um conjunto de números reais possivelmente infinito, para todo $D_i \in \mathcal{D}$, e por uma função objetivo $f : \mathbb{R}^n \mapsto \mathbb{R}$. A solução de um NCOP é uma valoração total de $\mathcal{X}$ consistente com $\mathcal{R}$ e que minimiza f em $I_{\mathcal{R}}$, onde $I_{\mathcal{R}}$ é o conjunto de todas as valorações totais de $\mathcal{X}$ consistentes com $\mathcal{R}$.

Dessa forma, um problema de otimização como introduzido na Seção 1.1 pode ser escrito da seguinte forma:

$$\min \quad f(x_1, \ldots, x_n) \tag{2.4}$$

$$\text{sujeito a} \quad \mathcal{R} \tag{2.5}$$

onde $f : \mathbb{R}^n \mapsto \mathbb{R}$ é a função objetivo e $\mathcal{R}$ é uma rede de restrições numéricas.

Nota-se que a Definição 2.20 contempla tanto NLP e LP, quanto otimização combinatória, como programação inteira por exemplo, pois não restringe os domínios das variáveis a conjuntos contínuos.

2.2.3 Rede dual e hipergrafo de restrições

Para toda rede de restrições $\mathcal{R}$ existe uma **rede dual** $\mathcal{R}^{dual}$ equivalente, de forma que toda valoração consistente de $\mathcal{R}$ pode ser eficientemente convertida em uma valoração consistente

de $\mathcal{R}^{dual}$, e vice-versa. Mais especificamente, em uma rede dual, cada escopo de restrição da rede original torna-se uma variável com domínio definido pelas tuplas permitidas pela respectiva restrição, isto é, cada variável dual é uma tupla ordenada. Cria-se, então, restrições de igualdade entre os valores associados à mesma variável da rede original, por pares de tuplas.

Exemplo 2.3. Seja a rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ tal que $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$ e $C = \{R_{C_1}, R_{C_2}\}$, onde $C_1 = \{x_1, x_2, x_3\}$, $C_2 = \{x_2, x_4\}$ e

$$R_{C_1} = \{(1, 1, 2), (1, 2, 3), (2, 1, 1), (3, 2, 3)\}$$
$$R_{C_2} = \{(1, 2), (1, 3), (3, 2), (2, 3)\}$$

Em $\mathcal{R}^{dual}$ há duas variáveis: $y_1 \equiv C_1$ e $y_2 \equiv C_2$, com domínios $D_{y_1} \equiv R_{C_1}$ e $D_{y_2} \equiv R_{C_2}$, isto é, y_1 é uma tripla ordenada, enquanto y_2 é um par ordenado. No entanto, existe uma restrição entre os valores que podem ser atribuídos a y_1 e y_2; por exemplo, se $\langle y_1 = (1, 1, 2)\rangle$, então apenas as atribuições $\langle y_2 = (1, 2)\rangle$ e $\langle y_2 = (1, 3)\rangle$ são válidas, pois o segundo elemento da tupla $(1, 1, 2)$ representa o valor da variável original x_2, que deve ser o mesmo valor do primeiro elemento da tupla atribuída a y_2 (denota-se estes elementos, respectivamente, por $y_1[x_2]$ e $y_2[x_2]$). Dessa forma, a rede dual pode ser escrita da forma $\mathcal{R}^{dual} = (\mathcal{X}^{dual}, \mathcal{D}^{dual}, C^{dual})$, onde $\mathcal{X}^{dual} = \{y_1, y_2\}$, $\mathcal{D}^{dual} = \{D_{y_1} \equiv R_{C_1}, D_{y_2} \equiv R_{C_2}\}$ e $C^{dual} = \{S_{\{y_1,y_2\}} \equiv y_1[x_2] = y_2[x_2]\}$. △

Definição 2.21 (**Rede dual**)**.** A rede dual $\mathcal{R}^{dual}$ de uma rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é definida por $\mathcal{R}^{dual} = (\mathcal{X}^{dual}, \mathcal{D}^{dual}, C^{dual})$, onde $\mathcal{X}^{dual} = \{C_i \mid R_{C_i} \in C\}$, $\mathcal{D}^{dual} = C$ e $C^{dual} = \{S_{\{C_i,C_j\}} \equiv C_i[x_k] = C_j[x_k] \mid i \neq j, R_{C_i}, R_{C_j} \in C \text{ e } x_k \in (C_i \cap C_j)\}$.

Exemplo 2.4. A instância de palavras-cruzadas da Figura 2.5, cuja rede de restrições é definida no Exemplo 2.2, possui a seguinte rede dual: $\mathcal{R}^{dual}_{PC} = (\mathcal{X}^{dual}, \mathcal{D}^{dual}, C^{dual})$, onde

$$\begin{aligned}
\mathcal{X}^{dual} &= \{y_1 \equiv C_1, \ldots, y_5 \equiv C_5\} \\
\mathcal{D}^{dual} &= \{D_{y_1} \equiv R_{C_1}, \ldots, D_{y_5} \equiv R_{C_5}\} \\
C^{dual} &= \{S_{\{y_1,y_3\}} \equiv y_1[x_4] = y_3[x_4], \\
&\qquad S_{\{y_1,y_4\}} \equiv y_1[x_{10}] = y_4[x_{10}], \\
&\qquad S_{\{y_2,y_4\}} \equiv y_2[x_{14}] = y_4[x_{14}], \\
&\qquad S_{\{y_2,y_5\}} \equiv y_2[x_{18}] = y_5[x_{18}]\}
\end{aligned}$$

Intuitivamente, na representação dual cada variável y_i é um espaço contíguo de quadrados, vertical ou horizontal, e seu domínio é o conjunto de palavras com as quais aquele espaço pode ser preenchido. Por exemplo, a variável dual y_1 representa o espaço formado pelos quadrados $\{1, 4, 7, 10, 15\}$ (ver Figura 2.5), e seu domínio consiste das palavras compostas por 5 letras: {(D, O, O, K, U), (L, A, N, D, O), (V, A, D, E, R), (W, I, N, D, U)}, isto é, y_1 é uma tupla com 5 elementos. Por outro lado, as restrições duais garantem que não haja sobreposição

de letras nos quadrados que pertencem a dois ou mais espaços de palavras. Por exemplo, a restrição $S_{\{y_1,y_3\}} \equiv y_1[x_4] = y_3[x_4]$ garante que as palavras escritas nos espaços de quadrados $\{1, 4, 7, 10, 15\}$ e $\{3, 4, 5\}$ sejam preenchidos com a mesma letra no quadrado 4. △

Redes duais apresentam a interessante característica de serem sempre binárias, pois suas restrições são definidas como restrições de igualdade (Definição 2.21). No entanto, nem sempre essa representação facilita a interpretação do problema. Se as restrições da rede original são expressas utilizando alguma linguagem de restrições, como restrições numéricas, por exemplo, então o domínio de cada variável dual também é definido de forma implícita por alguma expressão. Consequentemente, uma enumeração de atribuições possíveis exige o processamento de restrições de forma equivalente em ambas as representações.

Redes de restrições são geralmente associadas a hipergrafos, onde vértices representam variáveis e arestas representam escopos de restrições. Essa representação permite analisar a estrutura das relações entre variáveis que compartilham restrições na rede. Além disso, algoritmos e conceitos de hipergrafos, como ciclicidade, busca e conexidade, passam a ser aplicados no cenário de CSP. Ao representar uma rede de restrições por um hipergrafo, não se está interessado nas relações definidas pelas restrições, isto é, no conjunto de tuplas de atribuições permitidas por elas, mas nos conjuntos de variáveis que definem seus escopos.

Definição 2.22 (**Hipergrafo de restrições).** Dada uma rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, define-se o hipergrafo de restrições de $\mathcal{R}$ (ou simplesmente o hipergrafo de $\mathcal{R}$) como o hipergrafo $\mathcal{H} = (\mathcal{X}, C)$, onde $C = \{C_i \mid R_{C_i} \in C\}$, isto é, cada variável em $\mathcal{X}$ é um vértice em $\mathcal{H}$ e cada escopo de restrição em C define uma aresta no hipergrafo.

Nota-se que quando $\mathcal{R}$ é uma rede binária, o seu hipergrafo de restrições é 2-uniforme e, portanto, pode ser chamado simplesmente de grafo de restrições de $\mathcal{R}$ (ou grafo de $\mathcal{R}$). Os conceitos de grafo primal e dual são naturalmente estendidos a redes de restrições. Em particular, o grafo dual $\mathcal{H}^{dual}$ de uma rede de restrições $\mathcal{R}$ é exatamente o grafo de restrições do dual $\mathcal{R}^{dual}$ dessa rede. A Figura 2.6 apresenta o hipergrafo da rede de restrições da instância de palavras-cruzadas da Figura 2.5 (Exemplo 2.2), juntamente com o hipergrafo da sua rede dual, dada no Exemplo 2.4.

Teorema 2.1. *Dada uma rede de restrições $\mathcal{R}$ e seu hipergrafo $\mathcal{H}$, o grafo $\mathcal{H}^{dual}$ é idêntico ao grafo de restrições da rede $\mathcal{R}^{dual}$.*

Demonstração. Seja $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ uma rede de restrições com hipergrafo $\mathcal{H}$, isto é, $\mathcal{H} = (\mathcal{X}, \{C_i \mid R_{C_i} \in C\})$. Pela Definição 2.8, $\mathcal{H}^{dual} = (\{C_i \mid R_{C_i} \in C\}, \{\{C_i, C_j\} \mid R_{C_i}, R_{C_j} \in C, C_i \neq C_j \text{ e } C_i \cap C_j \neq \emptyset\})$. Por outro lado, pela Definição 2.21, $\mathcal{R}^{dual} = (\{C_i \mid R_{C_i} \in C\}, C, \{S_{\{C_i,C_j\}} \equiv C_i[x_k] = C_j[x_k] \mid i \neq j, R_{C_i}, R_{C_j} \in C \text{ e } x_k \in (C_i \cap C_j)\})$ e o seu grafo de restrições é dado por $G = (\{C_i \mid R_{C_i} \in C\}, \{\{C_i, C_j\} \mid i \neq j, R_{C_i}, R_{C_j} \in C \text{ e } \exists x_k \in (C_i \cap C_j)\})$. Logo, G e $\mathcal{H}^{dual}$ são idênticos. □

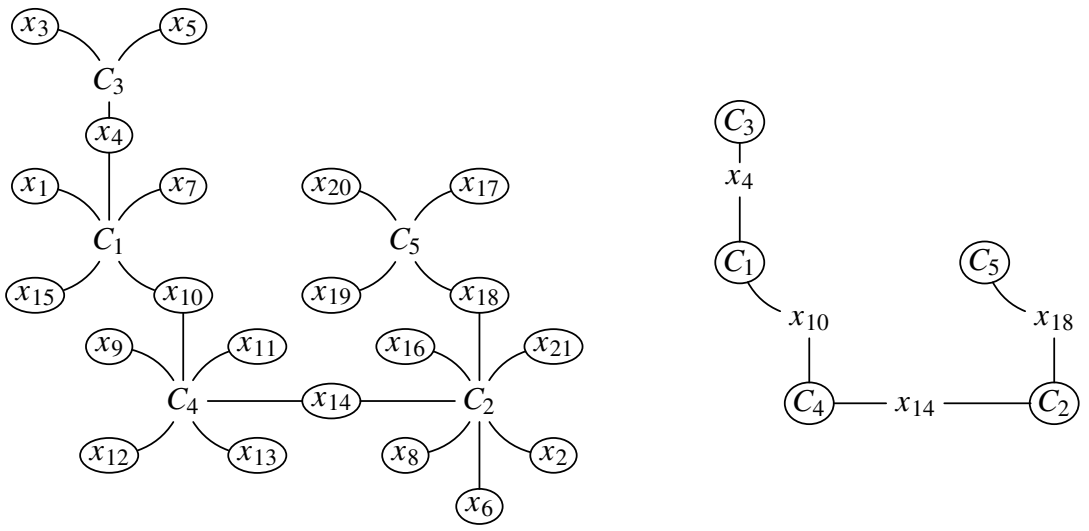


Figura 2.6: (a) Hipergrafo da rede de restrições da instância de palavras-cruzadas da Figura 2.5 (Exemplo 2.2) e o hipergrafo dual desta rede (b), que é idêntico ao hipergrafo da rede dual da mesma instância (Exemplo 2.4).

Por convenção, nesta tese, os termos "variável" e "vértice" são usualmente utilizados como sinônimos, bem como os termos "restrição" e "aresta".

2.2.4 Decomposição de restrições k-árias

Algumas classes de restrições k-árias podem ser decompostas em um conjunto de restrições de aridade menor. Restrições binárias são vastamente estudadas no campo da satisfação de restrições, pois detêm de características que tornam a aplicação de técnicas de inferência mais natural do que no caso geral, de forma que muitos conceitos de processamento de restrições foram inicialmente propostos para este caso restrito. No entanto, nem sempre é possível representar uma restrição k-ária por uma rede binária equivalente com o mesmo conjunto de variáveis.

Teorema 2.2. *O número de restrições sobre k variáveis com domínios de cardinalidade d é maior que o número de redes de restrições binárias sobre k variáveis com domínios de cardinalidade d.*

Demonstração. Prova-se essa afirmação mostrando que (i) o número de restrições sobre k variáveis com domínios de cardinalidade d é 2^{d^k} e (ii) o número de redes de restrições binárias sobre k variáveis com domínios de cardinalidade d é $2^{k^2 d^2}$.

(i) como cada uma das k variáveis pode assumir d valores de seu domínio, existem d^k tuplas de atribuições possíveis a essas k variáveis. Cada restrição é definida por um subconjunto dessas tuplas e, portanto, cada tupla pode ou não pertencer à relação. Logo, existem 2^{d^k} combinações de tuplas possíveis.

(ii) dadas as k variáveis, existem k^2 pares possíveis de variáveis. Para cada par, existe d^2 combinações possíveis de atribuições (cada variável do par pode ser atribuída a d

valores de seu domínio). Mas uma rede não necessariamente precisa ter todos os k^2 pares: cada par pode ou não estar na rede. Logo, existem $(2^{k^2})^{d^2}$ redes distintas.

□

Uma rede k-ária $\mathcal{R}$ pode ser aproximada por uma rede binária, chamada **rede de projeção** de $\mathcal{R}$, cujo conjunto de soluções contém o conjunto de soluções da rede original.

Exemplo 2.5. Seja R_C uma restrição tal que $C = \{x_1, x_2, x_3\}$ e $R_C = \{(1,1,2), (1,2,2), (2,1,3), (2,2,2)\}$. Supondo que os domínios das variáveis são $D_1 = D_2 = D_3 = \{1,2,3\}$, o conjunto de soluções da rede constituída unicamente por R_C é dado por

$$\begin{aligned} S = \{&\{\langle x_1 = 1\rangle, \langle x_2 = 1\rangle, \langle x_3 = 2\rangle\}, \\ &\{\langle x_1 = 1\rangle, \langle x_2 = 2\rangle, \langle x_3 = 2\rangle\}, \\ &\{\langle x_1 = 2\rangle, \langle x_2 = 1\rangle, \langle x_3 = 3\rangle\}, \\ &\{\langle x_1 = 2\rangle, \langle x_2 = 2\rangle, \langle x_3 = 2\rangle\}\} \end{aligned}$$

A rede de projeção da restrição R_C é dada por: $p(R_C) = \{P_{C_1}, P_{C_2}, P_{C_3}\}$, onde $C_1 = \{x_1, x_2\}$, $C_2 = \{x_1, x_3\}$, $C_3 = \{x_2, x_3\}$ e as restrições são definidas, respectivamente, pelos conjuntos $P_{C_1} = \{(1,1), (1,2), (2,1), (2,2)\}$, $P_{C_2} = \{(1,2), (2,3), (2,2)\}$ e $P_{C_3} = \{(1,2), (2,2), (1,3)\}$. O conjunto de soluções da rede $p(R_C)$ é um superconjunto do conjunto de soluções S da rede original, dado por $S \cup \{\{\langle x_1 = 2\rangle, \langle x_2 = 1\rangle, \langle x_3 = 2\rangle\}\}$. △

Alternativamente, restrições podem ser representadas por um conjunto de restrições de aridade menor adicionando-se novas variáveis à rede. Em alguns casos, no entanto, o número de variáveis adicionadas, juntamente com as novas restrições criadas, resultam em uma rede de tamanho exponencial em relação à restrição original. Esse é o caso, por exemplo, da representação de uma rede de restrições proposicionais (cláusulas) k-árias por uma conjunção equivalente de cláusulas binárias (Garey e Johnson, 1979).

Por outro lado, a transformação adicionando-se variáveis auxiliares é em geral eficiente quando considera-se como resultado uma rede de restrições ternárias. Em particular, restrições numéricas definidas por **funções fatoráveis** são eficientemente convertidas em um conjunto de restrições elementares constituídas de no máximo três variáveis (ou constantes) e um operador binário ou unário.

Definição 2.23 (**Função fatorável**)**.** (Hascoët et al., 2013) Uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$ é dita fatorável se pode ser computada por uma sequência finita de funções $(f_1, \ldots, f_n, f_{n+1}, \ldots, f_{n+c}, f_{n+c+1}, \ldots, f_{n+c+L})$, onde $f_i : \mathbb{R}^n \mapsto \mathbb{R}$, para todo $1 \leq i \leq n+c+L$, é tal que

1. $f_i(\mathbf{x}) = x_i$, se $1 \leq i \leq n$;
2. $f_i(\mathbf{x}) = a_i$, $a_i \in \mathbb{R}$, se $n+1 \leq i \leq n+c$;

3. $f_i(\mathbf{x}) = f_{j<i}(\mathbf{x}) \circ f_{k<i}(\mathbf{x})$ **ou**

   $f_i(\mathbf{x}) = g_i(f_{j<i}(\mathbf{x})),$ se $n + c + 1 \leq i \leq n + c + L;$

onde $\circ \in \{+, -, \cdot, /, \hat{}, \sqrt{}\}$ e g_i é uma função de transformação como exp, sin, cos, tan, ou outra operação unária computável.

Exemplo 2.6. Seja a restrição numérica $x^2 - \sqrt[3]{(xy + z)} \leq 5w$. Adicionando novas variáveis $u_1, \ldots, u_5$, essa restrição pode ser decomposta em um conjunto equivalente de restrições numéricas ternárias[1]:

$$\begin{array}{lll} u_1 - u_2 \leq u_3 & u_2 = \sqrt[3]{u_4} & u_5 = xy \\ u_1 = x^2 & u_4 = u_5 + z & u_3 = 5w \end{array}$$

△

A decomposição ternária pode ser obtida diretamente da sequência $(f_{n+c+1}, \ldots, f_{n+c+L})$ apresentada na Definição 2.23, criando uma restrição ternária[2] para cada função $f_i(\mathbf{x}) = f_{j<i}(\mathbf{x}) \circ f_{k<i}(\mathbf{x})$ ou $f_i(\mathbf{x}) = g_i(f_{j<i}(\mathbf{x}))$. Dessa forma, a representação ternária contém L restrições. Por outro lado, o número de símbolos necessários para representar uma restrição n-ária da forma $f(x_1, \ldots, x_n) = b$ é pelo menos $n_o + c + L$, onde n_o é o número de ocorrências das variáveis $x_1, \ldots, x_n$ na expressão de f, c é o número de constantes e L é o número de operações.

Mais detalhes sobre a decomposição ternária de redes numéricas podem ser encontrados em Sam (1995) e Faltings e Gelle (1997).

2.3 Análise Intervalar

A análise intervalar é estudada desde a década de 1960, quando foi proposta por Moore (1966) como uma abordagem formal à análise de propagação de erros de arredondamento em sistemas computacionais. Limitando valores reais não representáveis por números de máquina aproximados inferior e superiormente, Moore definiu a álgebra intervalar como uma extensão da álgebra sobre números reais, na qual cada valor é representado por um intervalo de precisão bem determinada; dessa forma, operações aritméticas sobre estes valores consideram, no resultado, as margens de precisão de cada operando. O estudo de Moore resultou em uma área de pesquisa bastante ampla, originando a revista *Interval Computations* (editora *Springer*) em 1991, mais tarde renomeada *Reliable Computing*. Desde então, a análise intervalar vem sendo aplicada nas mais diversas áreas, como robótica, redes neurais, mecânica quântica, sistemas temporais e, de forma mais geral, utilizada como forma de representação e processamento de problemas de satisfação de restrições numéricas.

[1] Considerando que constantes são substituídas por variáveis com domínios unitários.

[2] Uma restrição da forma $u = g(x)$, onde g é uma operação unária computável, pode ser escrita da forma ternária $u = x \circ_g y$, definindo-se que $x \circ_g y = g(x)$, para todo $y \in \mathbb{R}$.

Intervalos não são apenas uma notação para representar conjuntos contínuos de valores reais, mas de fato uma extensão dos números reais. Em Edmonson e van Emden (2008), uma extensão do padrão IEEE para aritmética de ponto flutuante definiu os principais indeterminantes envolvendo 0 e $\pm\infty$, como $\infty - \infty$, $0/0$ e $0 \cdot \infty$, justificados pela extensão intervalar dos números reais. Embora tais operações não façam sentido na aritmética convencional, sua aplicação no cenário intervalar é recorrente e de grande importância, como será mostrado no decorrer desta tese.

Através de uma rigorosa abordagem matemática, o livro pioneiro *Interval Analysis*, de Moore (1966), apresenta não apenas os conceitos base da análise intervalar, mas também sua aplicação na análise de erros numéricos, no cálculo diferencial e integral e, mais recentemente, em Moore et al. (2009), na otimização global. Outra importante publicação desse cenário é o livro de Hansen e Walster (2004), publicado originalmente em 1992, cujo primeiro parágrafo do prefácio, escrito por Moore, prepara o leitor para entender a álgebra intervalar como extensão da aritmética convencional:

> *Take note,* ***mathematicians****. Here you will find a new extension of real arithmetic to interval arithmetic for containment sets (csets) in which there are no undefined operand-operator combinations such as previously "****indeterminate forms****"* $0/0$*,* $\infty - \infty$*, etc.* (Hansen e Walster, 2004)

Nesta seção, as principais definições sobre intervalos e análise intervalar são elencadas.

2.3.1 Definição

Intervalos são definidos sobre o conjunto estendido dos números reais.

Definição 2.24 (**Conjunto estendido dos números reais**)**.** O conjunto estendido dos números reais $\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}$ é definido adicionando-se os símbolos de infinito negativo ($-\infty$) e infinito positivo ($+\infty$) ao conjunto dos números reais $\mathbb{R}$, de forma que $\forall x \in \mathbb{R} : -\infty < x < +\infty$.

Operações aritméticas são parcialmente estendidas de $\mathbb{R}$ a $\overline{\mathbb{R}}$ da seguinte forma:

$$
\begin{aligned}
|\pm\infty| &= +\infty && \\
-(+\infty) &= -\infty && \\
-(-\infty) &= +\infty && \\
x + (+\infty) = +\infty + x &= +\infty && \text{se } x \neq -\infty \\
x + (-\infty) = -\infty + x &= -\infty && \text{se } x \neq +\infty \\
x \cdot (\pm\infty) = \pm\infty \cdot x &= \pm\infty && \text{se } x > 0 \\
x \cdot (\pm\infty) = \pm\infty \cdot x &= \mp\infty && \text{se } x < 0 \\
x/(\pm\infty) &= 0 && \text{se } x \neq \pm\infty \\
|x/0| &= +\infty && \text{se } x \neq 0
\end{aligned}
\tag{2.6}
$$

As expressões $+\infty + (-\infty)$, $-\infty + (+\infty)$, $0/0$ e $\pm\infty/\pm\infty$ são indefinidas. A expressão $x/0$, para $x \neq 0$, é parcialmente definida, pois pode ser interpretada tanto como $-\infty$, quanto como $+\infty$. Nesta tese, a expressão $0 \cdot (\pm\infty)$ é definido como 0, como proposto em McShane (1983).

Definição 2.25 (**Intervalo**)**.** Um intervalo X é um conjunto contínuo de números reais, que pode ser fechado (2.7), aberto (2.8), fechado à esquerda (2.9) ou fechado à direita (2.10), onde $\underline{X}, \overline{X} \in \overline{\mathbb{R}}$ são, respectivamente, os extremos esquerdo e direito de X.

$$[\underline{X}, \overline{X}] = \{x \in \mathbb{R} \mid \underline{X} \leq x \leq \overline{X}\} \tag{2.7}$$

$$(\underline{X}, \overline{X}) = \{x \in \mathbb{R} \mid \underline{X} < x < \overline{X}\} \tag{2.8}$$

$$[\underline{X}, \overline{X}) = \{x \in \mathbb{R} \mid \underline{X} \leq x < \overline{X}\} \tag{2.9}$$

$$(\underline{X}, \overline{X}] = \{x \in \mathbb{R} \mid \underline{X} < x \leq \overline{X}\} \tag{2.10}$$

Por convenção, se $\underline{X} = -\infty$, então X é aberto ou fechado à direita, e se $\overline{X} = +\infty$, então X é aberto ou fechado à esquerda. É importante observar que $(-\infty, +\infty) = \mathbb{R}$ e que, se $\underline{X} = +\infty$ ou $\overline{X} = -\infty$ ou $\overline{X} < \underline{X}$ ou $\underline{X} = \overline{X}$ e X não é fechado, então $X = \emptyset$.

Definição 2.26 (**Conjunto de intervalos em** $\mathbb{R}$)**.** O conjunto de todos os intervalos em $\mathbb{R}$ é definido por $\mathbb{I} = \{[\underline{X}, \overline{X}], (\underline{X}, \overline{X}], [\underline{X}, \overline{X}), (\underline{X}, \overline{X}) \mid \underline{X}, \overline{X} \in \overline{\mathbb{R}}\}$.

Definição 2.27 (**Funções** τ_e **e** τ_d)**.** As funções $\tau_e : \mathbb{I} \mapsto \{0, 1\}$ e $\tau_d : \mathbb{I} \mapsto \{0, 1\}$ associam dois valores $\tau_e(X)$ e $\tau_d(X)$ a um intervalo $X \in \mathbb{I}$ de forma que

$$\tau_e(X) = \begin{cases} 0 & \text{se } X \text{ é aberto ou fechado à direita} \\ 1 & \text{caso contrário} \end{cases} \tag{2.11}$$

$$\tau_d(X) = \begin{cases} 0 & \text{se } X \text{ é aberto ou fechado à esquerda} \\ 1 & \text{caso contrário} \end{cases} \tag{2.12}$$

Definição 2.28 (**Comprimento**)**.** O comprimento de um intervalo X é dado por $c(X) = \overline{X} - \underline{X}$.

Intervalos permitem a representação de valores reais com uma margem de aproximação, cuja precisão é dada pelo comprimento de cada intervalo. Por exemplo, o valor $\sqrt{2}$ pode ser representado pelo intervalo $X = [1{,}4, 1{,}5]$, onde $c(X) = 0{,}1$, ou pelo intervalo mais preciso $Y = [1{,}414, 1{,}415]$, onde $c(Y) = 0{,}001$.

Na literatura existem diversas definições sobre a comparação entre intervalos. Em Suprajitno e bin Mohd (2010) é apresentada uma análise detalhada das definições mais comuns e suas possíveis aplicações. Por outro lado, para dois intervalos X e Y quaisquer, as relações $X = Y$, $X \neq Y$, $X \subset Y$, $X \subseteq Y$ e as operações $X \cap Y$, $X \cup Y$, $X \setminus Y$ são definidas de forma idêntica às respectivas relações e operações sobre conjuntos. Além disso, como intervalos são conjuntos contínuos de valores reais, é possível verificar quaisquer dessas relações apenas observando os extremos dos intervalos. Por exemplo, $X = Y$ se, e somente se, $\underline{X} = \underline{Y}$, $\overline{X} = \overline{Y}$, $\tau_e(X) = \tau_e(Y)$ e $\tau_d(X) = \tau_d(Y)$.

Definição 2.29 (**Refinamento e expansão**)**.** O intervalo X é um refinamento do intervalo Y se $X \subseteq Y$. Nesse caso, Y é uma expansão de X.

Definição 2.30 (**Caixa**)**.** Uma caixa $B \in \mathbb{I}^n$ é uma tupla de intervalos. Uma caixa $B = (X_1, \ldots, X_n)$ é um refinamento da caixa $D = (Y_1, \ldots, Y_n)$, e denota-se esse fato por $B \subseteq D$, se, e somente se, $X_i \subseteq Y_i$, para todo $1 \leq i \leq n$.

Definição 2.31 (**Intervalo degenerado**)**.** Um intervalo fechado X é dito degenerado se $\underline{X} = \overline{X}$.

Um intervalo degenerado é um conjunto de tamanho unitário que representa um número real com erro nulo, isto é, o único elemento do intervalo é o próprio número real que se deseja representar. Na literatura, intervalos degenerados e números reais são tratados de forma equivalente ($[x, x] \equiv x$), pois mesmo que um intervalo degenerado ainda seja um conjunto, ele representa um número real exatamente. Considerando a álgebra intervalar como extensão da álgebra sobre os números reais, intervalos degenerados são a interseção entre esses dois domínios.

Dados dois intervalos X e Y, a extensão intervalar de qualquer operador binário $\circ$ bem definido em $\mathbb{R}$ é dada por:

$$X \circ Y = \{x \circ y \mid x \in X,\ y \in Y \text{ e } x \circ y \text{ está definido em } \mathbb{R}\} \tag{2.13}$$

O conjunto (2.13) pode ser um intervalo, o conjunto vazio ou um conjunto não contínuo de números reais. É fácil verificar que essa definição detém a propriedade de **inclusão isotônica**, isto é, se $X' \subseteq X$ e $Y' \subseteq Y$, então $(X' \circ Y') \subseteq (X \circ Y)$. Se X e Y são degenerados, então $X \circ Y$ também resulta em um intervalo degenerado (se $X \circ Y \neq \emptyset$). Além disso, é possível computar $X \circ Y$, para todas as operações algébricas e as principais transcedentais, apenas observando os extremos de X e Y (Moore et al., 2009). Por exemplo, as operações de adição, subtração, multiplicação e divisão intervalares são dadas, respectivamente, por:

$$[\underline{X}, \overline{X}] + [\underline{Y}, \overline{Y}] = [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}] \tag{2.14}$$

$$[\underline{X}, \overline{X}] - [\underline{Y}, \overline{Y}] = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}] \tag{2.15}$$

$$[\underline{X}, \overline{X}] \cdot [\underline{Y}, \overline{Y}] = [\min\{\underline{XY}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{XY}\}, \max\{\underline{XY}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{XY}\}] \tag{2.16}$$

$$[\underline{X}, \overline{X}] \,/\, [\underline{Y}, \overline{Y}] = [\underline{X}, \overline{X}] \cdot [1/\overline{Y}, 1/\underline{Y}], \text{ se } 0 \notin [\underline{Y}, \overline{Y}] \tag{2.17}$$

As operações de potenciação, radiciação e outras funções algébricas e trigonométricas podem ser definidas de maneira análoga, analisando os extremos dos intervalos, com base na equação (2.13). As operações apresentadas nas equações (2.14) a (2.17) são fechadas em $\mathbb{I}$. A divisão por um intervalo que contém zero, no entanto, pode não resultar em um intervalo, pois $X/Y = \{x/y \mid x \in X,\ y \in Y \text{ e } y \neq 0\}$ não é necessariamente um conjunto contínuo de valores reais.

Definição 2.32 (**Multi-intervalo**)**.** Um multi-intervalo $X = \langle X_1, \ldots, X_n \rangle$ é o conjunto de números reais $X = \bigcup_{i=1}^{n} X_i$, onde $X_i \in \mathbb{I}$.

Dessa forma, um multi-intervalo é um conjunto finito e ordenado de intervalos que podem ser disjuntos entre si. Em particular, qualquer intervalo X é representado pelo multi-intervalo $\langle X \rangle$, de forma que o termo "multi-intervalo" também engloba intervalos de maneira geral. Como a Definição 2.32 não impõe restrição quanto aos intervalos que compõem um multi-intervalo, é possível representar o mesmo conjunto de valores por diferentes multi-intervalos. Por exemplo, os conjuntos $\langle [1, 3], [4, 9) \rangle$, $\langle [1, 2), [2, 3], [4, 9) \rangle$ e $\langle [4, 9), [2, 3], [1, 3) \rangle$ são equivalentes. Sem perda de generalidade, reduz-se os multi-intervalos à sua representação mínima e ordenada, de forma que se $X = \langle X_1, \ldots, X_n \rangle \neq \emptyset$ então, para todo $1 \leq i \leq n : X_i \neq \emptyset$ e

i) $\overline{X_i} < \underline{X_{i+1}}$ **ou**

ii) $\overline{X_i} = \underline{X_{i+1}}$ e $\tau_d(X_i) = \tau_e(X_{i+1}) = 0$

Definição 2.33 (**Conjunto de multi-intervalos em** $\mathbb{R}$**).** O conjunto de todos os multi-intervalos em $\mathbb{R}$ é definido por $\mathbb{M} = \{\langle X_1, \ldots, X_n \rangle \mid n \in \mathbb{N}^* \text{ e } X_i \in \mathbb{I}\}$.

Definição 2.34 (**Comprimento de multi-intervalos**)**.** O comprimento de um multi-intervalo $X = \langle X_1, \ldots, X_n \rangle$ é dado por $c(X) = \sum_{i=1}^{n} c(X_i)$.

Comparações e operações envolvendo multi-intervalos seguem de forma análoga aos intervalos, exceto pelo fato de que, em geral, é necessário analisar toda a combinação de intervalos contidos em cada multi-intervalo. Assim, dados $X = \langle X_1, \ldots, X_m \rangle$, $Y = \langle Y_1, \ldots, Y_n \rangle$ e uma operação binária $\circ$ bem definida em $\mathbb{R}$ ou sobre conjuntos (isto é, $\cap$, $\cup$ e $\setminus$), define-se que

$$X \circ Y = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n} X_i \circ Y_j \tag{2.18}$$

garantindo que as operações multi-intervalares são fechadas em $\mathbb{M}$.

2.3.2 Aritmética intervalar

Como mencionado na Seção 2.3.1, operações aritméticas bem definidas em $\mathbb{R}$ podem ser estendidas ao conjunto dos (multi-)intervalos. Nesta seção, são apresentadas as principais propriedades dessa extensão, considerando as operações de adição, subtração, multiplicação, divisão, potenciação e radiciação. Considera-se, para tanto, que os operandos são intervalos fechados e não vazios; no entanto, para intervalos abertos ou semi-abertos, as definições seguem de forma análoga (por exemplo, se um operador binário $\circ$, aplicado sobre dois intervalos X e Y, tais que X é fechado à esquerda e Y é fechado, resulta em um intervalo Z com o extremo esquerdo calculado em função de $\underline{X}$ e $\overline{Y}$ e o extremo direito calculado em função de $\overline{X}$ e $\overline{Y}$, então Z será fechado à esquerda, isto é, $\tau_e(Z) = \tau_e(X) \cdot \tau_d(Y) = 1$ e $\tau_d(Z) = \tau_d(X) \cdot \tau_d(Y) = 0$). Por fim, se um dos operandos for o intervalo vazio, então o resultado também será o intervalo vazio.

Adição:

$$X + Y = [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}] \tag{2.19}$$

A adição é comutativa e associativa e o intervalo $[0, 0]$ é o elemento neutro da adição.

Exemplo 2.7. $[-1, 3] + [2, +\infty) + [0, 0] = [-1, 3] + [2, +\infty) = [1, +\infty)$. △

Subtração:

$$X - Y = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}] \tag{2.20}$$

A subtração é associativa e também pode ser escrita definindo-se a operação unária $-Y = [-\overline{Y}, -\underline{Y}]$, de forma que $X - Y = X + (-Y) = [\underline{X}, \overline{X}] + [-\overline{Y}, -\underline{Y}] = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}]$. É importante observar que $X - X = [\underline{X} - \overline{X}, \overline{X} - \underline{X}]$, e portanto, $X - X = [0, 0]$ se, e somente se, $\underline{X} = \overline{X}$, isto é, X é degenerado. Do contrário, $[0, 0] \subset X - X$.

Exemplo 2.8. $[0, +\infty) - [0, 3] = [-3, +\infty)$. △

Multiplicação:

$$X \cdot Y = [\min\{\underline{XY}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{XY}, \}, \max\{\underline{XY}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{XY}\}] \tag{2.21}$$

A multiplicação é comutativa, associativa e subdistributiva em relação à adição e a subtração, isto é, $X \cdot (Y \pm Z) \subset (X \cdot Y \pm X \cdot Z)$. O intervalo $[1, 1]$ é o elemento neutro da multiplicação. Como definiu-se $0 \cdot \pm\infty = 0$, então $(-\infty, +\infty) \cdot [0, 0] = [0, 0]$, o que convém com a intuição de que $\forall x \in (-\infty, +\infty) : x \cdot 0 = 0$.

Exemplo 2.9. $[0, 2] \cdot [-1, +\infty) = [\min\{0, 0, -2, +\infty\}, \max\{0, 0, -2, +\infty\}) = [-2, +\infty)$. △

Divisão:

$$X/Y = \begin{cases} X \cdot [1/\overline{Y}, 1/\underline{Y}] & \text{se } \underline{Y} > 0 \text{ ou } \overline{Y} < 0 \\ X \cdot (-\infty, 1/\underline{Y}] & \text{se } \underline{Y} < \overline{Y} = 0 \\ X \cdot [1/\overline{Y}, +\infty) & \text{se } 0 = \underline{Y} < \overline{Y} \\ X \cdot \langle(-\infty, 1/\underline{Y}], [1/\overline{Y}, +\infty)\rangle & \text{se } \underline{Y} < 0 < \overline{Y} \\ \emptyset & \text{se } \underline{Y} = \overline{Y} = 0 \end{cases} \tag{2.22}$$

Da mesma forma que na subtração, $X/X = [1, 1]$ se, e somente se, $\underline{X} = \overline{X}$. Do contrário, $[1, 1] \subset X/X$.

Exemplo 2.10.

$$\begin{aligned}(-\infty, 3]/(-\infty, -1] &= (-\infty, 3] \cdot [1/(-1), 0) \\ &= (-\infty, 3] \cdot [-1, 0) \\ &= [\min\{+\infty, 0, -3, 0\}, \max\{+\infty, 0, -3, 0\}) \\ &= [-3, +\infty)\end{aligned}$$

△

Potenciação (expoente inteiro):

$$X^a = \begin{cases} [1,1]/X^{-a} & \text{se } a < 0 \\ [\underline{X}^a, \overline{X}^a] & \text{se } a > 0 \text{ é ímpar} \\ [\underline{X}^a, \overline{X}^a] & \text{se } a > 0 \text{ é par e } \underline{X} \geq 0 \\ [\overline{X}^a, \underline{X}^a] & \text{se } a > 0 \text{ é par e } \overline{X} < 0 \\ [0, \max\{\underline{X}^a, \overline{X}^a\}] & \text{se } a > 0 \text{ é par e } \underline{X} < 0 \leq \overline{X} \end{cases} \tag{2.23}$$

Exemplo 2.11.

$$\begin{aligned}(-\infty, -2]^{-3} &= [1,1]/((-\infty, -2]^3) \\ &= [1,1]/((-\infty)^3, (-2)^3] \\ &= [1,1]/(-\infty, -8] \\ &= [1,1] \cdot [1/(-8), 0) \\ &= [-1/8, 0)\end{aligned}$$

△

Na aritmética intervalar, $X^2 \subseteq X \cdot X$.

Radiciação (índice inteiro):

$$\sqrt[a]{X} = \begin{cases} [1,1]/\sqrt[-a]{X} & \text{se } a < 0 \\ [\sqrt[a]{\underline{X}}, \sqrt[a]{\overline{X}}] & \text{se } a > 0 \text{ é ímpar} \\ \langle[-\sqrt[a]{\overline{X}}, -\sqrt[a]{\underline{X}}], [\sqrt[a]{\underline{X}}, \sqrt[a]{\overline{X}}]\rangle & \text{se } a > 0 \text{ é par e } \underline{X} \geq 0 \\ \emptyset & \text{se } a > 0 \text{ é par e } \overline{X} < 0 \\ \sqrt[a]{[0, \overline{X}]} & \text{se } a > 0 \text{ é par e } \underline{X} < 0 \leq \overline{X} \end{cases} \tag{2.24}$$

Exemplo 2.12.

$$\begin{aligned}\sqrt[-3]{[1,+\infty)} &= [1,1]/(\sqrt[3]{[1,+\infty)}) \\ &= [1,1]/[\sqrt[3]{1},\sqrt[3]{+\infty}) \\ &= [1,1]/[1,+\infty) \\ &= [1,1]\cdot(0,1] \\ &= (0,1]\end{aligned}$$

△

2.3.3 Funções Intervalares

Uma função intervalar $\mathcal{F} : \mathbb{M} \mapsto \mathbb{M}^n$ é um mapeamento de n multi-intervalos $X_1, \ldots, X_n$ em um outro multi-intervalo $Y = \mathcal{F}(X_1, \ldots, X_n)$. As operações intervalares definidas na Seção 2.3.2, dessa forma, são casos específicos de funções intervalares, pois mapeiam seus dois operandos X_1 e X_2 em um multi-intervalo $Y = X_1 \circ X_2$. Em geral, qualquer função fatorável possui uma extensão intervalar em $\mathbb{I}$ ou $\mathbb{M}$.

Definição 2.35 (Conjunto imagem). Dada uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$ e um conjunto $B \subseteq \mathbb{R}^n$, a expressão $f[B]$ denota o conjunto imagem de f em B, isto é, $f[B] = \{f(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in B\}$.

Definição 2.36 (Extensão intervalar). Dada uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$, diz-se que $\mathcal{F} : \mathbb{M}^n \mapsto \mathbb{M}$ é uma extensão intervalar de f se $\forall B \in \mathbb{M}^n : f[B] \subseteq \mathcal{F}(B)$, isto é,

$$\forall X_1, \ldots, X_n \in \mathbb{M} : \{f(x_1, \ldots, x_n) \mid x_1 \in X_1, \ldots, x_n \in X_n\} \subseteq \mathcal{F}(X_1, \ldots, X_n) \tag{2.25}$$

Em outras palavras, uma função $\mathcal{F}$ é uma extensão intervalar da função f se $\mathcal{F}(B)$ é um multi-intervalo que contém o conjunto imagem de f em B. Essa definição permite a existência de inúmeras extensões intervalares de uma mesma função real.

Exemplo 2.13. Seja a função fatorável $f(x_1, x_2) = {x_1}^2 - x_2$ e três funções intervalares $\mathcal{F}_1(X_1, X_2) = {X_1}^2 - X_2$, $\mathcal{F}_2(X_1, X_2) = X_1 \cdot X_1 - X_2$ e $\mathcal{F}_3(X_1, X_2) = (-\infty, +\infty)$. Estas três funções são extensões intervalares da função f. Em particular, considerando os domínios $D_1 = [-2, 2]$ e $D_2 = [-1, 1]$ às variáveis x_1 e x_2, é fácil verificar que $f[D_1, D_2] = [-1, 5]$, enquanto as funções intervalares computam:

$$\begin{aligned}&\mathcal{F}_1(D_1, D_2) = [-2,2]^2 - [-1,1] = [0,4] - [-1,1] = [-1,5] = f[D_1, D_2] \\ &\mathcal{F}_2(D_1, D_2) = [-2,2]\cdot[-2,2] - [-1,1] = [-4,4] - [-1,1] = [-5,5] \supseteq f[D_1, D_2] \\ &\mathcal{F}_3(D_1, D_2) = (-\infty,\infty) \supseteq f[D_1, D_2]\end{aligned}$$

△

A extensão intervalar **natural** de uma função fatorável $f : \mathbb{R}^n \mapsto \mathbb{R}$ é uma função intervalar $\mathcal{F} : \mathbb{M}^n \mapsto \mathbb{M}$ composta por um mapeamento de cada operação elementar da expressão de f às respectivas operações intervalares na expressão de $\mathcal{F}$. No Exemplo 2.13, a função $\mathcal{F}_1$ é a extensão intervalar natural de f, enquanto $\mathcal{F}_2$ e $\mathcal{F}_3$ não são. Essa extensão possui a propriedade de inclusão isotônica (se $B, B' \in \mathbb{M}^n$, então $B' \subseteq B \Rightarrow \mathcal{F}(B') \subseteq \mathcal{F}(B)$). Se toda variável ocorre uma única vez na expressão de uma função f, então a sua extensão intervalar natural $\mathcal{F}$ computa exatamente o conjunto imagem de f; do contrário, $\mathcal{F}$ computa um superconjunto do conjunto imagem de f (Benhamou e Granvilliers, 2006). Esse problema é chamado de **problema da dependência** e ocorre porque a aritmética intervalar interpreta cada ocorrência da variável como uma variável distinta. Por exemplo, a extensão intervalar da função $f(x) = x \circ x$ é definida, pela expressão (2.13), como $\mathcal{F}(X) = X \circ X = \{x_1 \circ x_2 \mid x_1 \in X, x_2 \in X\}$; por outro lado, o conjunto imagem de f em X é definido por $f[X] = \{f(x_1) \mid x_1 \in X\} = \{x_1 \circ x_1 \mid x_1 \in X\}$ e, portanto, $f[X] \subseteq \mathcal{F}(X)$. O problema da dependência também fica evidente, por exemplo, na definição do operador intervalar de subtração, na Seção 2.3.2, onde, no caso geral, $X - X \neq [0, 0]$.

Exemplo 2.14. Sejam as funções fatoráveis $f_1(x_1, x_2) = {x_1}^2 - x_2$ e $f_2(x_1, x_2) = x_1 \cdot x_1 - x_2$, e suas respectivas extensões intervalares naturais $\mathcal{F}_1(X_1, X_2) = {X_1}^2 - X_2$ e $\mathcal{F}_2(X_1, X_2) = X_1 \cdot X_1 - X_2$. Considerando os domínios $D_1 = [-2, 2]$ e $D_2 = [-1, 1]$ para as variáveis x_1 e x_2, os conjuntos imagem dessas funções são idênticos: $f_1[D_1, D_2] = f_2[D_1, D_2] = [-1, 5]$. No entanto, as extensões intervalares computam intervalos distintos:

$$\mathcal{F}_1(D_1, D_2) = [-2, 2]^2 - [-1, 1] = [0, 4] - [-1, 1] = [-1, 5]$$
$$\mathcal{F}_2(D_1, D_2) = [-2, 2] \cdot [-2, 2] - [-1, 1] = [-4, 4] - [-1, 1] = [-5, 5]$$

△

Com o objetivo de reduzir o problema da dependência, outras extensões intervalares são propostas na literatura, como as extensões baseadas na monotocidade da função (Araya et al., 2012) e as extensões de Taylor (Kearfott e Walster, 2002). Uma revisão sobre essas extensões pode ser encontrada em Araya e Reyes (2016).

Alternativamente, funções fatoráveis podem ser decompostas em um conjunto de expressões elementares cujas variáveis ocorrem uma única vez. Essa estratégia é semelhante ao processo de transformar restrições k-árias em ternárias, descrito na Seção 2.2.4. Embora o problema de dependência seja resolvido e, portanto, a extensão intervalar natural de cada sub-expressão computa exatamente o seu conjunto imagem, a combinação dos multi-intervalos resultantes ainda pode superestimar (no sentido de ser um superconjunto) o conjunto imagem da função original. Esse problema é chamado de **problema da localidade**, e é inerente ao processo de computar cada expressão separadamente. Em geral, a decomposição de expressões transforma o problema da dependência no problema da localidade e vice-versa.

Exemplo 2.15. Seja a função fatorável $f(x_1) = x_1 \cdot x_1 + x_1$ e o domínio $D_1 = [-2, 2]$ à variável x_1. Essa função pode ser decomposta, de forma equivalente, no seguinte conjunto de expressões elementares (binárias ou ternárias):

$$\begin{aligned} f(x_1) &= f_1(x_1) + x_1 \\ f_1(x_1) &= f_2(x_1) \cdot x_1 \\ f_2(x_1) &= x_1 \end{aligned}$$

Considerando as extensões intervalares naturais $\mathcal{F}(X_1) = \mathcal{F}_1(X_1) + X_1$, $\mathcal{F}_1(X_1) = \mathcal{F}_2(X_1) \cdot X_1$ e $\mathcal{F}_2(X_1) = X_1$, obtém-se:

$$\begin{aligned} \mathcal{F}_2(D_1) &= D_1 = [-2, 2] \\ \mathcal{F}_1(D_1) &= \mathcal{F}_2(D_1) \cdot D_1 = [-2, 2] \cdot [-2, 2] = [-4, 4] \\ \mathcal{F}(D_1) &= \mathcal{F}_1(D_1) + D_1 = [-4, 4] + [-2, 2] = [-6, 6] \end{aligned}$$

E portanto, $\mathcal{F}(D_1) = [-6, 6]$ superestima o (é um superconjunto do) conjunto imagem $f[D_1] = [-2, 6]$. △

2.4 Conclusão

Neste capítulo, apresentou-se algumas definições acerca de conjuntos, grafos e hipergrafos. Em seguida, introduziu-se conceitos fundamentais de problemas de satisfação de restrições, apresentando alguns exemplos e as suas principais definições; como rede de restrições, rede dual, decomposição de restrições, CSP, NCSP, NCOP e a representação de CSPs através de grafos e hipergrafos. Na Seção 2.3, apresentou-se a aritmética intervalar como extensão da aritmética convencional. Definiu-se os conceitos de intervalo, multi-intervalo, funções intervalares, extensão intervalar e as operações intervalares básicas. Em particular, mostrou-se como a extensão intervalar pode ser utilizada para estimar o conjunto imagem de uma função e a relação entre o problema da localidade e o problema da dependência.

De modo geral, este capítulo focou na modelagem de problemas reais através de restrições. No Capítulo 3, técnicas de resolução desses modelos serão apresentadas, completando, assim, uma visão geral acerca da programação por restrições.

3 PROCESSAMENTO DE RESTRIÇÕES

A programação por restrições consiste em modelar problemas reais por meio de restrições e resolvê-los utilizando técnicas de processamento de restrições. No processo de modelagem, é necessário identificar qual o objetivo do problema, usualmente definido como: (a) determinar se o problema possui pelo menos uma solução; (b) encontrar uma ou todas as soluções do problema; ou (c) encontrar uma ou mais soluções ótimas do problema, de acordo com algum critério bem definido. No primeiro caso, tem-se um problema de decisão comumente chamado de satisfazibilidade, enquanto o caso (b) contempla os problemas de satisfação de restrições introduzidos no Capítulo 2, como CSP e NCSP; já o caso (c) exige a representação do problema real como um problema de otimização, ou NCOP, no qual uma função objetivo é utilizada como critério de otimalidade de soluções.

Métodos para resolver problemas com restrições podem ser específicos de domínio ou de propósito geral. No primeiro caso, algoritmos utilizam-se de informações e propriedades que instâncias de um determinado domínio apresentam para resolvê-las mais rapidamente. Por exemplo, o algoritmo Simplex para resolver programação linear (Dantzig, 1963), a eliminação de Fourier para resolver inequações lineares (Fourier, 1824) ou o algoritmo DPLL para o problema da satisfazibilidade proposicional (Davis et al., 1962). Por outro lado, métodos de propósito geral utilizam-se de técnicas de processamento de restrições livres de domínio, baseados em duas principais abordagens: busca e inferência. Embora não tirem proveito de particularidades do domínio do problema, estas técnicas não se restringem a uma classe específica de instâncias, sendo a principal ferramenta para resolver problemas com restrições no caso geral.

Neste capítulo, métodos de processamento de restrições de propósito geral são elencados. Em particular, foca-se em técnicas de inferência, também chamada de propagação de restrições, que garantem algum nível de consistência local entre subconjuntos de restrições da rede. No cenário contínuo, a análise intervalar mostra-se uma ferramenta fundamental para o processamento de restrições, pois é capaz de representar e processar restrições de maneira eficiente.

Este capítulo está dividido em cinco seções principais:

3.1. **Busca *versus* inferência**: apresenta-se uma visão geral de métodos de busca para CSPs e técnicas que aprimoram a busca. As principais referências são os livros de Dechter (2003) e Norvig e Russell (2017);

3.2. **Técnicas de consistência para CSPs finitos**: apresenta-se uma revisão das principais técnicas de consistência propostas na literatura para CSPs finitos. Toma-se como

referência os trabalhos que introduzem as respectivas consistências, citadas no texto, e as revisões apresentadas em Dechter (2003), Vu (2005) e Bessiere (2006);

3.3. **Técnicas de consistência para CSPs numéricos**: apresenta-se uma revisão das principais técnicas de consistência propostas na literatura para CSPs numéricos. Além das referências citadas no texto, toma-se como referência Benhamou et al. (1999), Cruz e Barahona (2001), Vu (2005) e Benhamou e Granvilliers (2006);

3.4. **Métodos intervalares aplicados a problemas numéricos**: métodos de ramificação e poda intervalares para resolver NCSP e NCOP são apresentados. As principais referências são van Hentenryck et al. (1997), Hansen e Walster (2004), Benhamou e Granvilliers (2006) e Araya e Reyes (2016);

3.5. **Busca livre de retrocesso**: apresenta-se o conceito de consistência direcionada e algumas classes de CSPs e NCSPs que são resolvidas sem a necessidade de retrocesso se alcançado um determinado nível de consistência. Métodos de decomposição de CSPs também são revisados. As principais referências estão citadas no texto, incluindo o livro de Dechter (2003) e os artigos que propõem as relações entre estrutura do problema e nível de consistência que garante solução livre de retrocesso.

3.1 Busca *versus* inferência

Nas últimas décadas, diversas técnicas foram propostas para resolver problemas de satisfação de restrições. Em geral, essas técnicas constituem duas classes principais de algoritmos: **busca** e **inferência**.

Um algoritmo de busca para CSP consiste em gerar valorações de variáveis (parciais ou totais) e verificar se cada valoração gerada é uma solução da instância, caso no qual o algoritmo encerra-se com sucesso, ou se implica algum conflito na rede de restrições e, portanto, não pode caracterizar uma solução, mesmo que parcialmente. Este processo é comumente representado graficamente por uma árvore de busca, na qual nós representam valorações parciais e arestas conectam valorações que estendem valorações anteriores. Os nós terminais dessa árvore representam valorações totais ou inconsistentes.

O algoritmo de busca mais simples que se pode obter é a **enumeração exaustiva**, também conhecida como "busca por força bruta", na qual todas as valorações totais possíveis são geradas e verificadas, uma a uma, até que se encontre uma que caracterize uma solução do CSP. Evidentemente, em instâncias descritas de maneira explícita (por enumeração de domínios), uma busca por enumeração exaustiva consome, no pior caso, tempo exponencial no tamanho da instância: em uma rede com n variáveis, cada uma com domínio de cardinalidade d, existem d^n valorações possíveis. Para instâncias descritas de forma implícita, essa complexidade pode ser ainda pior.

A **busca com retrocesso**, termo cunhado por Lehmer (1957), é um refinamento da busca por enumeração exaustiva que constrói valorações parciais de forma iterativa, seguindo uma ordenação total de variáveis: dada uma valoração parcial consistente I, obtém-se uma nova valoração estendendo-se I à próxima variável da ordenação ainda não valorada, atribuindo-lhe um valor arbitrário de seu domínio. Quando uma valoração parcial é detectada inconsistente, a atribuição da última variável é desfeita e outro valor lhe é atribuído. Se todos os valores já foram tentados, o método retrocede para a variável anterior e prossegue de forma análoga. Esse algoritmo possui dois estados terminas: quando uma valoração total consistente é alcançada, caracterizando uma solução da instância, ou quando todos os valores da primeira variável foram testados e, para todas as suas valorações, houve a ocorrência de um conflito em algum nível da árvore de busca; nesse caso, a instância não tem solução.

O Algoritmo 1 descreve um método de busca com retrocesso para resolver CSP. O algoritmo recebe como entrada uma rede de restrições $\mathcal{R}$ e uma ordenação de variáveis b e, a cada iteração, escolhe do domínio da variável x_i um valor que torna a valoração $I \cup \{\langle x_i = a_i \rangle\}$ consistente (linha 8). Este valor é então excluído de D_i para que, se houver um retrocesso, a valoração $I \cup \{\langle x_i = a_i \rangle\}$ não volte a ocorrer (linha 10). Nota-se que o algoritmo executa com cópias dos domínios originais, pois quando um retrocesso ocorre, todas as alterações de domínios de variáveis sucessoras em b devem ser desfeitas (linha 12).

Algoritmo 1 Busca com retrocesso para CSP (`retrocesso`$(\mathcal{R}, b)$)

Entrada: Rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ e ordenação de variáveis $b = (x_1, \ldots, x_n)$.
Saída: I é uma valoração total consistente com $\mathcal{R}$.

```
 1: I ← ∅
 2: i ← 1
 3: D_1' ← D_1
 4: enquanto 1 ≤ i ≤ n faça
 5:     se ∄a_i ∈ D_i' tal que I ∪ {⟨x_i = a_i⟩} é consistente com R então
 6:         i ← i − 1    // retrocesso
 7:     senão
 8:         seja a_i ∈ D_i' tal que I ∪ {⟨x_i = a_i⟩} é consistente com R
 9:         I ← I ∪ {⟨x_i = a_i⟩}
10:         D_i' ← D_i' \ {a_i}
11:         i ← i + 1
12:         D_i' ← D_i
13:     fim se
14: fim enquanto
15: se i = 0 então
16:     devolve INCONSISTENTE
17: senão
18:     devolve I
19: fim se
```

No pior caso, a busca com retrocesso consome tempo exponencial no tamanho da instância, mas a vantagem de detectar inconsistências parciais torna sua execução mais rápida que

uma busca por "força bruta" padrão. No melhor caso, o retrocesso não ocorre (o contador i nunca é decrementado na linha 6 do Algoritmo 1) e uma solução é encontrada em tempo polinomial.

Definição 3.1 (Rede livre de retrocesso). Uma rede $\mathcal{R}$ é dita livre de retrocesso em relação a uma ordenação de variáveis $b = (x_1, \ldots, x_n)$ se, para todo $1 < i \leq n$, toda valoração parcial consistente $I = \{\langle x_1 = a_1\rangle, \ldots, \langle x_{i-1} = a_{i-1}\rangle\}$ pode ser consistentemente estendida à variável x_i, isto é, se existe $a_i \in D_i$ tal que $I \cup \{\langle x_i = a_i\rangle\}$ é consistente com $\mathcal{R}$.

Uma condição suficiente para busca livre de retrocesso, embora não seja condição necessária, é a chamada consistência global.

Definição 3.2 (Consistência global). Uma rede $\mathcal{R}$ é dita globalmente consistente se, para todo $1 < i \leq n$, toda valoração parcial consistente de $i-1$ variáveis quaisquer pode ser consistentemente estendida a qualquer outra i-ésima variável da rede.

Diversas técnicas podem ser implementadas para aprimorar uma busca com retrocesso. De modo geral, pode-se classificar essas técnicas em dois grupos:

- **olhar para frente:** técnicas que processam a rede antes de a busca avançar para a valoração da próxima variável. A principal abordagem utilizada é não fixar a ordenação de variáveis no início da busca e construí-la dinamicamente conforme a busca avança. Assim, o algoritmo faria uso de um procedimento `escolhe_variavel`, responsável por decidir qual a próxima variável a ser valorada. Heurísticas livres de domínio são usualmente aplicadas nessa decisão, como escolher a variável com o menor número de valores consistentes em seu domínio, ou a variável envolvida no maior número de restrições sobre variáveis não valoradas. Outro procedimento que pode ser implementado com base em heurísticas é a escolha de valor consistente a ser atribuído à variável (linha 8 do Algoritmo 1), como escolher o valor que elimina o menor número de valores consistentes em domínios de outras variáveis, ou o valor mais recentemente atribuído à variável em tentativas anteriores de valoração (nas quais houve retrocesso).

  Uma outra técnica fundamental desse grupo, que potencializa a aplicação de heurísticas de escolha de variável e de valor, é aplicar, a cada iteração do laço do Algoritmo 1, métodos de inferência que removem valores inconsistentes de domínios de variáveis ainda não valoradas. Há diversas técnicas de inferência, ou propagação de restrições, que, combinadas com a busca com retrocesso, obtêm bons resultados na resolução de CSPs. Nas seções 3.2 e 3.3 algumas dessas técnicas serão elencadas.

- **olhar para trás:** técnicas que processam a rede antes de efetuar um retrocesso. Na prática, muitas vezes um conflito não foi ocasionado pela valoração da variável imediatamente anterior à atual e, portanto, retroceder e trocar o valor desta variável ocasionará no mesmo conflito. Apenas quando todas as valorações forem testadas e o algoritmo retroceder para um nível ainda mais anterior é que o conflito poderá ser

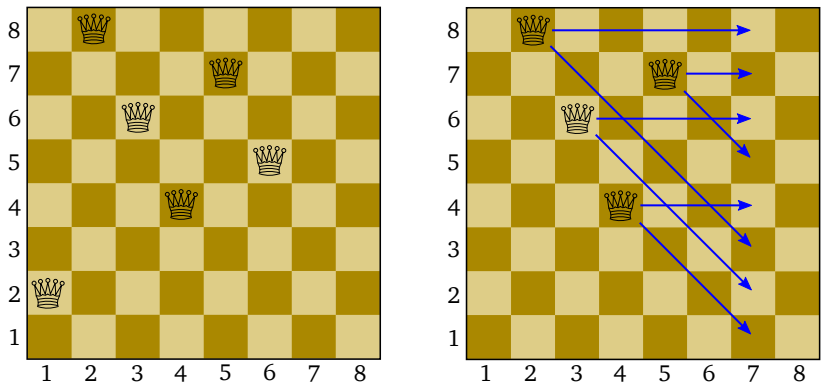


Figura 3.1: Exemplo de situação que permite retrocesso não cronológico. Nesta disposição de peças todas as linhas da coluna 7 estão sendo ameaçadas por rainhas das colunas 2 a 5 e, portanto, não basta alterar apenas a posição da sexta rainha; a busca deve retroceder 2 níveis na árvore de busca, alterando a posição da rainha 5.

resolvido. Por exemplo, a Figura 3.1 mostra um tabuleiro de xadrez com 6 rainhas no qual não há posição consistente na coluna 7 para que uma sétima rainha seja posicionada. Em uma busca com retrocesso usual, a sexta rainha (coluna 6) seria reposicionada e uma nova tentativa de valoração da sétima rainha ocorreria. No entanto, pode-se notar que o conflito não é ocasionado pela sexta rainha, pois todas as linhas da coluna 7 estão sendo atacadas por rainhas das colunas 2 a 5. Logo, o retrocesso pode retornar 2 níveis na árvore de busca, alterando a posição da quinta rainha. Este processo chama-se **retrocesso não cronológico**, e está relacionado também com o conceito de **aprendizado por conflitos**. Supondo-se que, em determinado momento, a busca retroceda até a primeira rainha posicionada no tabuleiro (coluna 1) e altere a sua posição; e supondo-se também que exista uma valoração parcial consistente de 6 rainhas na qual apenas a posição da primeira rainha mude; nesse caso, como as rainhas das colunas 2 a 5 estão nas mesmas posições, o mesmo conflito com relação à sétima rainha ocorre e, portanto, a busca é redundante. Alternativamente, no aprendizado por conflitos, ao descobrir que a causa do conflito foi a valoração $\{\langle x_2 = 8\rangle, \langle x_3 = 6\rangle, \langle x_4 = 4\rangle, \langle x_5 = 7\rangle\}$, adiciona-se à rede uma nova restrição garantindo que esta valoração não volte a ocorrer:

$$R_{\{x_2,x_3,x_4,x_5\}} \equiv (x_2 \neq 8) \vee (x_3 \neq 6) \vee (x_4 \neq 4) \vee (x_5 \neq 7)$$

Em suma, resolver um CSP utilizando apenas busca é uma abordagem, em geral, ineficiente. Do mesmo modo, tornar uma rede globalmente consistente utilizando apenas inferência demanda tempo de processamento e espaço de memória exponenciais no tamanho da instância, embora exista uma classe de problemas para os quais esse processo é polinomial (Seção 3.5.2). Em geral, busca e inferência costumam ser combinadas.

Nas Seções 3.2 e 3.3, apresenta-se uma revisão das principais técnicas de inferência que garantem algum nível de consistência em redes, respectivamente, finitas e numéricas.

3.2 Técnicas de consistência para CSPs finitos

Problemas de satisfação de restrições surgiram no contexto de problemas finitos, isto é, problemas nos quais variáveis podem assumir valores de um domínio discreto e finito. Em particular, técnicas de consistência foram inicialmente propostas em redes binárias, logo estendidas a redes de aridade qualquer. Embora conceitos fundamentais da consistência aplicada no cenário contínuo tenham sido emprestados do caso discreto, a grande maioria de técnicas previamente propostas não puderam ser eficientemente estendidas para o domínio dos números reais, de forma que novas abordagens, sobretudo utilizando-se da análise intervalar, foram desenvolvidas.

Nesta seção, as principais técnicas de consistência para problemas finitos são elencadas, enquanto as consistências para problemas numéricos serão abordadas na Seção 3.3. De modo geral, uma instância CSP (finito) é descrita por uma enumeração explícita de domínios e restrições. Dessa forma, o tamanho de uma instância pode ser expresso em função de 4 variáveis:

- n: número de variáveis na rede;
- m: número de restrições na rede;
- d: tamanho do maior domínio na rede;
- k: maior aridade de restrição na rede.

O número de tuplas nas restrições também conta no tamanho da instância, mas este número é limitado superiormente por d^k.

3.2.1 Consistência de arco

A consistência de arco é a forma mais simples de propagar restrições, reduzindo o domínio de variáveis inconsistentes com alguma restrição na rede. Informalmente, a consistência de arco verifica a existência de algum valor em um domínio de variável que não pode satisfazer uma dada restrição. Por exemplo, a restrição $x_1 < x_2$ não é arco-consistente sob os domínios $D_1 = \{1, 2, 3\}$ e $D_2 = \{2, 3\}$, pois para $\langle x_1 = 3\rangle$ não existe valor para x_2 que satisfaça a inequação; assim, o valor 3 pode ser excluído do domínio de x_1 sem alterar o conjunto de soluções da instância.

O conceito de consistência de arco foi introduzido por Waltz (1972, 1975) e formalizado por Mackworth (1977a). É o primeiro algoritmo de propagação de restrições conhecido, originalmente proposto para interpretar cenas com poliedros. Ao estabelecer restrições entre

arestas vizinhas de um poliedro desenhado, o método de Waltz foi capaz de reduzir o número de rótulos com os quais cada aresta poderia ser classificada a fim de interpretar a imagem como um objeto possível do mundo real. Assim, há uma propagação de restrições em cada aresta. Waltz evidenciou, empiricamente, que para algumas classes de poliedros o algoritmo resolve o problema sem efetuar retrocessos. Em 1977, Mackworth formalizou o algoritmo de Waltz sob a nomenclatura consistência de arco, introduzindo os algoritmos `AC-1`, `AC-2` e `AC-3` para alcançar essa consistência. A complexidade desses algoritmos foi analisada em Mackworth e Freuder (1985), onde estabeleceu-se que a consistência de arco é suficiente para resolver instâncias acíclicas, caracterizando a classe de figuras que Waltz resolveu sem a necessidade de retrocesso.

Definição 3.3 (Consistência de arco).

- Uma restrição binária $R_{\{x_i,x_j\}}$ é arco-consistente em relação à variável x_i se, e somente se, para todo valor $a_i \in D_i$ existe algum valor $a_j \in D_j$ tal que a valoração $\{\langle x_i = a_i\rangle, \langle x_j = a_j\rangle\}$ satisfaz $R_{\{x_i,x_j\}}$;
- Uma restrição binária $R_{\{x_i,x_j\}}$ é arco-consistente (total) se, e somente se, $R_{\{x_i,x_j\}}$ é arco-consistente em relação a x_i e x_j;
- Uma rede binária $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é arco-consistente se, e somente se, toda restrição $R_C \in C$ é arco-consistente.

Verificar se uma restrição binária $R_{\{x_i,x_j\}}$ é arco-consistente consome tempo $O(|D_i|\cdot|D_j|)$, pois, no caso geral, todas as combinações de valores devem ser analisadas. Dessa forma, verificar se uma rede $\mathcal{R}$ é arco-consistente consome tempo $O(md^2)$, onde m é o número de restrições da rede e d é o tamanho do maior domínio de variável em $\mathcal{R}$.

A consistência de arco não implica consistência global. Por exemplo, a rede composta pelas restrições $x_1 = x_2$ e $x_1 \neq x_2$ é claramente inconsistente, mas para quaisquer domínios $D_1 = D_2$ com mais de um elemento a rede é arco-consistente.

Exemplo 3.1. Seja a representação dual da instância de palavras-cruzadas da Figura 2.5, dada no Exemplo 2.4, isto é, a rede binária $\mathcal{R}_{PC}^{dual}$ sobre as variáveis $\{y_1, \ldots, y_5\}$ com domínios

$$
\begin{aligned}
D_{y_1} &= \{\text{DOOKU, LANDO, VADER, WINDU}\}\\
D_{y_2} &= \{\text{AMIDALA, SIDIOUS, TYRANUS}\}\\
D_{y_3} &= \{\text{HAN, JAR, POE}\}\\
D_{y_4} &= \{\text{ANAKIN, GREEDO, KENOBI}\}\\
D_{y_5} &= \{\text{KYLO, LEIA, LUKE}\}
\end{aligned}
$$

e com restrições

$$S_{\{y_1,y_3\}} \equiv y_1[x_4] = y_3[x_4]$$
$$S_{\{y_1,y_4\}} \equiv y_1[x_{10}] = y_4[x_{10}]$$
$$S_{\{y_2,y_4\}} \equiv y_2[x_{14}] = y_4[x_{14}]$$
$$S_{\{y_2,y_5\}} \equiv y_2[x_{18}] = y_5[x_{18}]$$

Obs.: as tuplas de valores no domínio de cada variável foram representadas como palavras.

A restrição $S_{\{y_1,y_3\}} \equiv y_1[x_4] = y_3[x_4]$ é arco-consistente em relação à variável y_3, pois, para todo valor em seu domínio, existe um valor no domínio de y_1 que satisfaz $S_{\{y_1,y_3\}}$:

$$y_3 = \text{H}\underline{\text{A}}\text{N} \Rightarrow \{y_1 = \text{L}\underline{\text{A}}\text{NDO}, y_1 = \text{V}\underline{\text{A}}\text{DER}\}$$
$$y_3 = \text{J}\underline{\text{A}}\text{R} \Rightarrow \{y_1 = \text{L}\underline{\text{A}}\text{NDO}, y_1 = \text{V}\underline{\text{A}}\text{DER}\}$$
$$y_3 = \text{P}\underline{\text{O}}\text{E} \Rightarrow \{y_1 = \text{D}\underline{\text{O}}\text{OKU}\}$$

Por outro lado, $S_{\{y_1,y_3\}}$ não é arco-consistente em relação a y_1, pois para $\langle y_1 = \text{WINDU}\rangle$ não existe uma tupla no domínio de y_3 que satisfaça a restrição, isto é, cujo segundo elemento (referente ao quadrado x_4) seja a letra "I". Dessa forma, a palavra WINDU pode ser removida do domínio de y_1, pois garantidamente não faz parte de nenhuma solução possível. △

Como motivado pelo Exemplo 3.1, não se deseja apenas saber se uma rede $\mathcal{R}$ é arco-consistente, mas obter uma nova rede $\mathcal{R}'$ arco-consistente equivalente a $\mathcal{R}$, isto é, que detenha o mesmo conjunto solução. Dessa forma, deseja-se **alcançar** a consistência de arco. Em Mackworth (1977a), três versões de algoritmos foram propostos com esse objetivo: `AC-1`, `AC-2` e `AC-3`, sendo o `AC-3` a sua versão mais popular, vastamente implementada e adaptada em diversas extensões da consistência de arco. Os três algoritmos fazem uso de uma rotina chamada `revisa`, descrita pelo Algoritmo 2, que alcança a consistência de arco de uma única restrição binária R_C, em relação a uma de suas variáveis. O algoritmo simplesmente remove do domínio dessa variável os valores que não têm correspondente que satisfaça R_C no outro domínio. Como todo par de valores é verificado, `revisa`$(R_{\{x_i,x_j\}}, D_i, D_j, x_i)$ consome tempo $O(d^2)$, onde $d = \max\{|D_i|, |D_j|\}$.

Algoritmo 2 Revisa restrição binária (`revisa`$(R_{\{x_i,x_j\}}, D_i, D_j, x_i)$)

Entrada: Restrição binária $R_{\{x_i,x_j\}}$ sobre as variáveis x_i e x_j com domínios D_i e D_j.
Saída: D_i atualizado, tal que $R_{\{x_i,x_j\}}$ é arco-consistente em relação a x_i.

```
1: para cada a_i ∈ D_i faça
2:     se não existe a_j ∈ D_j tal que {⟨x_i = a_i⟩, ⟨x_j = a_j⟩} satisfaz R_{x_i,x_j} então
3:         D_i ← D_i \ {a_i}
4:     fim se
5: fim para
```

O método AC−1, descrito pelo Algoritmo 3, aplica o procedimento revisa em todas as restrições da rede, até que não haja alteração de domínios, resultando em uma rede arco-consistente. É importante notar que uma rede com domínios vazios também é, por definição, arco-consistente.

Algoritmo 3 Consistência de arco 1 (AC−1($\mathcal{R}$))

Entrada: Rede de restrições binárias $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$.
Saída: $\mathcal{D}$ atualizado, tal que $\mathcal{R}$ arco-consistente.
1: **repete**
2: **para** toda $R_{\{x_i,x_j\}} \in C$ **faça**
3: revisa($R_{\{x_i,x_j\}}, D_i, D_j, x_i$)
4: revisa($R_{\{x_i,x_j\}}, D_i, D_j, x_j$)
5: **fim para**
6: **até que** não haja alteração nos domínios em $\mathcal{D}$

Teorema 3.1. AC−1($\mathcal{R}$) *consome tempo* $O(mnd^3)$*, onde m e n são, respectivamente, o número de restrições e de variáveis em* $\mathcal{R}$ *e d é o tamanho do maior domínio na rede.*

Demonstração. No pior caso, cada passo do laço das linhas 1 a 6 remove apenas um valor dos domínios das variáveis e o algoritmo só termina quando todos os nd valores são removidos. Além disso, em cada passo o procedimento revisa é chamado duas vezes para cada restrição, resultando em um tempo de processamento $O(mnd^3)$. □

O algoritmo AC−1 pode ser melhorado. Uma restrição não precisa ser revisada em toda iteração do laço, mas apenas quando o domínio de alguma de suas variáveis é alterado. Essa melhoria é implementada no método AC−3, descrito pelo Algoritmo 4. Uma pilha é utilizada para organizar as restrições que devem ser processadas; quando uma variável tem o seu domínio alterado pelo procedimento revisa todas as restrições que compartilham essa variável são adicionados na pilha. É importante observar que cada restrição é adicionada na pilha duas vezes, uma para cada variável do seu escopo. Além disso, quando o domínio de uma variável x_i é alterado, apenas a consistência de restrições $R_{\{x_i,x_k\}}$ em relação à variável x_k precisa ser verificada, pois o domínio de x_i já é arco-consistente. O algoritmo AC−2, proposto por Mackworth (1977a), implementa algumas melhorias entre os algoritmos AC−1 e AC−3 e não será apresentado aqui.

Teorema 3.2. AC−3($\mathcal{R}$) *consome tempo* $O(md^3)$*, onde m é o número de restrições de* $\mathcal{R}$ *e d é o tamanho do maior domínio na rede.*

Demonstração. Cada restrição é inserida na pilha 2 vezes, uma para cada variável de seu escopo. Além disso, cada vez que uma restrição é inserida pelo menos um valor foi removido do domínio de suas variáveis. Como há no máximo $2d$ valores, cada restrição é processada, no pior caso, $4d$ vezes. Como há m restrições e o processamento de cada uma consiste em uma chamada do procedimento refina, então o algoritmo demanda tempo de processamento $O(md^3)$. □

Algoritmo 4 Consistência de arco 3 (AC−3($\mathcal{R}$))

Entrada: Rede de restrições binárias $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$.
Saída: $\mathcal{D}$ atualizado, tal que $\mathcal{R}$ é arco-consistente.

```
 1: para cada R_{x_i,x_j} ∈ C faça
 2:     empilha((R_{x_i,x_j}, x_i), pilha)
 3:     empilha((R_{x_i,x_j}, x_j), pilha)
 4: fim para
 5: enquanto pilha ≠ ∅ faça
 6:     (R_{x_i,x_j}, x_i) ← desempilha(pilha)
 7:     revisa(R_{x_i,x_j}, D_i, D_j, x_i)
 8:     se houve alteração em D_i então
 9:         para cada R_{x_i,x_k} ∈ C tal que k ≠ j faça
10:             empilha((R_{x_i,x_k}, x_k), pilha)
11:         fim para
12:     fim se
13: fim enquanto
```

Exemplo 3.2. Considerando novamente a rede dual $\mathcal{R}_{PC}^{dual}$ da instância de palavras-cruzadas da Figura 2.5, mostrou-se, no Exemplo 3.1, que a consistência de arco da restrição $S_{\{y_1,y_3\}}$ é alcançada removendo-se a palavra WINDU do domínio da variável y_1. Seguindo a estratégia do algoritmo AC−3, como o domínio de y_1 foi alterado, todas as restrições sobre y_1, exceto $S_{\{y_1,y_3\}}$, são adicionadas na pilha de restrições a serem processadas, que nesse caso é apenas o par $(S_{\{y_1,y_4\}}, y_4)$ (vale lembrar que no início do algoritmo todas as restrições são inseridas na pilha). Esse par é então processado (retirado da pilha) e o procedimento revisa$(S_{\{y_1,y_4\}}, D_{y_1}, D_{y_4}, y_4)$ remove de D_{y_4} as palavras ANAKIN e GREEDO, pois apenas para $\langle y_4 = \text{KENOBI}\rangle$ existe uma tupla em D_{y_1} satisfazendo a restrição $S_{\{y_1,y_4\}}$. Da mesma forma, o par $(S_{\{y_2,y_4\}}, y_2)$ é o próximo a ser processado. Nesta nova iteração, a única tupla que permanece em D_{y_2} é SIDIOUS, pois apenas ela tem correspondente em D_{y_4} satisfazendo $S_{\{y_2,y_4\}}$. Como o domínio de y_2 foi alterado, o par $(S_{\{y_2,y_5\}}, y_5)$ é processado em seguida, onde o domínio D_{y_5} é reduzido à única palavra LUKE. A variável y_5 não pertence a nenhuma outra restrição e, portanto, não há nova inserção. Os pares que continuam na pilha são aqueles inseridos no início do algoritmo, excetuando-se os pares $(S_{\{y_1,y_3\}}, y_1)$ e $(S_{\{y_1,y_3\}}, y_3)$, processados no Exemplo 3.1. Supõe-se então o par $(S_{\{y_1,y_4\}}, y_1)$ como o próximo a ser processado. Nesse caso, apenas a valoração $\langle y_1 = \text{VADER}\rangle$ tem valor correspondente em $D_{y_4} = \{\text{KENOBI}\}$ satisfazendo a restrição $S_{\{y_1,y_4\}}$. Com a redução do domínio D_{y_1}, o par $(S_{\{y_1,y_3\}}, y_3)$ é o próximo processado, no qual D_{y_3} é reduzido ao conjunto {HAN, JAR}. O restante dos pares na pilha são processados de forma análoga, mas não há mais redução de domínios e, portanto, nenhum novo par é empilhado. A Tabela 3.1 mostra as restrições processadas e os domínios reduzidos em cada iteração deste exemplo. △

Outros algoritmos para alcançar a consistência de arco foram propostos na literatura, mantendo a tradição de nomenclatura: AC−4 (Mohr e Henderson, 1986), AC−6 (Bessière, 1994), AC−2001 (Bessière e Régin, 2001; Zhang e Yap, 2001), etc. Dentre estes, destaca-se o AC−4 por apresentar uma complexidade de tempo $O(md^2)$, que é o mesmo custo de apenas verificar se

iteração	restrição processada	domínio reduzido pelo procedimento `revisa`
1	$(S_{\{y_1,y_3\}}, y_3)$	–
2	$(S_{\{y_1,y_3\}}, y_1)^*$	$D_{y_1} \leftarrow \{$(LANDO), (DOOKU), (VADER)$\}$
3	$(S_{\{y_1,y_4\}}, y_4)^*$	$D_{y_4} \leftarrow \{$(KENOBI)$\}$
4	$(S_{\{y_2,y_4\}}, y_2)^*$	$D_{y_2} \leftarrow \{$(SIDIOUS)$\}$
5	$(S_{\{y_2,y_5\}}, y_5)$	$D_{y_5} \leftarrow \{$(LUKE)$\}$
6	$(S_{\{y_1,y_4\}}, y_4)$	–
7	$(S_{\{y_1,y_4\}}, y_1)^*$	$D_{y_1} \leftarrow \{$(VADER)$\}$
8	$(S_{\{y_1,y_3\}}, y_3)$	$D_{y_3} \leftarrow \{$(HAN), (JAR)$\}$
9	$(S_{\{y_2,y_4\}}, y_4)$	–
10	$(S_{\{y_2,y_4\}}, y_2)$	–
11	$(S_{\{y_2,y_5\}}, y_5)$	–
12	$(S_{\{y_2,y_5\}}, y_2)$	–

Tabela 3.1: Execução do algoritmo `AC-3` no Exemplo 3.2, supondo que no início a pilha é configurada da seguinte maneira: $pilha = ((S_{\{y_1,y_3\}}, y_3), (S_{\{y_1,y_3\}}, y_1), (S_{\{y_1,y_4\}}, y_4), (S_{\{y_1,y_4\}}, y_1), (S_{\{y_2,y_4\}}, y_4), (S_{\{y_2,y_4\}}, y_2), (S_{\{y_2,y_5\}}, y_5), (S_{\{y_2,y_5\}}, y_2))$. O marcador $*$ indica as restrições que causaram uma inserção na pilha devido à redução de domínios efetuada pelo procedimento `revisa`; consequentemente, a restrição a ser processada em seguida é justamente a qual foi empilhada.

a rede é arco-consistente, embora, experimentalmente, apresente um desempenho médio abaixo do `AC-3` (Wallace, 1993). Diferentemente dos métodos `AC-1`, `AC-2` e `AC-3`, o algoritmo `AC-4` não utiliza o procedimento `revisa`; alternativamente, a estrutura de cada restrição é explorada, armazenando um vetor de listas de pares variável-valor que têm suporte em alguma outra variável, de forma que pares já processados em iterações anteriores são aproveitados a cada nova restrição processada.

3.2.2 Consistência de caminho

Mesmo que aplicada em toda a rede, a consistência de arco considera apenas uma restrição de cada vez. Dessa forma, algumas inconsistências aparentemente óbvias não são detectadas. Por exemplo, a rede composta pelas restrições $x_1 \neq x_2$, $x_2 \neq x_3$ e $x_3 \neq x_1$ é arco-consistente sob os domínios $D_1 = D_2 = D_3 = \{1, 2\}$, embora não possua solução, pois todas as variáveis devem assumir valores distintos entre si. Neste caso, a consistência violada é chamada de consistência de caminho, proposta por Montanari (1974) como uma extensão do algoritmo de Waltz e formalizada, como a consistência de arco, por Mackworth (1977a).

Definição 3.4 (**Consistência de caminho**)**.**

- Dada uma rede binária $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, um conjunto de duas variáveis $\{x_i, x_j\} \subset \mathcal{X}$ é caminho-consistente em relação a uma terceira variável $x_k \in \mathcal{X}$ se, e somente se, para toda valoração $\{\langle x_i = a_i\rangle, \langle x_j = a_j\rangle\}$ consistente com $\mathcal{R}$ existe um valor $a_k \in D_k$ tal que as valorações $\{\langle x_i = a_i\rangle, \langle x_k = a_k\rangle\}$ e $\{\langle x_k = a_k\rangle, \langle x_j = a_j\rangle\}$ são consistentes com $\mathcal{R}$;

- Dada uma rede binária $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, um conjunto de três variáveis $\{x_i, x_j, x_k\} \subseteq \mathcal{X}$ é dito caminho-consistente (total) se, e somente se, qualquer par de variáveis nesse conjunto é caminho-consistente em relação à terceira variável.

- Uma rede binária $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é caminho-consistente se, e somente se, todo conjunto de três variáveis $\{x_i, x_j, x_k\} \subseteq \mathcal{X}$ é caminho-consistente.

Para que uma restrição $R_{\{x_i,x_j\}}$ torne-se caminho-consistente em relação a uma variável x_k, é preciso remover da relação as tuplas (a_i, a_j) que não têm correspondentes nas relações $R_{\{x_i,x_k\}}$ e $R_{\{x_k,x_j\}}$. Dessa forma, diferentemente da consistência de arco, a consistência de caminho não reduz o domínio de variáveis inconsistentes, mas altera as próprias restrições da rede. Se a restrição $R_{\{x_i,x_j\}}$ não está originalmente na rede, ela pode ser adicionada sem alterar o conjunto solução na forma de uma **restrição universal**, isto é, que permite qualquer atribuição de valores às variáveis x_i e x_j. Ao aplicar a consistência de caminho, no entanto, essa relação universal pode ser refinada, dando origem a uma nova restrição binária na rede, como descrito pelo Algoritmo 5. No exemplo da rede $x_1 \neq x_2$, $x_2 \neq x_3$ e $x_3 \neq x_1$ com domínios $D_1 = D_2 = D_3 = \{1, 2\}$, a consistência de caminho infere, a partir das restrições $x_1 \neq x_2$ e $x_2 \neq x_3$, a restrição $x_3 = x_1$, que é conflitante com a restrição $x_3 \neq x_1$, concluindo a inconsistência da rede.

Algoritmo 5 Revisa caminho (`revisaCaminho`$(R_{\{x_i,x_j\}}, R_{\{x_i,x_k\}}, R_{\{x_k,x_j\}}, x_k, D_k)$)

Entrada: Restrições binárias $R_{\{x_i,x_j\}}$, $R_{\{x_i,x_k\}}$ e $R_{\{x_k,x_j\}}$, sobre as variáveis x_i, x_j e x_k.
Saída: $R_{\{x_i,x_j\}}$ atualizada, caminho-consistente em relação a x_k.

1: **para** cada $(a_i, a_j) \in R_{\{x_i,x_j\}}$ **faça**
2: **se** não existe $a_k \in D_k$ tal que $\{\langle x_i = a_i\rangle, \langle x_k = a_k\rangle\}$ satisfaz $R_{\{x_i,x_k\}}$ e $\{\langle x_k = a_k\rangle, \langle x_j = a_j\rangle\}$ satisfaz $R_{\{x_k,x_j\}}$ **então**
3: $R_{\{x_i,x_j\}} \leftarrow R_{\{x_i,x_j\}} \setminus \{(a_i, a_j)\}$
4: **fim se**
5: **fim para**

Mackworth (1977a) propôs o algoritmo `PC-3` para alcançar a consistência de caminho em uma rede de maneira semelhante ao algoritmo `AC-3`, utilizando uma pilha para processar apenas as restrições alteradas pelo procedimento `revisaCaminho`. Na literatura, outros algoritmos foram propostos, como `PC-4` (Han e Lee, 1988) e `PC-2001` (Zhang e Yap, 2001), usualmente estendendo algum método de consistência de arco existente.

3.2.3 k-consistência

A consistência de caminho é uma extensão da consistência de arco. Enquanto esta estende uma valoração consistente de uma única variável x_i a uma outra variável x_j, a consistência de caminho estende uma valoração consistente de duas variáveis $\{x_i, x_j\}$ a uma terceira variável x_k. Seguindo essa ideia, Freuder (1978) generalizou a noção de consistência para um conjunto arbitrário de k variáveis, chamada k-consistência.

Definição 3.5 (**k-consistência**).

- Uma rede $\mathcal{R}$ é dita k-consistente se, e somente se, para toda valoração consistente $\{\langle x_1 = a_1\rangle, \ldots, \langle x_{k-1} = a_{k-1}\rangle\}$ de quaisquer $k-1$ variáveis, existe um valor a_k no domínio de qualquer outra variável x_k tal que $\{\langle x_1 = a_1\rangle, \ldots, \langle x_{k-1} = a_{k-1}\rangle, \langle x_k = a_k\rangle\}$ é consistente com $\mathcal{R}$.
- Uma rede $\mathcal{R}$ é dita fortemente k-consistente se é j-consistente, para todo $1 \leq j \leq k$.

A k-consistência é uma poderosa ferramenta para classificar problemas com restrições, embora sua aplicação seja, em geral, impraticável. Alcançar k-consistência exige a adição de novas restrições de aridade $k-1$ na rede, mesmo que esta seja originalmente binária, ocasionando um custo de tempo e espaço exponenciais em k. Em contrapartida, em uma rede fortemente k-consistente é possível estender qualquer valoração parcial de $1 \leq j < k$ variáveis a k variáveis de forma consistente. Logo, em uma rede com n variáveis, n-consistência forte garante solução sem retrocesso. Por exemplo, se o problema das 8 rainhas fosse fortemente 8-consistente, então para qualquer valoração da primeira rainha existiria uma posição possível para a segunda rainha e, da mesma forma, existiria uma casa na terceira coluna para a terceira rainha, e assim por diante. Infelizmente, esse problema não é fortemente 8-consistente, mas é fácil ver, por exemplo, que é fortemente 3-consistente: dada uma rainha em qualquer linha da primeira coluna do tabuleiro sempre é possível colocar uma segunda rainha na segunda coluna, pois a primeira rainha está atacando, no máximo, 3 casas da segunda coluna; do mesmo modo, é sempre possível colocar uma rainha na terceira coluna, pois as duas primeiras rainhas atacam, no máximo, 6 casas da terceira coluna.

Teorema 3.3. *Se uma rede $\mathcal{R}$ com n variáveis é fortemente n-consistente, então $\mathcal{R}$ é globalmente consistente.*

Demonstração. Consequência direta das Definições 3.2 e 3.5. □

Pela Definição 3.5, a k-consistência pode ser aplicada em redes não binárias. O Algoritmo 6 descreve um método "força bruta" para alcançar essa consistência, seguindo a ideia original do algoritmo `AC-1`. Para todo conjunto de $k-1$ variáveis (linha 2) testa-se quais das valorações consistentes destas variáveis (linha 4) possuem uma extensão consistente em qualquer outra k-ésima variável da rede (linha 5). Lembra-se que uma valoração $\{\langle x_1 = a_1\rangle, \ldots, \langle x_k = a_k\rangle\}$ é consistente com $\mathcal{R}$ se, e somente se, satisfaz todas as restrições cujos escopos estão contidos em $\{x_1, \ldots, x_k\}$, isto é,

$$\forall R_{C_i} \subseteq D_{i_1} \times \cdots \times D_{i_l} : C_i \subseteq \{x_1, \ldots, x_k\} \Rightarrow (a_{i_1}, \ldots, a_{i_l}) \in R_{C_i} \tag{3.1}$$

Se uma valoração consistente não possui extensão para alguma variável x_k, essa tupla é removida da relação R_C, onde C é o conjunto de $k-1$ variáveis que está sendo testado. Como na

consistência de caminho, se uma restrição sobre um conjunto de $k-1$ variáveis quaisquer não existir, ela é interpretada inicialmente como a restrição universal, mas pode ser refinada pelo algoritmo e, então, passa a ser uma nova restrição na rede. Por isso, no pior caso, é adicionado um número exponencial de restrições de aridade $k-1$. Para redes binárias, 2-consistência é equivalente à consistência de arco, enquanto 3-consistência é equivalente à consistência de caminho.

Algoritmo 6 k-consistência força bruta (`k-consistencia`$(k, \mathcal{R})$)

Entrada: Parâmetro k e rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$.
Saída: C atualizado, tal que $\mathcal{R}$ é k-consistente.

repete
  para todo $C = \{x_1, \ldots, x_{k-1}\} \subset \mathcal{X}$ **faça**
    para todo $x_k \in \mathcal{X} \setminus C$ **faça**
      para cada $(a_1, \ldots, a_{k-1}) \in R_C$ **faça**
        se $\nexists a_k \in D_k$ tal que $\{\langle x_1 = a_1\rangle, \ldots, \langle x_{k-1} = a_{k-1}\rangle, \langle x_k = a_k\rangle\}$ é consistente com $\mathcal{R}$ **então**
          $R_C \leftarrow R_C \setminus \{(a_1, \ldots, a_{k-1})\}$
        fim se
      fim para
    fim para
  fim para
até que não haja alteração de restrições

Teorema 3.4. (Dechter, 2003). *Dada uma rede $\mathcal{R}$, a complexidade de tempo e espaço de* `k-consistencia`$(k, \mathcal{R})$ *é, respectivamente, $O(2^k(nd)^{2k})$ e $O(n^k d^k)$, onde n é o número de variáveis da rede e d é o tamanho do maior domínio de variável. Um limitante inferior de tempo e espaço para alcançar k-consistência é $\Omega(n^k d^k)$.*

Exemplo 3.3. Seja a rede $\mathcal{R}_{PC} = (\mathcal{X}, \mathcal{D}, C)$ que representa a instância de palavras-cruzadas da Figura 2.5, dada no Exemplo 2.2, tal que $\mathcal{X} = \{x_1, \ldots, x_{21}\}$, $D_i = \{A, \ldots, Z\}$, para todo $D_i \in \mathcal{D}$, e $C = \{R_{C_1}, \ldots, R_{C_5}\}$ tal que

$$C_1 = \{x_1, x_4, x_7, x_{10}, x_{15}\} \qquad C_4 = \{x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}\}$$
$$C_2 = \{x_2, x_6, x_8, x_{14}, x_{16}, x_{18}\} \qquad C_5 = \{x_{17}, x_{18}, x_{19}, x_{20}\}$$
$$C_3 = \{x_3, x_4, x_5\}$$

e

$$R_{C_1} = \{(\text{D, O, O, K, U}), (\text{L, A, N, D, O}), (\text{V, A, D, E, R}), (\text{W, I, N, D, U})\}$$
$$R_{C_2} = \{(\text{A, M, I, D, A, L, A}), (\text{S, I, D, I, O, U, S}), (\text{T, Y, R, A, N, U, S})\}$$
$$R_{C_3} = \{(\text{H, A, N}), (\text{J, A, R}), (\text{P, O, E})\}$$
$$R_{C_4} = \{(\text{A, N, A, K, I, N}), (\text{G, R, E, E, D, O}), (\text{K, E, N, O, B, I})\}$$
$$R_{C_5} = \{(\text{K, Y, L, O}), (\text{L, E, I, A}), (\text{L, U, K, E})\}$$

Essa rede é trivialmente 1-consistente e 2-consistente, pois não existem restrições unárias ou binárias. Por outro lado, ela não é 3-consistente: para $\{\langle x_3 = \text{A}\rangle, \langle x_4 = \text{A}\rangle\}$, por exemplo, não existe $a_5 \in D_5$ que satisfaça R_{C_3}. Esse conflito ocorre porque essa valoração parcial deveria ser detectada inconsistente, mas não há restrição binária entre x_3 e x_4. Ao executar `k-consistencia`$(3, \mathcal{R})$, a restrição $R_{\{x_3,x_4\}}$ é criada, bem como as restrições $R_{\{x_3,x_5\}}$ e $R_{\{x_4,x_5\}}$, permitindo que apenas pares consistentes dessas variáveis sejam valoradas:

$$R_{\{x_3,x_4\}} = \{(\text{H, A}), (\text{J, A}), (\text{P, O})\}$$
$$R_{\{x_3,x_5\}} = \{(\text{H, N}), (\text{J, R}), (\text{P, E})\}$$
$$R_{\{x_4,x_5\}} = \{(\text{A, N}), (\text{A, R}), (\text{O, E})\}$$

Nota-se que agora $\mathcal{R}_{PC}$ deixou de ser 2-consistente, pois a valoração $\langle x_3 = \text{A}\rangle$, embora consistente, não pode ser estendida à variável x_4 satisfazendo $R_{\{x_3,x_4\}}$. Uma nova aplicação de `k-consistencia`$(2, \mathcal{R})$ gera novas restrições unárias sobre estas variáveis:

$$R_{\{x_3\}} = \{\text{H, J, P}\}$$
$$R_{\{x_4\}} = \{\text{A, A, O}\}$$
$$R_{\{x_5\}} = \{\text{N, R, E}\}$$

Nesse caso, as restrições unárias são equivalentes a reduzir os domínios das variáveis aos conjuntos $D_3 = \{\text{H, J, P}\}$, $D_4 = \{\text{A, A, O}\}$ e $D_5 = \{\text{N, R, E}\}$. Finalmente, com essas alterações, a rede torna-se fortemente 3-consistente. △

3.2.4 Consistência de arco generalizada

Mackworth (1977b) introduziu o conceito de consistência de arco generalizada (também chamada de consistência de hiper-arco) como uma extensão da consistência de arco para restrições não binárias. Embora a k-consistência seja uma generalização das consistências de arco e de caminho, o seu conceito difere ligeiramente da ideia original da consistência de arco. Mais especificamente, na consistência de arco original, para cada valor no domínio de uma variável deve existir um valor correspondente no domínio da outra variável da restrição (binária) considerada. Por outro lado, na k-consistência, para toda valoração consistente de $k-1$ variáveis deve existir um valor correspondente no domínio de toda outra k-ésima variável satisfazendo todas as restrições (binárias ou não) sobre estas k variáveis. De forma mais semelhante à primeira, dada uma única restrição k-ária R_C, a consistência de arco generalizada verifica se, para cada valor no domínio de uma variável $x_j \in C$, existem valores nos domínios das demais variáveis em C que satisfazem a relação R_C. Por exemplo, a restrição $x_1 < x_2 + x_3$ não é arco-consistente generalizada sobre os domínios $D_1 = \{0, 1, 2\}$ e $D_2 = D_3 = \{0, 1\}$, pois para $\langle x_1 = 2\rangle$ não existem valores em D_2 e D_3 que satisfaçam a restrição.

Definição 3.6 (Consistência de arco generalizada (GAC)).

- Uma restrição R_C é arco-consistente generalizada (ou simplesmente GAC, do inglês *generalized arc-consistent*) em relação à variável $x_j \in C$ se, e somente se, para todo valor $a_j \in D_j$ existe uma valoração I das variáveis em $C \setminus \{x_j\}$ tal que $I \cup \{\langle x_j = a_j \rangle\}$ satisfaz R_C;
- Uma restrição R_C é GAC (total) se, e somente se, R_C é GAC em relação a toda variável $x_j \in C$;
- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é GAC se, e somente se, toda restrição $R_C \in C$ é GAC.

A consistência de arco generalizada pode ser alcançada apenas reduzindo domínios de variáveis, isto é, sem adicionar novas restrições à rede. Em particular, algoritmos para alcançar a consistência de arco original são naturalmente extensíveis à GAC. Os Algoritmos 7 e 8 descrevem uma extensão do algoritmo `AC-3` denominada `GAC-3`, proposta originalmente em Mackworth (1977b). Assim como na consistência de arco original, GAC não garante a consistência da rede. Por exemplo, a rede composta pelas restrições $x_1 = x_2 + x_3$ e $x_1 \neq x_2 + x_3$ é GAC sobre os domínios $D_1 = D_2 = D_3 = \{0, 1\}$, embora seja claramente inconsistente.

Algoritmo 7 Revisa restrição k-ária (`revisaGAC`$(R_C, (D_1, \ldots, D_k), x_j)$)

Entrada: Restrição R_C sobre as variáveis $C = \{x_1, \ldots, x_j, \ldots, x_k\}$ com domínios $D_1, \ldots, D_k$.
Saída: D_j atualizado, tal que R_C é GAC em relação a x_j.

1: **para** cada $a_j \in D_j$ **faça**
2: **se** $\nexists a_1 \in D_1, \ldots, a_{j-1} \in D_{j-1}, a_{j+1} \in D_{j+1}, \ldots, a_k \in D_k$ tais que $\{\langle x_1 = a_1 \rangle, \ldots, \langle x_k = a_k \rangle\}$ satisfaz R_C **então**
3: $D_j \leftarrow D_j \setminus \{a_j\}$
4: **fim se**
5: **fim para**

Teorema 3.5. `GAC-3`$(\mathcal{R})$ *consome tempo* $O(mk^2d^{k+1})$*, onde* m *é o número de restrições de* $\mathcal{R}$*,* d *é o tamanho do maior domínio na rede e* k *é a maior aridade de restrições em* $\mathcal{R}$*.*

Demonstração. Cada restrição é inserida na pilha k vezes, uma para cada variável de seu escopo. Além disso, cada vez que uma restrição é inserida pelo menos um valor foi removido do domínio de variáveis de seu escopo. Como há no máximo dk valores, cada restrição é processada, no pior caso, dk^2 vezes. Como há m restrições e o processamento de cada uma consiste em uma chamada do procedimento `revisaGAC`, então `revisaGAC` é executado mdk^2 vezes. Cada execução desse procedimento verifica $d \cdot d^{k-1}$ tuplas e, portanto, o algoritmo consome tempo de processamento $O(mk^2d^{k+1})$. □

Exemplo 3.4. Na rede $\mathcal{R}_{PC}$ da instância de palavras-cruzadas (Figura 2.5) a restrição R_{C_1} não é GAC, pois para $\langle x_1 = \mathrm{A} \rangle$, por exemplo, não existe valoração das outras variáveis que satisfaça

Algoritmo 8 Consistência de arco generalizada 3 (`GAC-3`($\mathcal{R}$))

Entrada: Rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$.
Saída: $\mathcal{D}$ atualizado, tal que $\mathcal{R}$ é GAC.

```
 1: para cada R_C ∈ C faça
 2:     para cada x_i ∈ C faça
 3:         empilha((R_C, x_i), pilha)
 4:     fim para
 5: fim para
 6: enquanto pilha ≠ ∅ faça
 7:     (R_C, x_i) ← desempilha(pilha)
 8:     revisaGAC(R_C, (D_1, ..., D_k), x_i)
 9:     se houve alteração em D_i então
10:         para cada R_C ∈ C tal que x_i ∈ C faça
11:             para cada x_j ∈ C \ {x_i} faça
12:                 empilha((R_C, x_j), pilha)
13:             fim para
14:         fim para
15:     fim se
16: fim enquanto
```

esta restrição, ou, em outras palavras, não existe tupla em R_{C_1} cujo primeiro elemento seja a letra "A". Aplicando-se o procedimento `revisaGAC`(R_{C_1}, (D_1, D_4, D_7, D_{10}, D_{15}), x_1) o domínio de x_1 é reduzido ao conjunto D_1 = {D, L, V, W}. Do mesmo modo, executando `revisaGAC`, para todo $x_i \in C_1$, obtém-se a seguinte redução de domínios:

$$D_1 \leftarrow \{\text{D, L,V, W}\}$$
$$D_4 \leftarrow \{\text{O, A, I}\}$$
$$D_7 \leftarrow \{\text{O, N, D}\}$$
$$D_{10} \leftarrow \{\text{K, D, E}\}$$
$$D_{15} \leftarrow \{\text{U, O, R}\}$$

Se essas execuções de `revisaGAC` ocorrerem dentro do algoritmo `GAC-3`, as restrições que contêm as variáveis cujo domínio foi alterado são inseridas na pilha (linhas 9 a 15); nesse caso, R_{C_3} e R_{C_4}. A próxima restrição da pilha é então processada por `revisaGAC` de maneira análoga.

A Tabela 3.2 descreve a execução completa de GAC−3($\mathcal{R}$). Ao final, os domínios das variáveis são reduzidos aos seguintes conjuntos:

$D_1 = \{\text{V}\}$	$D_8 = \{\text{D}\}$	$D_{15} = \{\text{R}\}$
$D_2 = \{\text{S}\}$	$D_9 = \{\text{K}\}$	$D_{16} = \{\text{O}\}$
$D_3 = \{\text{H, J}\}$	$D_{10} = \{\text{E}\}$	$D_{17} = \{\text{L}\}$
$D_4 = \{\text{A}\}$	$D_{11} = \{\text{N}\}$	$D_{18} = \{\text{U}\}$
$D_5 = \{\text{N, R}\}$	$D_{12} = \{\text{O}\}$	$D_{19} = \{\text{K}\}$
$D_6 = \{\text{I}\}$	$D_{13} = \{\text{B}\}$	$D_{20} = \{\text{E}\}$
$D_7 = \{\text{D}\}$	$D_{14} = \{\text{I}\}$	$D_{21} = \{\text{S}\}$

△

3.2.5 Consistência relacional

Dechter e van Beek (1997) propuseram o conceito de consistência relacional como uma abordagem alternativa às técnicas de consistência baseadas em variáveis, como k-consistência ou GAC. Nesta nova abordagem, inferências são ocasionadas pela análise de subconjuntos de restrições.

Exemplo 3.5. Seja $\mathcal{R}$ uma rede composta pelas restrições (cláusulas proposicionais) $R_{C_1} \equiv (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$ e $R_{C_2} \equiv (x_3 \vee x_5 \vee \neg x_6)$, com domínios $D_1 = \{\texttt{falso}\}$ e $D_i = \{\texttt{falso}, \texttt{verdadeiro}\}$, para todo $2 \leq i \leq 6$. Pela restrição R_{C_1} e pelo domínio D_1 pode-se inferir uma nova restrição $R_{C_3} = (\neg x_2 \vee \neg x_3 \vee x_4)$, pois x_1 só pode assumir o valor `falso` e, portanto, algum dos outros literais $\neg x_2$, $\neg x_3$ ou x_4 deve ser obrigatoriamente `verdadeiro`. Além disso, pelas restrições R_{C_1} e R_{C_2} pode-se inferir uma nova restrição $R_{C_4} \equiv (x_1 \vee \neg x_2 \vee x_4 \vee x_5 \vee \neg x_6)$, pois para qualquer valoração de x_3 o seu literal resulta em `falso` em alguma dessas cláusulas e, consequentemente, algum dos outros literais em R_{C_1} ou R_{C_2} deve ser `verdadeiro`.

Embora a primeira inferência, que gerou R_{C_3}, seja obtida alcançando-se 4-consistência, pois 4 variáveis são consideradas simultaneamente, do ponto de vista de conjuntos de restrições essa inferência assemelha-se à consistência de arco, pois apenas uma restrição é processada. Da mesma forma, a restrição R_{C_4} é obtida considerando-se 2 restrições, o que assemelha-se à consistência de caminho, alternativamente à interpretação baseada em variáveis que, nesse caso, exigiria a aplicação da 6-consistência. △

Definição 3.7 (Consistência de arco relacional (RAC)).

- Dada uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, uma restrição $R_C \in C$ é arco-consistente relacional (ou simplesmente RAC, do inglês *relational arc-consistent*) em relação à variável $x_j \in C$

iteração	restrição processada	domínios reduzidos
1	(R_{C_1}, x_1)	$D_1 \leftarrow$ {L, W, D, V}
	$(R_{C_1}, x_4)^*$	$D_4 \leftarrow$ {A, I, O}
	(R_{C_1}, x_7)	$D_7 \leftarrow$ {N, O, D}
	$(R_{C_1}, x_{10})^*$	$D_{10} \leftarrow$ {D, K, E}
	(R_{C_1}, x_{15})	$D_{15} \leftarrow$ {O, U, R}
2 (x_4)	(R_{C_3}, x_3)	$D_3 \leftarrow$ {H, J, P}
	(R_{C_3}, x_5)	$D_5 \leftarrow$ {N, R, E}
3 (x_{10})	(R_{C_4}, x_9)	$D_9 \leftarrow$ {K}
	(R_{C_4}, x_{11})	$D_{11} \leftarrow$ {N}
	(R_{C_4}, x_{12})	$D_{12} \leftarrow$ {O}
	(R_{C_4}, x_{13})	$D_{13} \leftarrow$ {B}
	$(R_{C_4}, x_{14})^*$	$D_{14} \leftarrow$ {I}
4 (x_{14})	(R_{C_2}, x_2)	$D_2 \leftarrow$ {S}
	(R_{C_2}, x_6)	$D_6 \leftarrow$ {I}
	(R_{C_2}, x_8)	$D_8 \leftarrow$ {D}
	(R_{C_2}, x_{16})	$D_{16} \leftarrow$ {O}
	$(R_{C_2}, x_{18})^*$	$D_{18} \leftarrow$ {U}
	(R_{C_2}, x_{21})	$D_{21} \leftarrow$ {S}
5 (x_{18})	(R_{C_5}, x_{17})	$D_{17} \leftarrow$ {L}
	(R_{C_5}, x_{19})	$D_{19} \leftarrow$ {K}
	(R_{C_5}, x_{20})	$D_{20} \leftarrow$ {E}
6	(R_{C_2}, x_2)	–
	(R_{C_2}, x_6)	–
	(R_{C_2}, x_8)	–
	(R_{C_2}, x_{14})	–
	(R_{C_2}, x_{16})	–
	(R_{C_2}, x_{18})	–
	(R_{C_2}, x_{21})	–
7	(R_{C_3}, x_3)	–
	$(R_{C_3}, x_4)^*$	$D_4 \leftarrow$ {A, O}
	(R_{C_3}, x_5)	–
8 (x_4)	(R_{C_1}, x_1)	$D_1 \leftarrow$ {L, D, V}
	(R_{C_1}, x_7)	–
	(R_{C_1}, x_{10})	–
	(R_{C_1}, x_{15})	–
9	(R_{C_4}, x_9)	–
	$(R_{C_4}, x_{10})^*$	$D_{10} \leftarrow$ {E}
	(R_{C_4}, x_{11})	–
	(R_{C_4}, x_{12})	–
	(R_{C_4}, x_{13})	–
	(R_{C_4}, x_{14})	–
10 (x_{10})	(R_{C_1}, x_1)	$D_1 \leftarrow$ {V}
	$(R_{C_1}, x_4)^*$	$D_4 \leftarrow$ {A}
	(R_{C_1}, x_7)	$D_7 \leftarrow$ {D}
	(R_{C_1}, x_{15})	$D_{15} \leftarrow$ {R}
11 (x_4)	(R_{C_3}, x_3)	$D_3 \leftarrow$ {H, J}
	(R_{C_3}, x_5)	$D_5 \leftarrow$ {N, R}
12	(R_{C_5}, x_{17})	–
	(R_{C_5}, x_{18})	–
	(R_{C_5}, x_{19})	–
	(R_{C_5}, x_{20})	–

Tabela 3.2: Execução do algoritmo `GAC-3` no Exemplo 3.4, supondo que no início a pilha é configurada da seguinte maneira: $pilha = (R_{C_1}, R_{C_2}, R_{C_3}, R_{C_4}, R_{C_5})$. O marcador $*$ indica os pares que causaram uma inserção na pilha devido à redução de domínios. A variável descrita abaixo do contador de iteração indica qual variável causou a inserção desta restrição na pilha e, consequentemente, não é processada nesta iteração. Iterações que não indicam essa variável estão processando restrições que foram inseridas na pilha no início do algoritmo.

se, e somente se, para toda valoração I das variáveis em $C \setminus \{x_j\}$, consistente com $\mathcal{R}$, existe um valor $a_j \in D_j$ tal que $I \cup \{\langle x_j = a_j \rangle\}$ satisfaz R_C;

- Uma restrição R_C é RAC (total) se, e somente se, R_C é RAC em relação a toda $x_j \in C$;
- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é RAC se, e somente se, toda restrição $R_C \in C$ é RAC.

Uma propriedade interessante de redes RAC é que se o hipergrafo de restrições é k-uniforme, então todo escopo C_i de restrições tem cardinalidade k e, consequentemente, qualquer valoração das variáveis em $C_i \setminus \{x_j\}$, para todo $x_j \in C_i$, é consistente (não há restrições de aridade $k-1$ a serem violadas). Nesse caso, RAC e GAC são consistências complementares.

Alcançar RAC exige a adição de novas restrições na rede. Se uma valoração parcial consistente $I = \{\langle x_1 = a_1 \rangle, \ldots, \langle x_{j-1} = a_{j-1} \rangle\}$ não é consistentemente estendida a $I \cup \{\langle x_j = a_j \rangle\}$, então a adição de uma restrição sobre $x_1, \ldots, x_{j-1}$ que proíbe a tupla $(a_1, \ldots, a_{j-1})$ evita que I ocorra. Uma vantagem dessa consistência, em oposição à k-consistência, é que não são criadas restrições sobre variáveis que não estão originalmente conectadas na rede. Por outro lado, o processo recursivo de alcançar RAC também nas restrições adicionadas pode gerar um número de novas restrições exponencial na aridade das restrições.

Exemplo 3.6. A rede GAC $\mathcal{R}_{PC}$, obtida ao se aplicar o algoritmo `GAC-3` no Exemplo 3.4, não é RAC. A saber, a valoração $\{\langle x_3 = \mathrm{H} \rangle, \langle x_5 = \mathrm{R} \rangle\}$ é consistente com $\mathcal{R}_{PC}$, mas não existe $a_4 \in D_4$ que satisfaça a restrição $R_{\{x_3,x_4,x_5\}} = \{(\mathrm{H}, \mathrm{A}, \mathrm{N}), (\mathrm{J}, \mathrm{A}, \mathrm{R}), (\mathrm{P}, \mathrm{O}, \mathrm{E})\}$. Nesse caso, $\mathcal{R}_{PC}$ torna-se RAC com a adição da restrição binária $R_{\{x_3,x_5\}} = \{(\mathrm{H}, \mathrm{N}), (\mathrm{J}, \mathrm{R})\}$. △

A consistência de arco relacional também pode ser estendida a conjuntos com k restrições, chamada k-consistência relacional. Em particular, 1-consistência relacional e consistência de arco relacional são definições equivalentes.

Definição 3.8 (k**-consistência relacional**)**.**

- Dada uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, um conjunto de restrições $\{R_{C_1}, \ldots, R_{C_k}\} \subseteq C$ é k-consistente relacional em relação à variável $x_j \in \bigcap_{i=1}^{k} C_i$ se, e somente se, para toda valoração I das variáveis em $\bigcup_{i=1}^{k} C_i \setminus \{x_j\}$, consistente com $\mathcal{R}$, existe um valor $a_j \in D_j$ tal que $I \cup \{\langle x_j = a_j \rangle\}$ satisfaz $\{R_{C_1}, \ldots, R_{C_k}\}$ simultaneamente;
- Um conjunto de restrições $\{R_{C_1}, \ldots, R_{C_k}\}$ é k-consistente relacional (total) se, e somente se, $\{R_{C_1}, \ldots, R_{C_k}\}$ é k-consistente relacional em relação a toda variável $x_j \in \bigcap_{i=1}^{k} C_i$;
- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é k-consistente relacional se, e somente se, todo conjunto de k restrições $\{R_{C_1}, \ldots, R_{C_k}\} \subseteq C$ é k-consistente relacional.

Um método "força bruta" para alcançar k-consistência relacional semelhante ao método para alcançar k-consistência é descrito pelo Algoritmo 9. Para todo conjunto de k restrições (linha 2), e para toda variável x_j compartilhada por todas as restrições desse conjunto (linha 3), testa-se quais das valorações consistentes das variáveis restantes (linhas 4 e 5) possuem uma extensão em x_j consistente com o conjunto de k restrições (linhas 6 e 7), removendo as tuplas inconsistentes da relação R_C, onde C é o conjunto de todas as variáveis que aparecem em alguma das k restrições excluindo-se x_j. Como na k-consistência original, se R_C não está na rede ela é interpretada inicialmente como a restrição universal, podendo ser refinada pelo algoritmo.

Algoritmo 9 k-consistência relacional força bruta (`k-consistencia-relacional`$(k, \mathcal{R})$)

Entrada: Parâmetro k e rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$.
Saída: C atualizado, tal que $\mathcal{R}$ é k-consistente relacional.

1: **repete**
2: **para** todo $\{R_{C_1}, \ldots, R_{C_k}\} \subseteq C$ **faça**
3: **para** todo $x_j \in \bigcap_{i=1}^{k} C_i$ **faça**
4: seja $C = \{x_1, \ldots, x_{j-1}\} = \left(\bigcup_{i=1}^{k} C_i\right) \setminus \{x_j\}$
5: **para** cada $(a_1, \ldots, a_{j-1}) \in R_C$ **faça**
6: **se** $\nexists a_j \in D_j$ tal que $\{\langle x_1 = a_1\rangle, \ldots, \langle x_j = a_j\rangle\}$ é consistente com $\{R_{C_1}, \ldots, R_{C_k}\}$ **então**
7: $R_C \leftarrow R_C \setminus \{(a_1, \ldots, a_{j-1})\}$
8: **fim se**
9: **fim para**
10: **fim para**
11: **fim para**
12: **até que** não haja alteração de restrições

Exemplo 3.7. Seja $\mathcal{R}$ a rede do Exemplo 3.5 sem incluir as restrições inferidas R_{C_3} e R_{C_4}. Esta rede não é 1-consistente relacional, pois a valoração $I = \{\langle x_2 = \texttt{verdadeiro}\rangle, \langle x_3 = \texttt{verdadeiro}\rangle, \langle x_4 = \texttt{falso}\rangle\}$ é consistente com $\mathcal{R}$ (não existe restrição cujo escopo esteja contido em $\{x_2, x_3, x_4\}$), mas não pode ser estendida a x_1 satisfazendo R_{C_1}. A restrição R_{C_1} torna-se 1-consistente relacional adicionando-se uma nova restrição sobre as variáveis em $C_1 \setminus \{x_1\}$ (linha 4 do Algoritmo 9), permitindo apenas valorações que são consistentemente estendidas a x_1 (linhas 6 e 7), que nesse caso é exatamente a restrição $R_{C_3} \equiv (\neg x_2 \vee \neg x_3 \vee x_4)$.

Além disso, $\mathcal{R}$ não é 2-consistente relacional, pois a valoração $I = \{\langle x_1 = \texttt{falso}\rangle, \langle x_2 = \texttt{verdadeiro}\rangle, \langle x_4 = \texttt{falso}\rangle, \langle x_5 = \texttt{falso}\rangle, \langle x_6 = \texttt{verdadeiro}\rangle\}$ não pode ser estendida a x_3 satisfazendo R_{C_1} e R_{C_2} simultaneamente. O conjunto $\{R_{C_1}, R_{C_2}\}$ torna-se 2-consistente relacional adicionando-se a restrição $R_{C_4} \equiv (x_1 \vee \neg x_2 \vee x_4 \vee x_5 \vee \neg x_6)$. Da mesma forma, $\{R_{C_2}, R_{C_3}\}$ não é 2-consistente relacional, inferindo uma nova restrição $R_{C_5} \equiv (\neg x_2 \vee x_4 \vee x_5 \vee \neg x_6)$. Finalmente, todo par de restrições em $\mathcal{R}$ é 2-consistente relacional e, portanto, $\mathcal{R}$ é fortemente 2-consistente relacional.

É fácil ver que, na rede resultante, qualquer valoração parcial consistente pode ser estendida a uma valoração total da rede; portanto, $\mathcal{R}$ é globalmente consistente. De modo geral, 2-consistência relacional forte é suficiente para tornar qualquer instância do problema

da satisfazibilidade proposicional globalmente consistente (Dechter e Rish, 1994), embora um número exponencial de novas cláusulas seja adicionado à rede. △

Em redes binárias, k-consistência relacional é idêntica a $k + 1$ consistência. Em particular, no caso binário RAC e GAC são equivalentes. Devido ao crescimento exponencial da rede, em geral, k-consistência relacional é impraticável para $k > 2$. Um algoritmo para alcançar maiores níveis de consistência relacional, considerando uma versão relaxada desta consistência, é proposto em Karakashian et al. (2010).

3.2.6 Consistência por emparelhamento e hiper-k-consistência

Outras consistências baseadas em restrições propostas na literatura são a consistência por emparelhamento (Janssen et al., 1989), baseada em trabalhos da comunidade de banco de dados relacionais (Beeri et al., 1983), e a hiper-k-consistência, proposta por Jégou (1993) como uma generalização da primeira.

Definição 3.9 (**consistência por emparelhamento**)**.**

- Uma rede $\mathcal{R}$ é dita consistente por emparelhamento se, e somente se, para toda valoração consistente $\{\langle x_1 = a_1\rangle, \ldots, \langle x_k = a_k\rangle\}$ de todas as variáveis participando em qualquer restrição R_{C_i}, existe uma extensão às variáveis em $C_j \setminus C_i$, para qualquer outra restrição R_{C_j}, satisfazendo R_{C_i} e R_{C_j} simultaneamente.

Consistência por emparelhamento é equivalente à consistência de arco no grafo dual da rede de restrições. Com base nessa observação, Janssen et al. (1989) propôs um algoritmo polinomial para alcançar esta consistência.

Estendendo a consistência por emparelhamento, Jégou (1993) propôs a hiper-k-consistência como uma contraparte da k-consistência no grafo dual da rede de restrições. Intuitivamente, ao invés de garantir a valoração de qualquer k-ésima variável, dada qualquer valoração consistente de outras $k-1$ variáveis, considera-se a existência de uma tupla em qualquer k-ésima restrição que estende a "valoração" de outras $k - 1$ restrições (isto é, a valoração de todas as variáveis nessas $k - 1$ restrições). Em particular, hiper-2-consistência e consistência por emparelhamento são conceitos equivalentes.

Definição 3.10 (**hiper-k-consistência**)**.**

- Uma rede $\mathcal{R}$ é dita hiper-k-consistente se, e somente se, para toda valoração consistente $\{\langle x_1 = a_1\rangle, \ldots, \langle x_l = a_l\rangle\}$ de todas as variáveis participando em quaisquer $k - 1$ restrições (isto é, tal que $\{x_1, \ldots, x_l\}$ é a união dos escopos de todas as $k - 1$ restrições), existe uma extensão às variáveis em $C_k \setminus \{x_1, \ldots, x_l\}$, para qualquer outra k-ésima restrição R_{C_k}, satisfazendo as k restrições simultaneamente.

- Uma rede $\mathcal{R}$ é dita fortemente hiper-k-consistente se é hiper-j-consistente, para todo $1 \leq j \leq k$.

Equivalentemente, uma rede de restrições $\mathcal{R}$ é hiper-k-consistente se, e somente se, a rede $\mathcal{R}^{dual}$ é k-consistente.

Exemplo 3.8. Seja uma rede $\mathcal{R}$ composta pelas variáveis $x_1, \ldots, x_4$ com domínios $D_i = \{0, 1\}$, para todo $1 \leq i \leq 4$, e as restrições

$$R_{\{x_1,x_2,x_3\}} = \{(0,0,0), (0,1,0), (1,1,0)\}$$
$$R_{\{x_2,x_3,x_4\}} = \{(0,0,1), (1,0,1), (1,1,0)\}$$
$$R_{\{x_1,x_2,x_4\}} = \{(0,0,1), (0,1,1), (1,1,1)\}$$

Esta rede é hiper-3-consistente, pois para toda valoração consistente de variáveis participando em duas restrições quaisquer, existe uma extensão consistente satisfazendo também a terceira restrição. A saber, para todas as valorações consistentes das variáveis em $R_{\{x_1,x_2,x_3\}}$ e $R_{\{x_2,x_3,x_4\}}$ existe uma tupla em $R_{\{x_1,x_2,x_4\}}$:

$$\{\langle x_1 = 0\rangle, \langle x_2 = 0\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (0,0,1) \in R_{\{x_1,x_2,x_4\}}$$
$$\{\langle x_1 = 0\rangle, \langle x_2 = 1\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (0,1,1) \in R_{\{x_1,x_2,x_4\}}$$
$$\{\langle x_1 = 1\rangle, \langle x_2 = 1\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (1,1,1) \in R_{\{x_1,x_2,x_4\}}$$

Do mesmo modo, para toda valoração consistente com $R_{\{x_1,x_2,x_3\}}$ e $R_{\{x_1,x_2,x_4\}}$ existe uma tupla em $R_{\{x_2,x_3,x_4\}}$:

$$\{\langle x_1 = 0\rangle, \langle x_2 = 0\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (0,0,1) \in R_{\{x_2,x_3,x_4\}}$$
$$\{\langle x_1 = 0\rangle, \langle x_2 = 1\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (1,0,1) \in R_{\{x_2,x_3,x_4\}}$$
$$\{\langle x_1 = 1\rangle, \langle x_2 = 1\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (1,0,1) \in R_{\{x_2,x_3,x_4\}}$$

E para toda valoração consistente com $R_{\{x_2,x_3,x_4\}}$ e $R_{\{x_1,x_2,x_4\}}$ existe uma tupla em $R_{\{x_1,x_2,x_3\}}$:

$$\{\langle x_1 = 0\rangle, \langle x_2 = 0\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (0,0,0) \in R_{\{x_1,x_2,x_3\}}$$
$$\{\langle x_1 = 0\rangle, \langle x_2 = 1\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (0,1,0) \in R_{\{x_1,x_2,x_3\}}$$
$$\{\langle x_1 = 1\rangle, \langle x_2 = 1\rangle, \langle x_3 = 0\rangle, \langle x_4 = 1\rangle\} \Rightarrow (1,1,0) \in R_{\{x_1,x_2,x_3\}}$$

△

3.3 Técnicas de consistência para CSPs numéricos

Um grande número de problemas do mundo real envolvem variáveis e restrições numéricas, representando valores e informações que podem não ser naturalmente discretizáveis. Tais problemas podem ser representados por CSPs numéricos, nos quais uma instância é descrita através de expressões que definem os domínios e as restrições de forma implícita, em oposição à descrição explícita utilizada em instâncias de CSPs finitos.

Mais especificamente, no cenário contínuo, domínios são representados por multi-intervalos e restrições por (in)equações numéricas. Assim, o tamanho de uma instância pode ser expresso em função de 4 variáveis:

- n: número de variáveis na rede;
- m: número de restrições na rede;
- l: maior número de intervalos que compõem algum multi-intervalo (domínio) na rede;
- s: maior número de símbolos necessários para representar a expressão de uma restrição numérica.

Em geral, restrições são compostas por funções fatoráveis e o parâmetro s pode ser substituído pelo número de operações binárias e unárias da restrição.

As técnicas de consistência apresentadas na Seção 3.2 não são naturalmente extensíveis ao cenário contínuo. Em particular, observando os algoritmos para alcançar as consistências de arco (generalizada), de caminho, k-consistência e k-consistência relacional, nota-se que estes baseiam-se em percorrer todo o domínio das variáveis ou tuplas da relação, removendo valores localmente inconsistentes. Esse processo pode não ser trivial se estes conjuntos forem definidos de forma implícita por alguma expressão computável. Uma forma de adaptar os métodos de consistência que reduzem domínios de variáveis, como a consistência de arco generalizada, por exemplo, é utilizar os conceitos de função de projeção e contrator.

Definição 3.11 (**Função de projeção**)**.** Dada uma restrição numérica R_C da forma $R_C \equiv f(x_1, \ldots, x_k) = 0$, sobre as variáveis $x_1, \ldots, x_k$ com domínios $D_1, \ldots, D_k$, uma função de projeção $f_{[x_j]} : D_1 \times \cdots \times D_{j-1} \times D_{j+1} \times \cdots \times D_k \mapsto D_j$ de R_C em x_j, para algum $x_j \in C$, é uma função que satisfaz a seguinte propriedade: $f_{[x_j]}(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k) = x_j \Leftrightarrow f(x_1, \ldots, x_k) = 0$.

Na prática, uma função de projeção de R_C em x_j é obtida isolando-se essa variável na expressão de R_C.

Definição 3.12 (**Contrator**)**.** Dada uma restrição numérica R_C da forma $R_C \equiv f_{[x_j]}(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k) = x_j$, sobre as variáveis $x_1, \ldots, x_k$ com domínios $D_1, \ldots, D_k$, define-se o operador contrator pela seguinte operação:

$$D_j \leftarrow D_j \cap f_{[x_j]}[B_j] \tag{3.2}$$

onde $f_{[x_j]}[B_j]$ denota o conjunto imagem da função $f_{[x_j]}$ em $B_j = D_1 \times \cdots \times D_{j-1} \times D_{j+1} \times \cdots \times D_k$.

O contrator é uma operação atômica que remove do domínio de x_j os valores que são inconsistentes com a restrição R_C, pois se existe um valor $a_j \in D_j$ tal que $a_j \notin f_{[x_j]}[B_j]$, então não existe uma valoração $\{\langle x_1 = a_1\rangle, \ldots, \langle x_j = a_j\rangle, \ldots, \langle x_k = a_k\rangle\}$ que satisfaça R_C.

Exemplo 3.9. Seja a restrição $R_C \equiv {x_1}^2 - x_2 = 0$ sobre as variáveis x_1 e x_2 com domínios $D_1 = D_2 = (-\infty, +\infty)$. As funções de projeção de R_C em x_1 e x_2 são dadas, respectivamente, por $f_{[x_1]}(x_2) = \pm\sqrt{x_2}$ e $f_{[x_2]}(x_1) = {x_1}^2$. Aplicando o contrator em $f_{[x_1]}$ e $f_{[x_2]}$ obtém-se as seguintes reduções:

$$D_1 \leftarrow D_1 \cap f_{[x_1]}[D_2] = (-\infty, +\infty) \cap \underbrace{\pm\sqrt{(-\infty, +\infty)}}_{(-\infty, +\infty)}$$

$$D_2 \leftarrow D_2 \cap f_{[x_2]}[D_1] = (-\infty, +\infty) \cap \underbrace{(-\infty, +\infty)^2}_{[0, +\infty)}$$

E, portanto, os valores inconsistentes com R_C ($a_2 < 0$) são removidos de D_2. △

O procedimento `revisaGAC` (Algoritmo 7), que alcança a consistência de arco generalizada de uma restrição R_C, em relação a uma variável $x_j \in C$, pode ser escrito utilizando os conceitos de função de projeção e contrator, conforme descrito pelo Algoritmo 10. Esse procedimento faz uso de uma função `imagem_da_projecao`(R_C, x_j) que devolve o conjunto imagem da função de projeção de R_C em x_j (para restrições numéricas, essa função é equivalente à Definição 3.11).

Algoritmo 10 Revisa restrição k-ária com contrator (`contratorGAC`$(R_C, (D_1, \ldots, D_k), x_j)$)

Entrada: Restrição R_C sobre as variáveis $C = \{x_1, \ldots, x_j, \ldots, x_k\}$ com domínios $D_1, \ldots, D_k$.
Saída: D_j atualizado, de forma que R_C é GAC em relação a x_j.

1: $D_j \leftarrow D_j \cap$ `imagem_da_projecao`(R_C, x_j)
   // se R_C é uma restrição numérica da forma $f(x_1, \ldots, x_k) = 0$,
   // esse procedimento computa $f_{[x_j]}[D_1, \ldots, D_{j-1}, D_{j+1}, \ldots, D_k]$

Como apresentando na Seção 2.3.3, a extensão intervalar é capaz de computar uma estimativa do conjunto imagem de funções reais apenas observando os extremos dos intervalos operados. Mais especificamente, o conjunto computado é exatamente o conjunto imagem se a função não apresentar múltiplas ocorrências da mesma variável (Benhamou e Granvilliers, 2006). Além disso, é possível decompor eficientemente qualquer função fatorável em um conjunto de restrições numéricas ternárias, nas quais cada variável ocorre uma única vez e, consequentemente, computar o conjunto imagem de suas funções de projeção torna-se um processo eficiente.

3.3.1 Consistências de arco para redes numéricas ternárias

As funções de projeção de uma restrição ternária $R_C \equiv x_1 = x_2 \circ x_3$, onde $x_2 \circ x_3$ é uma expressão elementar tal que $\circ \in \{+, -, \cdot, /, \hat{}, \sqrt{}\}$, são trivialmente obtidas isolando-se cada

$\circ$	$\bullet$	$\diamond$	descrição
$+$	$-$	$-$	
$-$	$+$	$\ominus$	$x_2 \ominus x_1 \equiv x_1 - x_2$
$\cdot$	$/$	$/$	
$/$	$\cdot$	$\oslash$	$x_2 \oslash x_1 \equiv x_1 \,/\, x_2$
$\hat{}$	$\sqrt{}$	log	
$\sqrt{}$	$\hat{}$	$\wr$	$x_2 \wr x_1 \equiv \log_{x_2} x_1$

Tabela 3.3: Operadores inversos dos principais operadores algébricos.

uma das variáveis[1] x_1, x_2 e x_3. Para tanto, sejam $\bullet$ e $\diamond$ os dois inversos do operador $\circ$, isto é, que mantêm, quando definidos em $\mathbb{R}$, a seguinte equivalência :

$$(x_1 = x_2 \circ x_3) \Leftrightarrow (x_2 = x_1 \bullet x_3) \Leftrightarrow (x_3 = x_1 \diamond x_2) \tag{3.3}$$

A Tabela 3.3 apresenta os inversos dos principais operadores algébricos.

Com essa padronização, pode-se simplificar as definições das consistências de arco generalizada e relacional a restrições ternárias, conforme enuncia os Teoremas 3.6 e 3.7.

Teorema 3.6. *Uma restrição $R_C \equiv x_1 = x_2 \circ x_3$ sobre as variáveis x_1, x_2 e x_3 com domínios D_1, D_2 e D_3 é GAC se, e somente se,*

$$(D_1 \subseteq D_2 \circ D_3) \quad e \quad (D_2 \subseteq D_1 \bullet D_3) \quad e \quad (D_3 \subseteq D_1 \diamond D_2)$$

Demonstração. Pela equivalência (3.3), esta propriedade garante que para todo valor em um domínio D_i existem valores nos demais domínios que satisfazem R_C. □

Teorema 3.7. *Em uma rede puramente ternária (isto é, sem restrições binárias), uma restrição $R_C \equiv x_1 = x_2 \circ x_3$ sobre as variáveis x_1, x_2 e x_3 com domínios D_1, D_2 e D_3 é RAC se, e somente se,*

$$(D_1 \supseteq D_2 \circ D_3) \quad e \quad (D_2 \supseteq D_1 \bullet D_3) \quad e \quad (D_3 \supseteq D_1 \diamond D_2)$$

Demonstração. Pela equivalência (3.3), esta propriedade garante que para todo par de valores em dois domínio D_i e D_j existe um valor no domínio restante que satisfaz R_C. □

Nota-se que estes resultados são estendidos de maneira análoga a restrições binárias da forma $R_C \equiv x_1 = g(x_2)$, se g for um operador unário que admite inverso g^{-1}. Mais especificamente, se a equivalência $(x_1 = g(x_2)) \Leftrightarrow (x_2 = g^{-1}(x_1))$ é assegurada, então R_C é GAC se, e somente se, $D_1 \subseteq G(D_2)$ e $D_2 \subseteq G^{-1}(D_1)$, e é RAC se, e somente se, $D_1 \supseteq G(D_2)$ e $D_2 \supseteq G^{-1}(D_1)$, onde G e G^{-1} são, respectivamente, as extensões intervalares naturais dos operadores g e g^{-1}.

[1] Um variável x_i também pode representar uma constante a_i definindo-se $D_i = \{a_i\}$.

Algoritmos para alcançar GAC e RAC podem ser escritos de forma especializada para restrições numéricas ternárias, conforme descrito pelos Algoritmos 11 e 12. Considerando toda a rede de restrições, estes procedimentos são utilizados para processar cada restrição separadamente, seja por meio de um método "força bruta", como o `k-consistencia-relacional` (Algoritmo 9), ou utilizando uma pilha de restrições processadas, como o `GAC-3` (Algoritmo 8).

Algoritmo 11 Contrator GAC ternário (`contratorGAC_ternario`(R_C, D_1, D_2, D_3))

Entrada: Restrição $R_C \equiv x_1 = x_2 \circ x_3$ sobre as variáveis $C = \{x_1, x_2, x_3\}$ com domínios D_1, D_2, D_3.
Saída: D_1, D_2 e D_3 atualizados, tais que R_C é GAC.

1: $D_1 \leftarrow D_1 \cap (D_2 \circ D_3)$
2: $D_2 \leftarrow D_2 \cap (D_1 \bullet D_3)$
3: $D_3 \leftarrow D_3 \cap (D_1 \diamond D_2)$

Algoritmo 12 Contrator RAC ternário (`contratorRAC_ternario`(R_C, D_1, D_2, D_3))

Entrada: Restrição $R_C \equiv x_1 = x_2 \circ x_3$ sobre as variáveis $C = \{x_1, x_2, x_3\}$ com domínios D_1, D_2, D_3.
Saída: Conjunto de restrições $\{R_C, R_{\{x_2,x_3\}}, R_{\{x_1,x_3\}}, R_{\{x_1,x_2\}}\}$ sob o qual R_C é RAC.

1: seja $R_{\{x_2,x_3\}} \equiv x_2 \circ x_3 \in D_1$
2: seja $R_{\{x_1,x_3\}} \equiv x_1 \bullet x_3 \in D_2$
3: seja $R_{\{x_1,x_2\}} \equiv x_1 \diamond x_2 \in D_3$
4: **devolve** $\{R_C, R_{\{x_2,x_3\}}, R_{\{x_1,x_3\}}, R_{\{x_1,x_2\}}\}$

Teorema 3.8. *Dada uma restrição ternária $R_C \equiv x_1 = x_2 \circ x_3$, onde $\circ \in \{+, -, \cdot, /, \hat{}, \sqrt{}\}$ e, se $\circ \in \{\hat{}, \sqrt{}\}$, então D_3 contém um único valor inteiro,* `contratorGAC_ternario`(R_C, D_1, D_2, D_3) *consome tempo $O(l^5 \cdot \log l)$, onde l é o maior número de intervalos que compõem algum dos multi-intervalos D_1, D_2 ou D_3.*

Demonstração. Cada operação multi-intervalar deve operar os intervalos que compõem seus operandos dois a dois. Pelas definições das operações básicas apresentadas na Seção 2.3.2, as operações intervalares $+$, $-$, $\cdot$, $/$, $\hat{}$ e $\sqrt{}$ são computadas em tempo constante. Na operação da linha 1, efetua-se l^2 operações intervalares e consome-se tempo $O(l^2 \cdot \log l)$ para tornar o multi-intervalo resultante ordenado. A interseção de multi-intervalos ordenados é linear no seu número de intervalos. As operações das linhas 2 e 3 ocorrem de maneira análoga, exceto pelo fato de que, no pior caso, na linha 2 o multi-intervalo D_1 é composto por l^2 intervalos, e portanto consome-se tempo $O(l^3 \cdot \log l)$, e, na linha 3, D_1 e D_2 podem ter até l^2 e l^3 intervalos, respectivamente, consumindo tempo $O(l^5 \cdot \log l)$. □

Exemplo 3.10. Seja $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ uma rede de restrições numéricas tal que $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$ e $C = \{R_{C_1}, R_{C_2}\}$, onde $C_1 = \{x_1, x_2, x_3\}$, $C_2 = \{x_1, x_2, x_4\}$ e

$$D_1 = [1, 5] \qquad D_2 = [-1, 1]$$
$$D_3 = [1, 3] \qquad D_4 = [1, 2]$$
$$R_{C_1} \equiv x_1 = x_2 + x_3 \qquad R_{C_2} \equiv x_4 = x_1 \cdot x_2$$

A restrição R_{C_1} não é GAC, pois $D_1 \not\subseteq (\underbrace{D_2 + D_3}_{[0,4]})$.

Executando `contratorGAC_ternario`(R_{C_1}, D_1, D_2, D_3), obtém-se:

$$D_1 \leftarrow D_1 \cap (D_2 + D_3) = [1,5] \cap (\underbrace{[-1,1] + [1,3]}_{[0,4]}) = [1,4]$$
$$D_2 \leftarrow D_2 \cap (D_1 - D_3) = [-1,1] \cap (\underbrace{[1,4] - [1,3]}_{[-2,3]}) = [-1,1]$$
$$D_3 \leftarrow D_3 \cap (D_1 - D_2) = [1,3] \cap (\underbrace{[1,4] - [-1,1]}_{[0,5]}) = [1,3]$$

Nesse caso, a restrição R_{C_2} também não é GAC, pois $D_2 \not\subseteq (\underbrace{D_4/D_1}_{[\frac{1}{4},2]})$.

De forma análoga, `contratorGAC_ternario`(R_{C_2}, D_1, D_2, D_4) computa:

$$D_4 \leftarrow D_4 \cap (D_1 \cdot D_2) = [1,2] \cap (\underbrace{[1,4] \cdot [-1,1]}_{[-4,4]}) = [1,2]$$
$$D_1 \leftarrow D_1 \cap (D_4/D_2) = [1,4] \cap (\underbrace{[1,2]/[-1,1]}_{\langle(-\infty,-1],[1,+\infty)\rangle}) = [1,4]$$
$$D_2 \leftarrow D_2 \cap (D_4/D_1) = [-1,1] \cap (\underbrace{[1,2]/[1,4]}_{[\frac{1}{4},2]}) = [\frac{1}{4},1]$$

Embora R_{C_2} torne-se GAC com a redução de D_2, essa modificação altera a consistência da primeira restrição, sendo necessário uma nova aplicação do contrator:

$$D_1 \leftarrow D_1 \cap (D_2 + D_3) = [1,4] \cap (\underbrace{[\frac{1}{4},1] + [1,3]}_{[\frac{5}{4},4]}) = [\frac{5}{4},4]$$
$$D_2 \leftarrow D_2 \cap (D_1 - D_3) = [\frac{1}{4},1] \cap (\underbrace{[\frac{5}{4},4] - [1,3]}_{[-\frac{7}{4},\ 3]}) = [\frac{1}{4},1]$$
$$D_3 \leftarrow D_3 \cap (D_1 - D_2) = [1,3] \cap (\underbrace{[\frac{5}{4},4] - [\frac{1}{4},1]}_{[\frac{1}{4},\frac{15}{4}]}) = [1,3]$$

Com esses novos domínios, ambas as restrições tornam-se GAC. No entanto, elas não são RAC; por exemplo, para $\{\langle x_1 = 4\rangle, \langle x_2 = 0{,}25\rangle\}$, não existe $a_3 \in D_3$ tal que $a_3 = 4 - 0{,}25 = 3{,}75$; ou ainda, para $\{\langle x_1 = 4\rangle, \langle x_2 = 1\rangle\}$, não existe $a_4 \in D_4$ tal que $a_4 = 4 \cdot 1 = 4$. Considerando a definição de consistência de arco relacional estabelecida pelo Teorema 3.7, as restrições não são RAC porque $D_3 \not\supseteq (D_1 - D_2)$ e $D_4 \not\supseteq (D_1 \cdot D_2)$, e não há restrições binárias na rede proibindo alguma valoração parcial de x_1 e x_2. A consistência de arco

relacional de $\mathcal{R}$ é alcançada, nesse caso, adicionando-se novas restrições binárias à rede, conforme executado pelos procedimentos `contratorRAC_ternario`(R_{C_1}, D_1, D_2, D_3) e `contratorRAC_ternario`(R_{C_2}, D_1, D_2, D_4):

$$R_{C_3} \equiv x_2 + x_3 \in [\frac{5}{4}, 4] \qquad R_{C_4} \equiv x_1 - x_3 \in [\frac{1}{4}, 1]$$
$$R_{C_5} \equiv x_1 - x_2 \in [1, 3] \qquad R_{C_6} \equiv x_1 \cdot x_2 \in [1, 2]$$
$$R_{C_7} \equiv x_3/x_2 \in [\frac{5}{4}, 4] \qquad R_{C_8} \equiv x_3/x_1 \in [\frac{1}{4}, 1]$$

Vale lembrar que nem GAC e nem RAC garantem, no caso geral, consistência global. A valoração $\langle x_1 = 4\rangle$, por exemplo, é consistente com $\mathcal{R}$, mas não pode ser consistentemente estendida a x_2 (por R_{C_5} infere-se $x_2 \in [1, 3]$, enquanto, por R_{C_6}, infere-se $x_2 \in [\frac{1}{4}, \frac{1}{2}]$). △

Davis (1987) mostra que alcançar GAC em uma rede de restrições numéricas demanda um número possivelmente infinito de iterações. Por exemplo, a rede composta pelas restrições $x_1 = x_2$ e $x_1 = x_2/2$ é GAC apenas com os domínios $D_1 = D_2 = [0, 0]$; no entanto, a partir de quaisquer domínios $D_1 \ni 0$ e $D_2 \ni 0$ arbitrários, a aplicação do contrator bissecta D_1 infinitamente. De forma alternativa, pode-se alcançar uma aproximação de GAC estipulando uma precisão mínima aos multi-intervalos D_1 e D_2, ou um número máximo de aplicações de contratores. Embora essas estratégias garantam que o método termina e que os domínios são reduzidos de forma correta, isto é, sem excluir soluções, a rede resultante não é GAC: a partir da valoração de uma variável qualquer de uma restrição, não há garantia da existência de extensão consistente a todas as outras variáveis do escopo dessa restrição. Faltings (1998) avançou os resultados de Davis mostrando que em redes acíclicas GAC é computável.

Outro problema em relação a GAC no cenário contínuo é que os extremos de intervalos que compõem um domínio podem não ser representados por números de máquina, como ocorre na restrição $x_1 = {x_2}^2$, para $D_1 = [1, 2]$, que infere, através do contrator GAC, o domínio $D_2 = \langle[-\sqrt{2}, -1], [1, \sqrt{2}]\rangle$. Além disso, no pior caso, alcançar GAC de forma exata demanda espaço exponencial no tamanho da instância, pois intervalos podem ser particionados indefinidamente (Hyvönen, 1992; Faltings, 1998; Chabert et al., 2004). No entanto, a estrutura da rede e a forma de representação de multi-intervalos são fatores que podem amenizar esse custo na maioria dos casos (Sidebottom e Havens, 1992; Batnini et al., 2005). Em geral, não há consenso na literatura sobre a eficiência de manter ou não a representação de domínios por multi-intervalos.

Exemplo 3.11. (Chabert et al., 2004). Seja $\mathcal{R}$ uma rede composta pelas restrições $R_{C_1} \equiv x_1 = x_2$ e $R_{C_2} \equiv x_2 = (\frac{3}{4} \cdot (x_1 - 5))^2$ e domínios $D_1 = D_2 = [1, 9]$. Enquanto R_{C_1} é consistente, R_{C_2} não é GAC em relação a x_1. Aplicando-se o contrator na função de projeção de R_{C_2} em x_1, dada por $f_{[x_1]}(x_2) = \pm\sqrt{x_2} \cdot \frac{4}{3} + 5$, obtém-se

$$
\begin{aligned}
D_1 \leftarrow & D_1 \cap \pm\sqrt{D_2} \cdot \frac{4}{3} + 5 \\
&= [1,9] \cap \sqrt{[1,9]} \cdot \frac{4}{3} + 5 \\
&= [1,9] \cap \langle [-3,-1], [1,3] \rangle \cdot \frac{4}{3} + 5 \\
&= [1,9] \cap \left\langle [1, \frac{11}{3}], [\frac{19}{3}, 9] \right\rangle \\
&= \left\langle [1, \frac{11}{3}], [\frac{19}{3}, 9] \right\rangle
\end{aligned}
$$

Com a alteração de D_1, o contrator deve também ser aplicado em R_{C_1}, obtendo:

$$
\begin{aligned}
D_2 \leftarrow & D_2 \cap D_1 \\
&= [1,9] \cap \left\langle [1, \frac{11}{3}], [\frac{19}{3}, 9] \right\rangle \\
&= \left\langle [1, \frac{11}{3}], [\frac{19}{3}, 9] \right\rangle
\end{aligned}
$$

Com a atualização de D_2, a primeira restrição não é mais GAC. Uma nova aplicação do contrator divide cada intervalo do multi-intervalo D_1 em um intervalo distinto:

$$
D_1 \leftarrow \left\langle [1, 5 - \frac{4}{3}\sqrt{\frac{19}{3}}], [5 - \frac{4}{3}\sqrt{\frac{11}{3}}, \frac{11}{3}], [\frac{19}{3}, 5 + \frac{4}{3}\sqrt{\frac{11}{3}}], [5 + \frac{4}{3}\sqrt{\frac{19}{3}}, 9] \right\rangle
$$

Esse processo repete-se infinitamente, computando, em cada iteração, um multi-intervalo com o dobro de intervalos da iteração anterior. Cada intervalo, por sua vez, não é representável por números de máquina de forma exata. △

3.3.2 Consistência de Envoltória

O conceito de consistência de envoltória[2] foi introduzido em Benhamou e Older (1992, 1997) como uma relaxação da consistência de arco visando contornar o problema da representação numérica de extremos de intervalos, bem como evitar o crescimento exponencial da representação multi-intervalar. Informalmente, cada domínio é reduzido ao menor intervalo representável que contém o conjunto de valores arco-consistentes da rede. Embora não garanta a extensão consistente de uma valoração de maneira tão forte quanto a consistência de arco, a consistência de envoltória é considerada uma técnica de inferência completa, isto é, não exclui soluções da rede. Sua aplicação é eficiente e vastamente utilizada em métodos intervalares.

[2]Do inglês *hull consistency*. Também pode ser traduzido para consistência de fecho (convexo).

Definição 3.13 (**Envoltória**)**.** A envoltória de um conjunto de números reais $X \subseteq \mathbb{R}$ é o intervalo $\blacklozenge X$ de menor comprimento que contém X, isto é, $\blacklozenge X = [\min X, \max X]$.

Considerando sistemas de representação computacionais, nos quais apenas um conjunto finito de números são representados, define-se a envoltória de um conjunto de números X como o intervalo $\blacklozenge X = [a, b]$, onde a é o maior número de máquina menor ou igual a $\min X$ e b é o menor número de máquina maior ou igual a $\max X$.

Definição 3.14 (**Consistência de envoltória**)**.**

- Uma restrição R_C é envoltória-consistente em relação à variável $x_j \in C$ se, e somente se, $D_j = \blacklozenge D_j{}'$ e R_C é GAC em relação a x_j sob o domínio $D_j{}'$;
- Uma restrição R_C é envoltória-consistente (total) se, e somente se, R_C é envoltória-consistente em relação a toda variável $x_j \in C$;
- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é envoltória-consistente se, e somente se, toda restrição $R_C \in C$ é envoltória-consistente.

No cenário ternário, a Definição 3.14 é simplificada pelo Teorema 3.9.

Teorema 3.9. *A restrição $R_C \equiv x_1 = x_2 \circ x_3$ sobre as variáveis x_1, x_2 e x_3 com domínios D_1, D_2 e D_3 é envoltória-consistente se, e somente se, $D_1 = \blacklozenge D_1{}'$, $D_2 = \blacklozenge D_2{}'$ e $D_3 = \blacklozenge D_3{}'$, tais que*

$$(D_1{}' \subseteq D_2{}' \circ D_3{}') \quad e \quad (D_2{}' \subseteq D_1{}' \bullet D_3{}') \quad e \quad (D_3{}' \subseteq D_1{}' \diamond D_2{}')$$

Demonstração. Consequência direta do Teorema 3.6 e da Definição 3.14. □

Dada uma restrição numérica R_C, a consistência de envoltória não garante que para todo valor no domínio da variável $x_j \in C$ existam valores nos domínios das variáveis em $C \setminus \{x_j\}$ que satisfaçam R_C, pois o domínio envoltória-consistente contém os "buracos" do domínio GAC e, para qualquer valor que a variável x_j assuma nesses buracos, não há uma extensão a $C \setminus \{x_j\}$ consistente com a restrição. Por outro lado, para os extremos do domínio D_j, garantidamente há uma extensão consistente, pois esses valores também pertencem ao conjunto arco-consistente $D_j{}'$, tal que $D_j = \blacklozenge D_j{}'$ (na prática, são os valores máximo e mínimo desse conjunto). Cruz e Barahona (2001) apresentam a consistência de envoltória baseando-se exatamente nessa observação: a restrição R_C é envoltória-consistente em relação à variável $x_j \in C$ se, e somente se, $D_j = [a, b]$ é um intervalo e as valorações $\langle x_j = a \rangle$ e $\langle x_j = b \rangle$ possuem extensões em $C \setminus \{x_j\}$ que satisfazem R_C.

O Algoritmo 13 descreve uma modificação do contrator da consistência de arco para alcançar a consistência de envoltória.

Algoritmo 13 Contrator envoltória ternário (`contratorEnvoltoria`(R_C, D_1, D_2, D_3))

Entrada: Restrição $R_C \equiv x_1 = x_2 \circ x_3$ sobre as variáveis $C = \{x_1, x_2, x_3\}$ com domínios D_1, D_2, D_3.
Saída: D_1, D_2 e D_3 atualizados, tais que R_C é envoltória-consistente.

1: $D_1 \leftarrow \blacklozenge(D_1 \cap D_2 \circ D_3)$
2: $D_2 \leftarrow \blacklozenge(D_2 \cap D_1 \bullet D_3)$
3: $D_3 \leftarrow \blacklozenge(D_3 \cap D_1 \diamond D_2)$

Teorema 3.10. *Dada uma restrição ternária $R_C \equiv x_1 = x_2 \circ x_3$, onde $\circ \in \{+, -, \cdot, /, \hat{}, \sqrt{}\}$ e, se $\circ \in \{\hat{}, \sqrt{}\}$, então D_3 contém um único valor inteiro,* `contratorEnvoltoria`(R_C, D_1, D_2, D_3) *consome tempo $O(l^2 \cdot \log l)$, onde l é o maior número de intervalos que compõem algum dos multi-intervalos D_1, D_2 ou D_3.*

Demonstração. As operações multi-intervalares da linha 1 são as mais custosas do algoritmo, pois na linha 2 o domínio D_1 foi reduzido a um único intervalo e, na linha 3, ambos os domínios D_1 e D_2 são intervalos. Na linha 1, efetua-se l^2 operações intervalares e consome-se tempo $O(l^2 \cdot \log l)$ para tornar o multi-intervalo resultante ordenado. A interseção de multi-intervalos ordenados é linear no seu número de intervalos e calcular a envoltória é constante. Pelas definições das operações básicas apresentadas na Seção 2.3.2, as operações intervalares $+$, $-$, $\cdot$, $/$, $\hat{}$ e $\sqrt{}$ são computadas em tempo constante. □

Exemplo 3.12. Seja a restrição $R_C \equiv x_1 = {x_2}^2$ sobre as variáveis x_1 e x_2 com domínios $D_1 = [1, 4]$ e $D_2 = [-2, 2]$. Aplicando, primeiramente, o contrator GAC, apresentado na seção anterior, obtém-se os seguintes refinamentos:

$$D_1 \leftarrow D_1 \cap {D_2}^2 = [1, 4] \cap [0, 4] = [1, 4]$$
$$D_2 \leftarrow D_2 \cap \pm\sqrt{D_1} = [-2, 2] \cap \langle [-2, -1], [1, 2] \rangle = \langle [-2, -1], [1, 2] \rangle$$

Por outro lado, utilizando a modificação que garante apenas a consistência de envoltória:

$$D_1 \leftarrow \blacklozenge(D_1 \cap {D_2}^2) = \blacklozenge(\underbrace{[1, 4] \cap [0, 4]}_{[1,4]}) = [1, 4]$$
$$D_2 \leftarrow \blacklozenge(D_2 \cap \pm\sqrt{D_1}) = \blacklozenge(\underbrace{[-2, 2] \cap \langle [-2, -1], [1, 2] \rangle}_{\langle [-2,-1],[1,2] \rangle}) = [-2, 2]$$

△

Benhamou et al. (1994) propuseram o algoritmo `Nar` para alcançar a consistência de envoltória em uma rede. Este algoritmo também é chamado de `HC3`, pois aplica o procedimento `contratorEnvoltoria` utilizando uma pilha de restrições de forma semelhante ao algoritmo `AC3`. Uma melhoria deste método, chamada `HC4`, foi proposta em Benhamou et al. (1999).

3.3.3 Consistência de Caixa

Quando uma restrição k-ária é decomposta em um conjunto de restrições ternárias, além do aumento no número de variáveis da rede, há uma ampliação do problema de localidade (ver Seção 2.3.3). Por outro lado, não é trivial verificar se uma restrição k-ária, para $k > 3$, é consistente, ou aplicar sobre esta um contrator. Apesar disso, é possível aproximar essa verificação de consistência (isto é, alcançar uma condição necessária, embora não suficiente) utilizando o conceito de extensão intervalar natural.

Exemplo 3.13. Seja a restrição $R_C \equiv {x_1}^2 - (x_1 x_2/x_3) + x_4 = 0$ sobre as variáveis x_1, x_2, x_3 e x_4 com domínios $D_1 = [-1, 1]$, $D_2 = [-1, 3]$, $D_3 = [1, 2]$ e $D_4 = [1, 2]$. A fim de verificar se R_C é envoltória-consistente, valora-se cada variável com os extremos do seu domínio e verifica-se se disto infere-se valores para as variáveis restantes dentro de seus respectivos domínios.

$$\langle x_1 = -1 \rangle \Rightarrow (-1)^2 - \frac{(-1)x_2}{x_3} + x_4 = 0 \Rightarrow 1 + \frac{x_2}{x_3} + x_4 = 0$$

Basicamente, é necessário verificar se 0 pertence ao conjunto imagem de $1 + \frac{x_2}{x_3} + x_4$. Seja $\mathcal{F}(X_1, X_2, X_3, X_4) = {X_1}^2 - (X_1 \cdot X_2)/X_3 + X_4$ a extensão intervalar natural da função que define R_C. Computando o valor de $\mathcal{F}$ com os domínios de x_2, x_3, x_4 e $\langle x_1 = -1 \rangle$, obtém-se:

$$\mathcal{F}([-1,-1], D_2, D_3, D_4) = \underbrace{[-1,-1]^2}_{[1,1]} - \underbrace{(\underbrace{[-1,-1] \cdot [-1,3]}_{[-3,1]} / [1,2])}_{[-3,1]} + [1,2] = [1,6]$$

Como $\mathcal{F}$ superestima o conjunto imagem de ${x_1}^2 - (x_1 x_2/x_3) + x_4$ e, para $\langle x_1 = -1 \rangle$, $\mathcal{F}([-1,-1], X_2, X_3, X_4)$ resulta em um intervalo que não contém 0, então a restrição não tem solução para essa valoração. Consequentemente, R_C não é envoltória-consistente. △

Em geral, como a extensão intervalar sempre superestima o conjunto imagem da função, se 0 não pertence ao intervalo computado pela extensão intervalar, então garantidamente não existe valor 0 no conjunto imagem da função original. Esse conceito de consistência, chamado de consistência de caixa, foi introduzido por Benhamou et al. (1994) e é uma relaxação da consistência de envoltória.

Definição 3.15 (Consistência de caixa).

- Uma restrição R_C da forma $R_C \equiv f(x_1, \ldots, x_k) = 0$ é caixa-consistente em relação à variável $x_j \in C$ se, e somente se, $D_j = \blacklozenge D_j'$ e, para todo $a_j \in D_j'$, $0 \in \mathcal{F}(D_1, \ldots, D_{j-1}, \{a_j\}, D_{j+1}, \ldots, D_k)$, onde $\mathcal{F}$ é a extensão intervalar natural de f;
- Uma restrição R_C é caixa-consistente (total) se, e somente se, R_C é caixa-consistente em relação a toda variável $x_j \in C$;

- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é caixa-consistente se, e somente se, toda restrição $R_C \in C$ é caixa-consistente.

Na prática, como a consistência de caixa também considera a envoltória do conjunto de valores consistentes, basta verificar se os extremos a_j de cada intervalo D_j satisfazem a condição: $0 \in \mathcal{F}(D_1, \ldots, D_{j-1}, \{a_j\}, D_{j+1}, \ldots, D_k)$.

Por outro lado, não é possível definir um contrator eficiente para a consistência de caixa, como ocorre com as consistências de arco e de envoltória, pois o principal objetivo dessa consistência é não efetuar a decomposição de restrições. Um método simples que visa refinar domínios inconsistentes como substituto ao contrator baseia-se em ramificar (particionar) intervalos inconsistentes e verificar a consistência de cada subintervalo separadamente.

Exemplo 3.14. Dando continuidade ao Exemplo 3.13, verifica-se que $D_1 = [-1, 1]$ não é caixa-consistente. Dessa forma, particiona-se esse intervalo em dois conjuntos $D_1{}' = [-1, 0]$ e $D_2{}'' = [0, 1]$ e computa-se a extensão intervalar $\mathcal{F}$ para os extremos de cada subintervalo separadamente:

$$\left.\begin{array}{l} \langle x_1 = -1 \rangle \Rightarrow \underbrace{[-1,-1]^2}_{[1,1]} - \underbrace{([-1,-1] \cdot [-1,3])/[1,2]}_{[-3,1]} + [1,2] = [1,6] \not\ni 0 \\ \langle x_1 = 0 \rangle \Rightarrow \underbrace{[0,0]^2}_{[0,0]} - \underbrace{([0,0] \cdot [-1,3])/[1,2]}_{[0,0]} + [1,2] = [1,2] \not\ni 0 \end{array}\right\} D_1{}' \text{ eliminado}$$

$$\left.\begin{array}{l} \langle x_1 = 0 \rangle \Rightarrow \underbrace{[0,0]^2}_{[0,0]} - \underbrace{([0,0] \cdot [-1,3])/[1,2]}_{[0,0]} + [1,2] = [1,2] \not\ni 0 \\ \langle x_1 = 1 \rangle \Rightarrow \underbrace{[1,1]^2}_{[1,1]} - \underbrace{([1,1] \cdot [-1,3])/[1,2]}_{[-1,3]} + [1,2] = [-1,4] \ni 0 \end{array}\right\} D_1{}'' \text{ inconclusivo}$$

Nesse caso, o subintervalo $[-1, 0]$ do domínio D_1 garantidamente não tem solução e, portanto, é seguro descartá-lo. O novo domínio $D_1 \leftarrow [0, 1]$ apresenta apenas um dos extremos consistentes. Logo, é preciso efetuar uma nova ramificação nesse intervalo. Quando ambos os extremos forem consistentes, então o subintervalo pode conter solução. Nesse caso, outra variável é escolhida e a estratégia é reaplicada, até que todas as restrições sejam caixa-consistentes. △

A consistência de caixa também possui as variantes **consistência de caixa(Γ)** (Benhamou et al., 1999) e **consistência de caixa⟨±φ⟩** (Granvilliers et al., 2000). No primeiro caso, para cada variável x_i de uma restrição $f(x_1, \ldots, x_k) = 0$, há uma extensão intervalar $\mathcal{F}_i$ de f distinta no conjunto Γ, e a verificação de consistência é efetuada para cada $\mathcal{F}_i$ separadamente; a consistência de caixa(Γ) é, portanto, uma extensão da consistência de caixa original (se todas as extensões $\mathcal{F}_i$ de f são idênticas, então consistência de caixa(Γ) e consistência de caixa são equivalentes). Na segunda variante, o teste $0 \in \mathcal{F}(D_1, \ldots, D_{j-1}, a_j, D_{j+1}, \ldots, D_k)$, para $a_j \in D_j$, é substituído pela relaxação $0 \in \mathcal{F}(D_1, \ldots, D_{j-1}, [a_j - \varphi, a_j + \varphi], D_{j+1}, \ldots, D_k)$ (naturalmente, se $\varphi = 0$, então consistência de caixa⟨±φ⟩ e consistência de caixa são equivalentes).

3.3.4 Consistência 3B

Como citado na Seção 3.3.2, um conceito equivalente à consistência de envoltória define que uma restrição R_C é envoltória-consistente em relação à variável $x_j \in C$ se, e somente se, $D_j = [a, b]$ é um intervalo e as valorações $\langle x_j = a \rangle$ e $\langle x_j = b \rangle$ possuem extensões em $C \setminus \{x_j\}$ que satisfazem R_C. Essa noção de consistência foi introduzida por Lhomme (1993) sob a nomenclatura 2B-consistência. No mesmo artigo, Lhomme apresenta uma extensão global dessa consistência chamada 3B-consistência.

Definição 3.16 (**3B-consistência**)**.**

- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é 3B-consistente em relação à variável $x_j \in \mathcal{X}$ se, e somente se, $D_j = [a, b]$ é um intervalo e as redes $\mathcal{R}' = (\mathcal{X}, \mathcal{D}, C \cup \{x_j = a\})$ e $\mathcal{R}'' = (\mathcal{X}, \mathcal{D}, C \cup \{x_j = b\})$ são envoltória-consistentes;
- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é 3B-consistente (total) se, e somente se, $\mathcal{R}$ é 3B-consistente em relação a toda $x_j \in \mathcal{X}$.

Exemplo 3.15. Seja uma rede composta pelas restrições $R_{C_1} \equiv x_1 + x_2 = 0$ e $R_{C_2} \equiv x_1 - x_2 = 0$, sobre as variáveis x_1 e x_2 com domínios $D_1 = D_2 = [-1, 1]$. Essa rede é GAC, RAC, envoltória-consistente e caixa-consistente. No entanto, a única solução possível de $\mathcal{R}$ é $\{\langle x_1 = 0 \rangle, \langle x_2 = 0 \rangle\}$, de forma que os domínios D_1 e D_2 poderiam ser reduzidos ao intervalo degenerado $[0, 0]$ sem perda de solução. Verificando-se a 3B-consistência de $\mathcal{R}$, a rede $\mathcal{R}' := \{R_{C_1}, R_{C_2}, R_{C_3} \equiv x_1 = -1\}$ deve ser envoltória-consistente:

$$
\begin{aligned}
&R_{C_3} \Rightarrow D_1 \leftarrow \blacklozenge(D_1 \cap [-1, -1]) = [-1, -1] \\
&R_{C_1} \Rightarrow D_2 \leftarrow \blacklozenge(D_2 \cap -D_1) = \blacklozenge([-1, 1] \cap [1, 1]) = [1, 1] \\
&R_{C_2} \Rightarrow D_2 \leftarrow \blacklozenge(D_2 \cap D_1) = \blacklozenge([1, 1] \cap [-1, -1]) = \emptyset
\end{aligned}
$$

Como D_2 foi reduzido ao conjunto vazio, então a rede $\mathcal{R}'$ não é localmente consistente e, portanto, $\mathcal{R}$ não é 3B-consistente. △

A mesma estratégia de partição de intervalos da consistência de caixa pode ser utilizada para reduzir domínios de restrições que não são 3B-consistentes. Assim, a cada iteração, reaplica-se os contratores locais em cada uma das restrições do sistema, descartando subintervalos ou efetuando novas ramificações, até que obtenha-se domínios consistentes. Nota-se que esse processo demanda um elevado esforço computacional, sendo impraticável no caso geral (Benhamou e Granvilliers, 2006). No Exemplo 3.15, a ramificação excluiria qualquer subintervalo de D_1 e D_2 que não contivesse 0, convergindo aos domínios $D_1 = D_2 = [0, 0]$.

Por fim, é possível generalizar a 3B-consistência em consistências globais ainda mais restritivas.

Definição 3.17 (*k*B**-consistência**)**.**

- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é kB-consistente ($k > 2$) em relação à variável $x_j \in \mathcal{X}$ se, e somente se, $D_j = [a, b]$ é um intervalo e as redes $\mathcal{R}' = (\mathcal{X}, \mathcal{D}, C \cup \{x_j = a\})$ e $\mathcal{R}'' = (\mathcal{X}, \mathcal{D}, C \cup \{x_j = b\})$ são $(k-1)$B-consistentes;
- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é kB-consistente (total) se, e somente se, $\mathcal{R}$ é kB-consistente em relação a toda $x_j \in \mathcal{X}$.

3.4 Métodos intervalares aplicados a problemas numéricos

Métodos intervalares são capazes de resolver problemas numéricos de forma completa e rigorosa, isto é, sem excluir ou aproximar soluções devido a erros de representação numérica. Os primeiros métodos intervalares foram propostos por Moore (1966) como extensões de técnicas usuais para encontrar raízes de funções e integração numérica. Hansen (1980) mostrou como a análise intervalar pode ser utilizada para computar mínimos globais de funções diferenciáveis. Na década de 1990, um grande número de trabalhos estenderam conceitos de CSPs para o contexto numérico através da análise intervalar (Hyvönen, 1989; Lhomme, 1993; Sam-Haroud e Faltings, 1996; Faltings, 1998). Em particular, Neumaier (1991) e van Hentenryck et al. (1997) combinaram técnicas intervalares com métodos de ramificação e poda. Um trabalho central no cenário de otimização e satisfazibilidade intervalar é a linguagem Numerica, desenvolvida por Van Hentenryck (1997), cujo algoritmo central constitui a base da grande maioria de resolvedores intervalares atuais.

Informalmente, métodos intervalares computam um conjunto de domínios atômicos (intervalos ou multi-intervalos de comprimento mínimo) que contém todas as soluções da rede. Por exemplo, como mencionado na Seção 3.3.1, alcançar GAC em uma rede numérica através da aplicação sucessiva de contratores é um processo que pode não terminar, exceto pela representação finita dos números de máquinas; por outro lado, ao estabelecer uma precisão mínima aos multi-intervalos resultantes, esse método sempre termina e, no sentido de não excluir soluções da rede, continua sendo completo. Em contrapartida, não há garantia de que redes inconsistentes sejam detectadas por métodos intervalares, pois nem todo o espaço de busca é percorrido.

Dessa forma, dada uma rede de restrições numéricas $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, considera-se o problema de reduzir o conjunto de domínios $\mathcal{D}$ de forma que todo $D_i \in \mathcal{D}$ tenha um comprimento menor ou igual a um valor $\Delta > 0$ prefixado, sem excluir soluções da rede original. Esse problema é uma relaxação do NCSP original cuja solução caracteriza-se pela associação de cada variável x_i a um subdomínio consistente $D_i{}'$ de comprimento no máximo Δ (Benhamou e Granvilliers, 2006). Alternativamente, se $\Delta = 0$, isto é, se cada domínio contém um único valor, então essa associação é de fato uma valoração da forma $\{\langle x_1 = a_1\rangle, \ldots, \langle x_n = a_n\rangle\}$ consistente com a rede.

Diversos trabalhos propõem e analisam algoritmos para abordar este problema, inclusive quando deseja-se obter uma solução ótima sob algum critério (otimização global). Nas Seções 3.4.1 e 3.4.2 apresenta-se uma breve descrição dos métodos intervalares mais usuais. Mais detalhes podem ser encontrados nos livros de Hansen e Walster (2004) e Moore et al. (2009), bem como em Benhamou e Granvilliers (2006). Uma revisão atualizada e mais sucinta pode ser encontrada em Araya e Reyes (2016).

3.4.1 Satisfação de restrições numéricas

As técnicas de consistência de NCSPs apresentadas na Seção 3.3 reduzem domínios de variáveis garantindo completude; no entanto, não se fixa uma precisão mínima que cada intervalo deve satisfazer. Como resultado, se a rede original já é localmente consistente, então os domínios não serão reduzidos à precisão preestabelecida. O método de **ramificação e poda**[3], introduzido originalmente em Land e Doig (1960) para resolver problemas discretos, é usualmente utilizado para processar problemas intervalares (Hyvönen, 1989; Neumaier, 1991; Lhomme, 1993; van Hentenryck et al., 1997; Cruz e Barahona, 2001). Intuitivamente, cada domínio consistente é particionado em dois subdomínios disjuntos, aos quais aplica-se os contratores separadamente; esse processo repete-se para cada subdomínio até que intervalos de precisão pelo menos Δ sejam obtidos. De modo geral, um método de ramificação e poda (como descrito pelo algoritmo[4] 14) consiste de duas fases principais :

- **Poda** (`contrator_consistencia`): contratores são aplicados sobre o conjunto de domínios $\mathcal{D}$ a fim de reduzir domínios inconsistentes. Se algum domínio D_j for reduzido ao intervalo vazio, então o conjunto $\mathcal{D}$ não contém solução e, portanto, deve ser descartado do espaço de busca;

- **Ramificação** (`seleciona_e_bissecta`): se o conjunto de domínios obtido na fase anterior não for suficiente como solução da instância, isto é, houver algum domínio $D_j \in \mathcal{D}$ de comprimento maior que Δ, então esse intervalo deve ser particionado em subintervalos D_j' e D_j'', gerando dois conjuntos de domínios $\mathcal{D}' = \{D_1, \ldots, D_{j-1}, D_j', D_{j+1}, \ldots, D_n\}$ e $\mathcal{D}'' = \{D_1, \ldots, D_{j-1}, D_j'', D_{j+1}, \ldots, D_n\}$ distintos, para os quais o método é aplicado recursivamente.

O método de ramificação e poda é, em essência, um método de retrocesso no qual cada ramificação constitui um nó na árvore de busca. Primeiramente, uma das partições do domínio é processada, para então processar-se a outra, sendo que soluções parcialmente inconsistentes são descartadas do espaço de busca (linha 6 do Algoritmo 14). A principal diferença é que nesse método deseja-se obter um conjunto S com todas as possíveis soluções da rede (linha 12), ao invés da primeira encontrada.

[3] Há uma ambiguidade nas traduções de *Branch and Prune* e *Branch and Bound* para o português, ambos usualmente traduzidos para Ramificação e Poda, embora sejam métodos diferentes.

[4] Os Algoritmos 14 e 15 são baseados nos respectivos algoritmos propostos em Araya e Reyes (2016).

Algoritmo 14 NCSP intervalar (`NCSP_intervalar`$(\mathcal{R}, \Delta)$)

Entrada: Rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ e precisão $\Delta > 0$.
Saída: $S = \{\mathcal{D}_1, \ldots, \mathcal{D}_l\}$ tal que $\forall \mathcal{D}_j \in S : \forall D_i \in \mathcal{D}_j : 0 < c(D_i) \leq \Delta$.

```
 1: S ← ∅
 2: empilha(D, pilha)
 3: enquanto pilha ≠ ∅ faça
 4:     D ← desempilha(pilha)
 5:     D ← contrator_consistencia(X, D, C)
 6:     se ∀D_i ∈ D : D_i ≠ ∅ então
 7:         se ∃D_i ∈ D tal que c(D_i) > Δ então
 8:             (D', D'') ← seleciona_e_bissecta((X, D, C), Δ)
 9:             empilha(D', pilha)
10:             empilha(D'', pilha)
11:         senão
12:             S ← S ∪ {D}
13:         fim se
14:     fim se
15: fim enquanto
16: devolve INCONSISTENTE
```

A função `contrator_consistencia` (linha 5) alcança alguma das consistências apresentadas na Seção 3.3. Por exemplo, o método `Newton`, proposto por van Hentenryck et al. (1997), aplica um contrator da consistência de caixa na fase de poda, enquanto o método `iSAT`, proposto por Franzle et al. (2007), alcança a consistência de envoltória em restrições decompostas.

A função `seleciona_e_bissecta` é responsável por decidir o próximo domínio a ser particionado na fase de ramificação. Nos vários métodos de resolução de sistemas de restrições essa escolha é baseada em heurísticas, e tem um forte impacto no tempo de processamento de caso médio do algoritmo. Em geral, escolhe-se o domínio de maior comprimento ou utiliza-se algum esquema baseado em Smear (Csendes e Ratz, 1997) ou em ranqueamento de variáveis com base na frequência de redução com a aplicação do contrator (Moskewicz et al., 2001). Alternativamente, ao invés de bissectar o domínio escolhido, pode-se utilizar métodos de busca local, como o método de Newton (Moore, 1966), para uma convergência mais rápida.

Exemplo 3.16. Seja $\mathcal{R}$ a rede de restrições da instância *sample* do *benchmark* COCONUT (Exemplo 1.1, desconsiderando-se a função objetivo), uma precisão $\Delta = 10^{-1}$ e o contrator da consistência de envoltória `HC3` como método de poda no Algoritmo 14. Devido a esse contrator, a rede deve ser decomposta em uma rede equivalente de restrições ternárias, adicionando-se variáveis auxiliares $\{y_1, \ldots, y_7\}$, com domínios $D_{y_i} = (-\infty, +\infty)$, e $\{z_1, \ldots, z_7\}$, com domínios $D_{z_i} = (-\infty, +\infty)$, e duas variáveis c_1 e c_2 chamadas variáveis de folga, tais que $D_{c_1} = (-\infty,\ 0{,}0401]$ e $D_{c_2} = (-\infty,\ 0{,}010085]$, que permitem a representação de inequações:

$$4/x_1 + 2{,}25/x_2 + 1/x_3 + 0{,}25/x_4 \leq 0{,}0401 \Rightarrow \begin{cases} y_1 = 4/x_1 & y_5 = y_1 + y_2 \\ y_2 = 2{,}25/x_2 & y_6 = y_5 + y_3 \\ y_3 = 1/x_3 & y_7 = y_6 + y_4 \\ y_4 = 0{,}25/x_4 & y_7 - c_1 = 0 \end{cases}$$

$$0{,}16/x_1 + 0{,}36/x_2 + 0{,}64/x_3 + 0{,}64/x_4 \leq 0{,}010085 \Rightarrow \begin{cases} z_1 = 0{,}16/x_1 & z_5 = z_1 + z_2 \\ z_2 = 0{,}36/x_2 & z_6 = z_5 + z_3 \\ z_3 = 0{,}64/x_3 & z_7 = z_6 + z_4 \\ z_4 = 0{,}64/x_4 & z_7 - c_2 = 0 \end{cases}$$

Ao executar o Algoritmo 14, o contrator é aplicado sobre esta rede, obtendo o seguinte conjunto de domínios envoltória-consistentes:

$D_1 = [100, 400000]$	$D_{y_4} = [0{,}0000025,\ 0{,}0025]$	$D_{z_4} = [0{,}0000064,\ 0{,}0064]$
$D_2 = [100, 300000]$	$D_{y_5} = [0{,}0000175,\ 0{,}0400925]$	$D_{z_5} = [0{,}0000016,\ 0{,}0052]$
$D_3 = [100, 200000]$	$D_{y_6} = [0{,}0000225,\ 0{,}0400975]$	$D_{z_6} = [0{,}0000048,\ 0{,}0100786]$
$D_4 = [100, 100000]$	$D_{y_7} = [0{,}000025,\ 0{,}0401]$	$D_{z_7} = [0{,}0000112,\ 0{,}010085]$
$D_{y_1} = [0{,}00001,\ 0{,}04]$	$D_{z_1} = [0{,}0000004,\ 0{,}0016]$	$D_{c_1} = [0{,}000025,\ 0{,}0401]$
$D_{y_2} = [0{,}0000075,\ 0{,}0225]$	$D_{z_2} = [0{,}0000012,\ 0{,}0036]$	$D_{c_2} = [0{,}0000112,\ 0{,}010085]$
$D_{y_3} = [0{,}000005,\ 0{,}01]$	$D_{z_3} = [0{,}0000032,\ 0{,}0064]$	

Embora não existam domínios vazios (condicional da linha 6), os intervalos não apresentam a precisão numérica necessária ($c(D_1) = 399900 > \Delta$). Assim, uma variável é escolhida, por exemplo x_1, que apresenta o domínio de maior comprimento, e seu domínio é particionado em subdomínios $D_1' = [100, 200000]$ e $D_1'' = (200000, 400000]$ (linha 8). Então, o método é executado recursivamente para cada subdomínio, verificando se o conjunto obtido contém intervalos vazios ou se a condição de comprimento mínimo é satisfeita. Ao final, o conjunto S (linha 12) é construído contendo todas as partições de domínios que são consistentes e, portanto, podem conter uma solução da rede. △

3.4.2 Otimização global

Em diversos problemas de áreas como engenharia, economia, produção, planejamento e logística, não basta encontrar uma valoração de variáveis que satisfaça um conjunto de restrições, mas deseja-se que esta valoração também seja ótima sob algum critério. Esses problemas são, na grande maioria das vezes, representados matematicamente pela programação não linear, ou, utilizando a terminologia da área de satisfação de restrições, por NCOPs (ver Definição 2.20), nos quais deseja-se encontrar uma valoração consistente de uma rede de restrições numéricas que

minimiza uma dada função objetivo. Da mesma forma que métodos intervalares são capazes de aproximar soluções de NCSPs de forma completa e rigorosa, é possível aplicá-los em problemas de otimização global a fim de obter um conjunto de intervalos de precisão mínima que contém uma solução ótima do problema.

Em contraste com os métodos da comunidade de programação matemática, usualmente aplicados a problemas de otimização global, que não garantem, no caso mais geral, uma solução ótima da instância (apenas um ótimo local), métodos intervalares garantem a otimalidade global da solução, pois percorrem todo o espaço de busca.

O Algoritmo 15 descreve uma modificação do Algoritmo 14 capaz de resolver problemas de otimização (minimização da função objetivo). Além dos procedimentos `contrator_consistencia` e `seleciona_e_bissecta`, presentes no primeiro método, o procedimento `limitante_superior` é adicionado à linha 7 do algoritmo. Este procedimento encontra uma valoração qualquer da rede de restrições e computa a função objetivo com essa valoração, devolvendo o seu resultado z. Nesse sentido, z é um limitante superior do ótimo global da instância. Diversos métodos numéricos podem ser utilizados para implementar esse procedimento, como o método de Newton (Moore, 1966), métodos baseados no vetor gradiente (Hestenes, 1969; Potra e Wright, 1997), ou construir uma solução a partir de uma aproximação linear obtida através do método Simplex (Suprajitno e bin Mohd, 2010). Nota-se que, neste ponto, a solução encontrada está sujeita a erros numéricos. Alternativamente, pode-se utilizar a própria aritmética intervalar para verificar a existência de solução, computando um subdomínio de precisão mínima consistente; intuitivamente, valora-se algumas variáveis em seu ponto médio e computa-se o conjunto de intervalos aplicando contratores. Alguns resolvedores intervalares, como o IBBA e o IbexOpt, propostos, respectivamente, em Messine (2005) e Trombettoni et al. (2011), implementam esta estratégia.

Outra diferença desse método em relação ao Algoritmo 14 é o segundo condicional da linha 6 ($\min \mathcal{F}(D_1, \ldots, D_n) < z^*$), que exclui os ramos da árvore de busca que garantidamente não possuem uma solução melhor que z^*. A corretude desta poda baseia-se na inclusão isotônica da extensão intervalar, isto é, se $D_1{}' \subseteq D_1, \ldots, D_n{}' \subseteq D_n$, então $\mathcal{F}(D_1{}', \ldots, D_n{}') \subseteq \mathcal{F}(D_1, \ldots, D_n)$. Dessa forma, se o menor valor em $\mathcal{F}(D_1, \ldots, D_n)$ é maior que z^*, então não há necessidade de prosseguir com a busca em subdomínios de $\{D_1, \ldots, D_n\}$. Alternativamente, esse condicional pode ser inserido na rede na forma de uma restrição $f(x_1, \ldots, x_n) < z^*$. Assim, contratores são capazes de reduzir ainda mais os domínios de $x_1, \ldots, x_n$, conforme mostra o Exemplo 3.17.

Exemplo 3.17. Seja $f(x_1) = {x_1}^2$ uma função objetivo, $z^* = 4$ um limitante superior do ótimo global dessa função, e dois domínios $D_1{}' = [0, 5)$ e $D_1{}'' = [5, 10]$. Pelo condicional $\min \mathcal{F}(D) < z^*$, apenas o domínio $D_1{}'$ é processado, pois $\min \mathcal{F}(D_1{}') = \min[0, 25) < 4$ e $\min \mathcal{F}(D_1{}'') = \min[25, 100] \not< 4$. Por outro lado, ao considerar-se a restrição $f(x) < z^*$, é possível inferir $D_1{}' \leftarrow D_1{}' \cap \pm\sqrt{(-\infty, 4]} = [0, 2]$, reduzindo ainda mais o espaço de busca. $\triangle$

Algoritmo 15 NCOP intervalar (`NCOP_intervalar`$(f, \mathcal{R}, \Delta)$)

Entrada: Função objetivo f, rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ e precisão $\Delta > 0$.
Saída: $\mathcal{D}^*$ contém a solução da instância NCOP definida por f e $\mathcal{R}$.

```
1: (D*, z*) ← (∅, +∞)
2: empilha(D, pilha)
3: enquanto pilha ≠ ∅ faça
4:    D ← desempilha(pilha)
5:    D ← contrator_consistencia(X, D, C)
6:    se ∀D_i ∈ D : D_i ≠ ∅ e min F(D_1, ..., D_n) < z* então
7:       z ← limitante_superior(f, (X, D, C))
8:       se z < z* então
9:          (D*, z*) ← (D, z)
10:      fim se
11:      se ∃D_i ∈ D tal que c(D_i) > Δ então
12:            (D', D'') ← seleciona_e_bissecta((X, D, C), Δ)
13:            empilha(D', pilha)
14:            empilha(D'', pilha)
15:      fim se
16:   fim se
17: fim enquanto
18: devolve (D*, z*)
```

Exemplo 3.18. Seja a instância de programação não linear *sample*, apresentada no Exemplo 1.1, e a decomposição ternária da rede de restrições conforme definida no Exemplo 3.16, resultando no seguinte problema:

$$\min \quad x_1 + x_2 + x_3 + x_4 \tag{3.4}$$

sujeito a

$$\begin{array}{lll} y_1 = 4/x_1 & y_7 = y_6 + y_4 & z_5 = z_1 + z_2 \\ y_2 = 2{,}25/x_2 & y_7 - c_1 = 0 & z_6 = z_5 + z_3 \\ y_3 = 1/x_3 & z_1 = 0{,}16/x_1 & z_7 = z_6 + z_4 \\ y_4 = 0{,}25/x_4 & z_2 = 0{,}36/x_2 & z_7 - c_2 = 0 \\ y_5 = y_1 + y_2 & z_3 = 0{,}64/x_3 & \\ y_6 = y_5 + y_3 & z_4 = 0{,}64/x_4 & \end{array} \tag{3.5}$$

Utilizando uma precisão $\Delta = 10^{-1}$ e a consistência de envoltória, o procedimento `contrator_consistencia` obtém o conjunto de intervalos consistentes apresentado no Exemplo 3.16, sob o qual a extensão intervalar da função objetivo computa $\mathcal{F}(D_1, D_2, D_3, D_4) = D_1 + D_2 + D_3 + D_4 = [400, 1000000]$. Como não há domínios vazios e $\min \mathcal{F}(D_1, D_2, D_3, D_4) = 400 < z^* = +\infty$, o procedimento `limitante_superior` é executado, o qual deve devolver uma valoração consistente I, não necessariamente ótima, sob a qual $f(x_1, x_2, x_3, x_4) = z$. A solução ótima é então atualizada para z, na linha 9 do algoritmo. Nesse caso, no entanto, o condicional da linha 11 é satisfeito, pois existem intervalos de comprimento maior que Δ (por exemplo, D_1). Como no Exemplo 3.16, D_1 é particionado e o método prossegue de forma

análoga para cada partição de domínio $\mathcal{D}'$ e $\mathcal{D}''$ inserido na pilha (linhas 13 e 14). Após mais algumas iterações deste método, aplicado ao subproblema definido por $\mathcal{D}'$, os seguintes intervalos consistentes são obtidos:

$$
\begin{array}{lll}
D_1 = [193{,}44794,\ 193{,}44794] & D_{y_4} = [0{,}0014825,\ 0{,}0014825] & D_{z_4} = [0{,}003795,\ 0{,}00379] \\
D_2 = [179{,}43635,\ 179{,}43635] & D_{y_5} = [0{,}0332166,\ 0{,}0332166] & D_{z_5} = [0{,}002833,\ 0{,}00283] \\
D_3 = [185{,}162721,\ 185{,}21844] & D_{y_6} = [0{,}0386156,\ 0{,}0386173] & D_{z_6} = [0{,}006288,\ 0{,}00628] \\
D_4 = [168{,}63395,\ 168{,}63395] & D_{y_7} = [0{,}0400981,\ 0{,}0400998] & D_{z_7} = [0{,}010083,\ 0{,}01008] \\
D_{y_1} = [0{,}0206773,\ 0{,}0206773] & D_{z_1} = [0{,}0008270,\ 0{,}0008270] & D_{c_1} = [0{,}040098,\ 0{,}04009] \\
D_{y_2} = [0{,}0125392,\ 0{,}0125392] & D_{z_2} = [0{,}0020062,\ 0{,}0020062] & D_{c_2} = [0{,}010083,\ 0{,}01008] \\
D_{y_3} = [0{,}0053990,\ 0{,}0054006] & D_{z_3} = [0{,}003455,\ 0{,}00345] &
\end{array}
$$

Nesse caso, a extensão intervalar da função objetivo computa $\mathcal{F}(D_1, D_2, D_3, D_4) = [726{,}68098,\ 726{,}73670]$ e z^* é atualizado para um valor que é, no máximo, 726,73670; nota-se que z^* não é necessariamente o ótimo global da instância, mas, dentro dos domínios atuais, difere deste por um fator de no máximo $|726{,}68098 - 726{,}73670| < 0{,}1$. Quando a partição $\mathcal{D}''$, inserida na pilha na primeira iteração do algoritmo, for executada, a extensão intervalar da função objetivo computará o intervalo $(200300, 1000000]$ que, pela inclusão isotônica da aritmética intervalar, não contém solução melhor que z^*. Portanto, todo esse ramo será descartado do espaço de busca sem a necessidade de novas bissecções. △

Embora a análise intervalar tenha sido proposta há cinco décadas, como forma de medir erros de aproximação em sistemas computacionais, sua aplicação na otimização global é relativamente recente. De modo geral, algoritmos de ramificação e poda, tais como enumeração exaustiva em cenários combinatórios, são de difícil aplicação prática. Por outro lado, técnicas que permitem extrair informações implícitas das instâncias do problema, seja reduzindo o espaço de busca, tais como propagação de restrições, aprendizado e retrocesso não cronológico, ou guiando a busca a fim de encontrar soluções mais rapidamente (métodos heurísticos e meta-heurísticos), vêm tornando tais algoritmos uma opção real de solução de problemas de otimização.

Atualmente, muitos problemas são resolvidos através de métodos de otimização local da comunidade de programação matemática (Sahinidis, 1996; Misener e Floudas, 2014), mesmo que o sistema seja não convexo ou apresente múltiplos mínimos locais. Essa relaxação se dá, principalmente, pela eficiência desses métodos em comparação aos métodos de busca exaustiva para resolver otimização global (Neumaier et al., 2005; Araya et al., 2014). Com o poder computacional atual, no entanto, existem recursos e ferramentas, como o paralelismo (Kreinovich e Bernat, 1994) por exemplo, que viabilizam a aplicação de métodos de otimização global em algumas classes de instâncias de grande porte. Um ótimo exemplo dessa situação é o problema de encontrar o menor número de movimentos capazes de resolver qualquer posição do Cubo de Rubik, também conhecido como cubo mágico. Esse problema é estudado desde a década de

1980, através de algoritmos de otimização que utilizavam-se de relaxações e estimativas, obtendo resultados que não eram, de fato, o ótimo global do sistema. Apenas em 2014 foi possível efetuar uma busca exaustiva pelo menor número de movimentos que resolve todas as posições do cubo (aproximadamente[5] $4 \cdot 10^{19}$), utilizando técnicas de redução de domínios e limitantes de ótimo global, obtendo como resultado o valor mínimo de 26 movimentos (Rokicki et al., 2014).

No cenário intervalar, Kampen (2010) utilizou métodos intervalares para resolver uma série de problemas aeroespaciais. Em Zhu et al. (2011), a análise intervalar foi aplicada em problemas de otimização no cenário energético de Beijing, China. Uma revisão de algoritmos intervalares para otimização global pode ser encontrada em Hansen e Walster (2004) e métodos que se utilizam de informações de convexidade e diferenciabilidade da função de custo, no cenário intervalar, são apresentados em Sotiropoulos e Grapsa (2001) e Bhurjee e Panda (2012). Uma ampla gama de otimizadores baseados em métodos intervalares foram propostos na última década, como o IBBA (Messine, 2005), GlobSol (Kearfott, 2009), Icos (Lebbah, 2009), IbexOpt (Trombettoni et al., 2011), e o já citado NUMERICA (Van Hentenryck, 1997).

3.5 Busca livre de retrocesso

Na resolução de um problema de satisfação de restrições, busca e inferência são usualmente combinadas. Em alguns casos, no entanto, a quantidade de inferência efetuada (ou, equivalentemente, o nível de consistência alcançado) garante que a busca não efetue retrocesso. Por exemplo, em uma rede com n variáveis, n-consistência forte é suficiente para garantir consistência global, e, portanto, qualquer valoração parcial pode ser consistentemente estendida a uma solução do problema. Infelizmente, alcançar esse nível de consistência demanda um elevado esforço computacional, de forma que, na prática, mostra-se um processo menos eficaz do que aplicar apenas uma parcela dessas inferências e obter a solução através de uma busca com retrocesso.

Por outro lado, consistência global não é condição necessária para a existência de uma ordenação de variáveis $b = (x_1, \ldots, x_n)$ com a qual seja possível valorar todas as variáveis sem efetuar retrocesso; outras propriedades também podem ser condições suficientes para a existência dessa ordenação. Além disso, conhecendo a ordenação b *a priori*, não é necessário que se aplique consistência em relação a toda combinação de variáveis na rede, apenas em relação a cada variável com suas antecessoras em b. Intuitivamente, se a busca vai ocorrer de acordo com esta ordenação, então, para cada valor no domínio de x_i, devem existir valores consistentes com a rede nos domínios das variáveis $x_{i+1}, \ldots, x_n$, para todo $1 \leq i < n$. Por exemplo, não é necessário que toda a rede seja fortemente n-consistente, mas apenas os subconjuntos de restrições com j

[5] O número exato é 43.252.003.274.489.856.000.

variáveis antecessoras a x_i, para todo $1 \leq j \leq i$ e todo $1 \leq i \leq n$. Este conceito de consistência é chamado de consistência direcionada[6], e será abordado brevemente na Seção 3.5.1.

A relação entre o número de variáveis da rede e o nível de k-consistência forte alcançada não é a única condição suficiente para uma busca sem retrocesso proposta na literatura. Esse tema de pesquisa iniciou-se com Freuder (1982), embora já tivesse sido idealizado por Waltz ao notar que determinadas instâncias de figuras bidimensionais de poliedros poderiam ser rotuladas sem a necessidade de retrocesso, apenas alcançando-se o que viria a ser chamada de consistência de arco. Na Seção 3.5.2, alguns dos resultados mais importantes sobre essas relações são apresentados. Por fim, mesmo que determinada instância não seja caracterizada por alguma dessas relações, ela ainda pode ser decomposta em subproblemas para os quais existe uma condição suficiente de solução sem retrocesso. Esse procedimento de decomposição e resolução é então utilizado para resolver a instância original. A Seção 3.5.3 apresenta duas dessas técnicas: o método de corte de ciclo e a decomposição em árvore.

3.5.1 Consistência direcionada

Dechter e Pearl (1987b) propuseram a consistência direcionada motivados pelo fato de que consistência total é por vezes desnecessária se uma solução será gerada por uma busca em uma ordenação fixa de variáveis. Define-se aqui a versão direcionada das principais consistências apresentadas na Seção 3.2, mas outras consistências locais, inclusive de NCSPs, pode ser definidas de forma análoga.

Definição 3.18 (Consistência de arco direcionada). Dada uma rede binária $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ e uma ordenação $b = (x_1, \ldots, x_n)$ das variáveis em $\mathcal{X}$, uma restrição $R_C \in C$ é arco-consistente direcionada se, e somente se, R_C é arco-consistente em relação à variável $x_i \in C$ com o menor índice em b. A rede $\mathcal{R}$, por sua vez, é arco-consistente direcionada se, e somente se, toda restrição $R_C \in C$ é arco-consistente direcionada.

Informalmente, uma restrição binária $R_{\{x_i,x_j\}}$ é arco-consistente direcionada se, e somente se, para todo valor a_i no domínio da variável x_i com menor índice em b (isto é, $i < j$), existe $a_j \in D_j$ tal que $\{\langle x_i = a_i\rangle, \langle x_j = a_j\rangle\}$ satisfaz $R_{\{x_i,x_j\}}$.

Exemplo 3.19. Seja $\mathcal{R}$ uma rede binária composta pelas restrições $R_{C_1} \equiv x_1 < x_2$ e $R_{C_2} \equiv x_2 < x_3$, e pelos domínios contínuos $D_1 = [2, 3]$, $D_2 = [1, 5]$, $D_3 = [0, 6]$. Esta rede é arco-consistente direcionada em relação à ordenação $b_1 = (x_1, x_2, x_3)$, pois (i) para todo $a_1 \in [2, 3]$ existe $a_2 \in [1, 5]$ tal que $a_1 < a_2$ e (ii) para todo $a_2 \in [1, 5]$ existe $a_3 \in [0, 6]$ tal que $a_2 < a_3$. Por outro lado, $\mathcal{R}$ não é arco-consistente direcionada em relação à ordenação $b_2 = (x_3, x_2, x_1)$, pois a valoração parcial $\langle x_3 = 0\rangle$ não possui extensão na variável x_2 satisfazendo R_{C_2}. De fato, b_1 é a única ordenação de variáveis sob a qual $\mathcal{R}$ é arco-consistente direcionada. △

[6]Em Norvig e Russell (2017), o termo original *directional arc-consistency* foi traduzido para "consistência orientada de arco". Aqui, optou-se pela tradução mais literal "consistência de arco direcionada".

O algoritmo para alcançar consistência de arco direcionada é semelhante ao algoritmo `AC-1` original. No entanto, basta que cada restrição seja processada uma única vez. O Algoritmo 16 descreve esse método, recebendo como entrada uma ordenação de variáveis b e processando as variáveis no sentido inverso desta ordenação. Isso garante que cada restrição processada seja arco-consistente em relação à sua variável com menor índice em b e que restrições previamente processadas não deixem de ser arco-consistentes direcionadas, pois apenas domínios de variáveis antecessoras estão sendo reduzidos

Algoritmo 16 Consistência de arco direcionada (`DAC`$(\mathcal{R}, b)$)

Entrada: Rede de restrições binárias $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ e ordenação de variáveis $b = (x_1, \ldots, x_n)$.
Saída: $\mathcal{D}$ atualizado, tal que $\mathcal{R}$ é arco-consistente direcionada.

```
1: para i ← n .. 1 faça
2:     para todo R_{x_i,x_j} ∈ C tal que j < i faça
3:         revisa(R_{x_i,x_j}, D_i, D_j, x_j)
4:     fim para
5: fim para
```

Teorema 3.11. `DAC`$(\mathcal{R}, b)$ *consome tempo* $O(md^2)$*, onde* m *é o número de restrições de* $\mathcal{R}$ *e* d *é o tamanho do maior domínio na rede.*

Demonstração. Cada restrição $R_{\{x_i,x_j\}}$ é processada uma única vez, pois o contador i é decrementado em cada iteração do laço. Cada processamento, por sua vez, é composto por uma chamada da função `revisa`, que consome tempo $O(d^2)$. □

O conceito de consistência de arco direcionada é naturalmente estendido ao de k-consistência direcionada. Nesse caso, deve-se verificar a consistência em relação a todas as restrições da rede com variáveis antecessoras na ordenação.

Definição 3.19 (**k-consistência direcionada**)**.** Uma rede $\mathcal{R}$ é dita k-consistente direcionada, em relação a uma ordenação de variáveis $b = (x_1, \ldots, x_n)$, se, e somente se, para toda valoração consistente $\{\langle x_1 = a_1\rangle, \ldots, \langle x_{k-1} = a_{k-1}\rangle\}$ de quaisquer $k-1$ variáveis, existe um valor a_k no domínio de qualquer outra variável x_k que sucede as $k-1$ variáveis em b tal que $\{\langle x_1 = a_1\rangle, \ldots, \langle x_{k-1} = a_{k-1}\rangle, \langle x_k = a_k\rangle\}$ é consistente com $\mathcal{R}$. Uma rede $\mathcal{R}$ é dita fortemente k-consistente direcionada se é j-consistente direcionada , para todo $1 \leq j \leq k$.

Do mesmo modo, as consistências de arco generalizada (GAC) e relacional (RAC) são definidas também nas versões direcionadas.

Definição 3.20 (**Consistência de arco generalizada (GAC) direcionada**)**.** Dada uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ e uma ordenação $b = (x_1, \ldots, x_n)$ das variáveis em $\mathcal{X}$, uma restrição $R_C \in C$ é GAC direcionada se, e somente se, R_C é GAC em relação à variável $x_i \in C$ com o menor índice em b. A rede $\mathcal{R}$, por sua vez, é GAC direcionada se, e somente se, toda restrição $R_C \in C$ é GAC direcionada.

Definição 3.21 (Consistência de arco relacional (RAC) direcionada). Dada uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ e uma ordenação $b = (x_1, \ldots, x_n)$ das variáveis em $\mathcal{X}$, uma restrição $R_C \in C$ é RAC direcionada se, e somente se, R_C é RAC em relação à variável $x_i \in C$ com o maior índice em b. A rede $\mathcal{R}$, por sua vez, é RAC direcionada se, e somente se, toda restrição $R_C \in C$ é RAC direcionada.

Informalmente, uma restrição R_C é GAC direcionada se para todo valor no domínio da variável $x_i \in C$ com menor índice em b, existem valores em todas as variáveis sucessoras de x_i em b de forma a satisfazer R_C. Por outro lado, uma restrição R_C é RAC direcionada se, para toda valoração consistente de variáveis em C, excluindo-se a variável x_i de maior índice em b, existe uma extensão a esta variável satisfazendo R_C. Aqui também percebe-se a propriedade complementar entre GAC e RAC.

O Algoritmo 17 descreve um método de alcançar GAC direcionada em uma rede $\mathcal{R}$, dada uma ordenação de variáveis b. Do mesmo modo, processa-se as variáveis em ordem decrescente em b. Ao processar-se uma variável x_i, todas as restrições que contêm x_i e outra variável de índice menor em b são processadas pelo método `revisaGAC`, tornando estas restrições GAC em relação a sua variável de menor índice x_j (isto é, apenas D_j é reduzido). Se esse processamento alterar a consistência de uma restrição $R_C{}'$ já processada, é porque existe uma outra variável nesta restrição de índice menor que j em b e, portanto, $R_C{}'$ será processada novamente. Por outro lado, restrições compostas apenas por variáveis com índices maiores que x_j em b mantêm-se GAC, pois os domínios de suas variáveis não serão reduzidos.

Algoritmo 17 Consistência de arco generalizada direcionada (`GAC_direcionada`$(\mathcal{R}, b)$)

Entrada: Rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ e ordenação de variáveis $b = (x_1, \ldots, x_n)$.
Saída: $\mathcal{D}$ atualizado, tal que $\mathcal{R}$ é GAC direcionada.

```
1: para i ← n .. 1 faça
2:     para todo R_C ∈ C tal que x_i ∈ C e existe x_j ∈ C tal que j < i faça
3:         seja x_{p_j} a variável em C = {x_{p_1}, ..., x_{p_k}} com o menor índice em b
4:         revisaGAC(R_C, (D_{p_1}, ..., D_{p_k}), x_{p_j})
5:     fim para
6: fim para
```

Teorema 3.12. `GAC_direcionada`$(\mathcal{R}, b)$ *consome tempo* $O(mkd^k)$*, onde* m *é o número de restrições de* $\mathcal{R}$*,* k *é a maior aridade de restrições em* $\mathcal{R}$ *e* d *é o tamanho do maior domínio na rede.*

Demonstração. Cada restrição R_C é processada no máximo $k - 1$ vezes, uma para cada uma de suas variáveis que não têm o menor índice em b. Cada processamento, por sua vez, é composto por uma chamada da função `revisaGAC`, que consome tempo $O(d^k)$. Como existem m restrições, então `GAC_direcionada`$(\mathcal{R}, b)$ consome tempo $O(mkd^k)$. □

Exemplo 3.20. O Exemplo 3.4 mostrou o processo de alcançar GAC total na rede $\mathcal{R}_{PC}$ da instância de palavras cruzadas da Figura 2.5. Considerando apenas a versão direcionada dessa

consistência, sob a ordenação de variáveis $b = (x_1, \ldots, x_{21})$, apenas os domínios D_1, D_2, D_3, D_9 e D_{17} são reduzidos, pois `revisaGAC` é executado apenas para a variável de cada restrição com menor índice em b. Por exemplo, ao processar x_{21}, na primeira iteração do laço 1, o Algoritmo 17 executa `revisaGAC` $(R_{C_2}, (D_2, D_6, D_8, D_{14}, D_{16}, D_{18}, D_{21}), x_2)$, inferindo $D_2 \leftarrow \{\mathrm{A}, \mathrm{S}, \mathrm{T}\}$. $\triangle$

No cenário contínuo, consistência direcionada evita a possibilidade de processamento infinito. Em particular, para instâncias NCSP ternárias o procedimento `revisaGAC` do Algoritmo 17 é substituído pelo contrator $D_j \leftarrow D_j \cap (D_i \circ D_k)$, onde D_j é o domínio da variável x_j de menor índice em b.

Teorema 3.13. *Dada uma rede ternária $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, onde $\forall (R_C \equiv x_1 = x_2 \circ x_3) \in C : \circ \in \{+, -, \cdot, /, \hat{}, \sqrt{}\}$ e, se $\circ \in \{\hat{}, \sqrt{}\}$, então D_3 contém um único valor inteiro, se $\mathcal{R}$ puder ser ordenada de forma que toda restrição compartilhe apenas uma variável com restrições antecessoras, então existe uma ordenação de variáveis b tal que* `GAC_direcionada`$(\mathcal{R}, b)$ *consome tempo $O(l^{m+1}(m+1)\log l)$, onde l é o maior número de intervalos que compõem os multi-intervalos da rede e m é o seu número de restrições.*

Demonstração. Seja d uma ordenação de restrições de $\mathcal{R}$ tal que toda restrição R_{C_i} compartilha apenas uma variável x_i com restrições antecessoras em d. Seja b uma ordenação de variáveis definida de forma que toda variável x_i anteceda as demais em C_i. Ao executar `GAC_direcionada`$(\mathcal{R}, b)$, cada restrição é processada duas vezes, mas apenas uma de suas variáveis pode ter sido processada anteriormente. Assim, cada processamento consta de duas aplicações do contrator $D_j \leftarrow D_j \cap (D_i \circ D_k)$, que resulta, no pior caso, em um multi-intervalo com $l_i \cdot l$ intervalos e consome tempo $O(l_i \cdot l \cdot \log(l_i \cdot l))$ para ser ordenado, onde l_i é o número de intervalos da variável que já havia sido processada. É fácil ver que a cada restrição processada o número de intervalos no domínio da sua variável de menor índice é multiplicado por um fator l. Logo, na última restrição processada, o domínio D_1 é composto por no máximo l^{m+1} intervalos e consome tempo $O(l^{m+1}(m+1)\log l)$ para ser ordenado. Pelas definições das operações básicas apresentadas na Seção 2.3.2, as operações intervalares $+$, $-$, $\cdot$, $/$, $\hat{}$ e $\sqrt{}$ são computadas em tempo constante. $\square$

Na Seção 4.2.1, será mostrado que redes ternárias podem ser ordenadas de acordo com a condição enunciada no Teorema 3.13 se o seu hipergrafo de restrições é Berge-acíclico. Nesse caso, as ordenações d e b são obtidas por uma ordenação topológica do hipergrafo.

Por fim, o conceito de consistência de arco relacional direcionada é naturalmente estendido ao de k-consistência relacional direcionada.

Definição 3.22 (**k-consistência relacional direcionada**)**.** Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é dita k-consistente relacional direcionada, em relação a uma ordenação de variáveis $b = (x_1, \ldots, x_n)$, se, e somente se, para todo conjunto de k restrições $\{R_{C_1}, \ldots, R_{C_k}\} \subseteq C$ e para toda valoração I consistente com $\mathcal{R}$, das variáveis em $\bigcup_{i=1}^{k} C_i \setminus \{x_j\}$, onde x_j é a variável do conjunto $\bigcap_{i=1}^{k} C_i$

com o maior índice em b, existe um valor $a_j \in D_j$ tal que $I \cup \{\langle x_j = a_j \rangle\}$ satisfaz $\{R_{C_1}, \ldots, R_{C_k}\}$ simultaneamente. Uma rede $\mathcal{R}$ é dita fortemente k-consistente relacional direcionada se é j-consistente relacional direcionada, para todo $1 \leq j \leq k$.

3.5.2 Condições suficientes para busca sem retrocesso

Freuder (1982) estabeleceu uma relação entre a estrutura do hipergrafo de restrições e o nível de k-consistência forte que garante uma busca sem retrocesso em redes binárias. Para isso, Freuder propôs um parâmetro que mede o nível de ciclicidade do grafo de restrições, chamado de largura.

Definição 3.23 (Largura).

- Dado um grafo $G = (V, E)$ e uma ordenação $b = (x_1, \ldots, x_n)$ dos vértices de G, o conjunto de pais do vértice x_i em b é o conjunto $\{x_j \in V \mid \{x_i, x_j\} \in E \text{ e } j < i\}$;
- A largura de um vértice x_i em b é a cardinalidade do conjunto de pais de x_i;
- A largura da ordenação b é a maior largura dentre todos os vértices em b;
- A largura do grafo G é a menor largura dentre todas as ordenações de vértices de G.

Freuder propôs um algoritmo linear para encontrar a ordenação de largura mínima de um grafo. Basicamente, constrói-se a ordenação do último para o primeiro elemento, escolhendo em cada iteração o vértice de grau mínimo no subgrafo induzido por vértices ainda não escolhidos.

Teorema 3.14. *Se o grafo de restrições de uma rede binária $\mathcal{R}$ possui uma ordenação de vértices b com largura $k-1$ e $\mathcal{R}$ é fortemente k-consistente direcionada em relação a b, então $\mathcal{R}$ é livre de retrocesso em b.*

Demonstração. Seja $b = (x_1, \ldots, x_n)$ uma ordenação de vértices do grafo de restrições de $\mathcal{R}$ com largura $k-1$. Toda variável x_i compartilha restrições com no máximo $k-1$ variáveis antecessoras em b, isto é, existem no máximo $k-1$ restrições $R_{\{x_i, x_{i_1}\}}, \ldots, R_{\{x_i, x_{i_l}\}}$, $l \leq k-1$, $i_1, \ldots, i_l < i$. Logo, uma valoração consistente de $(x_1, \ldots, x_{i-1})$ pode ser estendida a x_i se toda valoração consistente de $\{x_{i_1}, \ldots, x_{i_l}\}$ puder ser estendida a x_i, o que é satisfeito por k-consistência direcionada forte. □

O Teorema 3.14 é significativo, pois estabelece uma complexidade de instâncias de CSPs de acordo com a estrutura do hipergrafo de restrições. Dessa forma, quanto menor a largura do grafo, menor o nível de consistência que garante uma busca sem retrocesso. Na prática, no entanto, alcançar k-consistência forte exige a adição de novas restrições na rede, alterando a sua estrutura e, consequentemente, a largura do seu grafo de restrições. Dessa forma, ao tentar alcançar as condições suficientes para uma busca livre de retrocesso, altera-se essas condições. Dechter e Pearl (1987b) propuseram um método para alcançar k-consistência direcionada forte de

maneira dinâmica, adaptando o nível de k-consistência conforme novas restrições são adicionadas à rede.

Um caso particular do resultado de Freuder eficientemente aplicado na prática é quando o grafo de restrições tem largura 1, no qual a consistência de arco direcionada (2-consistência) é suficiente para encontrar solução sem retrocesso, pois alcançar essa consistência não altera a estrutura do problema, apenas reduz domínios de variáveis. Mackworth e Freuder (1985) analisaram o algoritmo original de Waltz (consistência de arco) e estabeleceram que este é suficiente para resolver instâncias acíclicas. De fato, toda árvore tem largura 1, pois é possível ordenar os seus vértices de forma que cada vértice possua apenas um pai. De forma semelhante, Faltings (1998) mostrou que GAC é suficiente para encontrar solução sem retrocesso de problemas numéricos acíclicos, isto é, cujo hipergrafo de restrições é Berge-acíclico. Uma revisão acerca de GAC como condição suficiente para resolução de instâncias CSP sem retrocesso pode ser encontrada em Cohen e Jeavons (2017).

Exemplo 3.21. Seja $\mathcal{R}_{PC}^{dual}$ a rede dual da instância de palavras-cruzadas da Figura 2.5, apresentada no Exemplo 3.1, e a ordenação de variáveis duais $b = (y_1, y_3, y_4, y_2, y_5)$. Executando $\texttt{DAC}(\mathcal{R}_{PC}^{dual}, b)$ (Algoritmo 16), obtém-se as seguintes reduções de domínios:

$$y_5 \Rightarrow \texttt{revisa}(S_{\{y_2,y_5\}}, D_{y_2}, D_{y_5}, y_2) \Rightarrow D_2 \leftarrow \{\text{SIDIOUS, TYRANUS}\}$$
$$y_2 \Rightarrow \texttt{revisa}(S_{\{y_2,y_4\}}, D_{y_2}, D_{y_4}, y_4) \Rightarrow D_4 \leftarrow \{\text{KENOBI}\}$$
$$y_4 \Rightarrow \texttt{revisa}(S_{\{y_1,y_4\}}, D_{y_1}, D_{y_4}, y_1) \Rightarrow D_1 \leftarrow \{\text{VADER}\}$$
$$y_3 \Rightarrow \texttt{revisa}(S_{\{y_1,y_3\}}, D_{y_1}, D_{y_3}, y_1) \Rightarrow D_1 \leftarrow \{\text{VADER}\}$$

É fácil ver que b tem largura 1 (na verdade, o grafo de $\mathcal{R}_{PC}^{dual}$ é acíclico, conforme mostra a Figura 2.6) e, portanto, a rede obtida é livre de retrocesso nesta ordenação. △

Através do conceito de hiper-k-consistência, Jégou (1993) propôs uma extensão dos resultados de Freuder ao cenário não binário, estabelecendo uma relação entre a largura do hipergrafo de restrições e o nível de hiper-k-consistência que garante uma busca sem retrocesso.

Definição 3.24 (**Largura de hipergrafo**)**.**

- Dado um hipergrafo $\mathcal{H} = (V, E)$ e uma ordenação $d = (C_1, \ldots, C_m)$ das arestas de $\mathcal{H}$, o conjunto de interseções maximais da aresta C_i em d é o conjunto $\{C_j \mid j < i \text{ e } \nexists k < i \text{ tal que } C_i \cap C_k \supset C_i \cap C_j\}$;
- A largura de uma aresta D_i em d é a cardinalidade do conjunto de interseções maximais de C_i;
- A largura da ordenação d é a maior largura dentre todos as arestas em d;

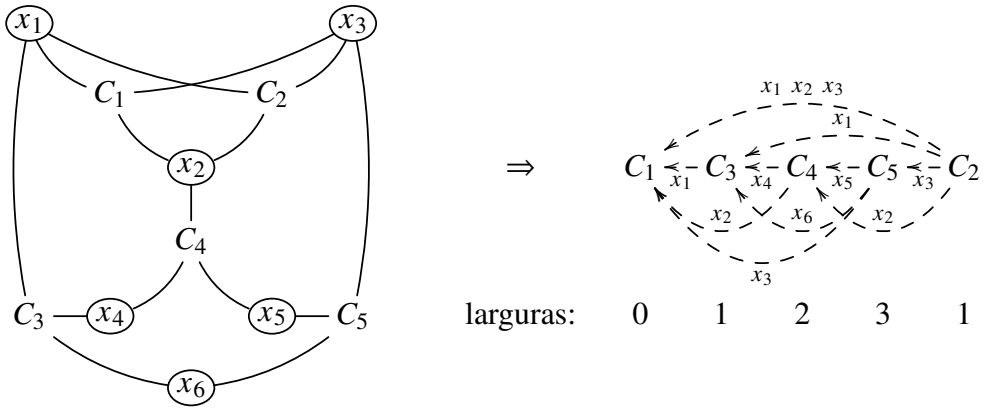


Figura 3.2: Um hipergrafo $\mathcal{H}$ com as interseções de cada aresta em relação à ordenação $d = (C_1, C_3, C_4, C_5, C_2)$. A aresta C_2 possui quatro interseções, mas apenas uma é maximal: $\{x_1, x_2, x_3\}$.

- A largura do hipergrafo $\mathcal{H}$ é a menor largura dentre todas as ordenações de arestas de $\mathcal{H}$.

Largura de hipergrafo também é um parâmetro de nível de ciclicidade, de forma que hipergrafos com largura 1 são α-acíclicos, e vice-versa (Jégou, 1993).

Exemplo 3.22. (Jégou, 1993). Seja a rede $\mathcal{R}$ cujo hipergrafo de restrições está ilustrado na Figura 3.2 e a ordenação $d = (C_1, C_3, C_4, C_5, C_2)$. As interseções de cada aresta C_i em relação a d também estão representados na figura através das arestas tracejadas direcionadas. Por exemplo, a aresta C_2 possui 4 interseções nesta ordenação: $C_2 \cap C_5 = \{x_3\}$, $C_2 \cap C_4 = \{x_2\}$, $C_2 \cap C_3 = \{x_1\}$ e $C_2 \cap C_1 = \{x_1, x_2, x_3\}$. No entanto, apenas a interseção $C_2 \cap C_1$ é maximal (as demais estão contidas nesta); logo, C_2 tem largura de hipergrafo 1 em d. A largura da ordenação d é 3 e a largura do hipergrafo de restrições de $\mathcal{R}$ também é 3, pois não existe ordenação de largura menor neste hipergrafo.

Teorema 3.15. (Jégou, 1993). *Se o hipergrafo de restrições de uma rede $\mathcal{R}$ possui largura $k-1$ e $\mathcal{R}$ é fortemente hiper-k-consistente, então existe uma ordenação de variáveis b na qual $\mathcal{R}$ é livre de retrocesso.*

Corolário 3.1. (Janssen et al., 1989). *Se o hipergrafo de restrições de uma rede $\mathcal{R}$ é α-acíclico e $\mathcal{R}$ é consistente por emparelhamento, então existe uma ordenação de variáveis b na qual $\mathcal{R}$ é livre de retrocesso.*

Como nos resultados de Freuder, alcançar hiper-k-consistência altera o hipergrafo de restrições, sendo, em geral, de difícil aplicação prática.

Outras relações entre propriedades da rede e o nível de consistência que garante uma solução livre de retrocesso foram propostas na literatura. Em particular, algumas relações interessantes envolvendo consistência relacional foram propostas em diversos trabalhos de Dechter, van Beek, Rish, entre outros:

- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ fortemente k-consistente relacional direcionada em relação à ordenação de variáveis b, na qual $|D_i| \leq k$, para todo $D_i \in \mathcal{R}$, é livre de retrocesso em b (Dechter, 1992). Em particular, para domínios binários (por exemplo, o problema da satisfazibilidade proposicional) consistência de arco (2-consistência) relacional direcionada é suficiente para encontrar uma solução sem a necessidade de retrocesso. Dechter e Rish (1994) observaram que ao especializar o algoritmo que alcança a consistência de arco relacional para o cenário proposicional, obtém-se um método equivalente ao algoritmo original de Davis-Putnam para satisfazibilidade proposicional[7] (Davis e Putnam, 1960).

- Uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ 3-consistente relacional, na qual existe uma ordenação $(D_1, \ldots, D_n)$ dos domínios em $\mathcal{D}$ tal que todas as restrições de $\mathcal{R}$ são linha-convexas[8], é globalmente consistente (van Beek e Dechter, 1995).

- O algoritmo para alcançar consistência de arco relacional direcionada especializado com um método de eliminação linear é suficiente para gerar uma representação equivalente e livre de retrocesso de uma rede de inequações lineares (Dechter, 2003). De fato, este método é equivalente à eliminação de Fourier (1824).

- Uma rede numérica ternária convexa $(3, 2)$-consistente relacional[9] é livre de retrocesso (Sam-Haroud e Faltings, 1996).

3.5.3 Técnicas de decomposição

Dechter e Pearl (1987a) propuseram o método de **corte de ciclo** motivados pelo fato de que grafos de restrições acíclicos são resolvidos de forma eficiente, apenas aplicando a consistência de arco direcionada. Um conjunto de corte de ciclo é um conjunto de variáveis que, ao serem removidas da rede, juntamente com as restrições às quais pertencem, resultam em um grafo de restrições acíclico. Alternativamente, ao valorar estas variáveis e propagá-las a seus vizinhos, o restante da busca não afetará (ou será afetada por) estas variáveis.

As variáveis de um conjunto de corte de ciclo podem ser valoradas de diferentes maneiras e, para cada valoração, pode ou não existir uma extensão consistente para toda a rede. Dessa

[7] Como o algoritmo de Davis-Putnam original adiciona um número exponencial de novas cláusulas à rede, Davis et al. (1962) propuseram o conhecido método DPLL, que faz uso de uma busca com retrocesso para evitar o consumo de memória exponencial.

[8] Uma restrição binária $R_{\{x_i,x_j\}}$ é linha-convexa se, e somente se, ao ser representada por uma matriz $M_{|D_i|\times|D_j|}$, onde $M_{kl} = 1$ se $(a_k, b_l) \in R_{\{x_i,x_j\}}$ e $M_{kl} = 0$ caso contrário, não existe 0 entre dois 1 consecutivos em cada linha ou coluna de M. Uma extensão dessa definição para restrições k-árias pode ser encontrada em Dechter (2003).

[9] Sam-Haroud e Faltings (1996) definem $(3, 2)$-consistência relacional para restrições ternárias da seguinte forma: para todo conjunto de 3 restrições com 2 variáveis em comum, qualquer valoração consistente das variáveis restantes é consistentemente estendida a essas 2 variáveis. Em Dechter e van Beek (1997), (j, k)-consistência relacional é proposta da seguinte forma: para todo conjunto de k restrições e todo conjunto de j variáveis nessas restrições, toda valoração consistentes dessas j variáveis pode ser estendida às demais variáveis do conjunto satisfazendo as k restrições simultaneamente. Em particular, $(1, 1)$-consistência relacional e GAC são equivalentes.

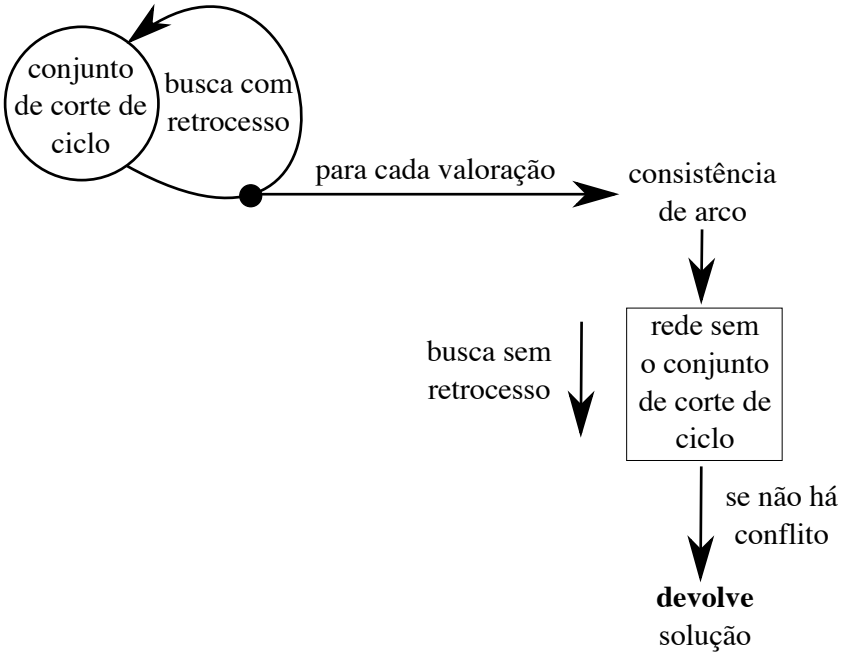


Figura 3.3: Esquema do método de corte de ciclo.

forma, o método proposto por Dechter e Pearl (1987a) consiste em efetuar uma busca com retrocesso nas variáveis do conjunto de corte de ciclo e, para cada valoração obtida, gerar uma rede acíclica sem estas variáveis e tentar a extensão total da valoração utilizando um algoritmo de consistência de arco direcionado. Se uma valoração total é obtida, então encontrou-se uma solução da rede. Caso contrário, deve-se voltar ao conjunto de corte de ciclo e obter uma nova valoração parcialmente consistente. A Figura 3.3 ilustra esse método.

Exemplo 3.23. Seja uma rede binária $\mathcal{R}$ composta pelas variáveis proposicionais $x_1, \ldots, x_6$ e as cláusulas

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_4) \wedge (x_3 \vee x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_5 \vee x_6)$$

A Figura 3.4 apresenta o grafo de restrições desta rede. Nota-se que ao remover o vértice x_2 o grafo torna-se acíclico e, portanto, $\{x_2\}$ é um conjunto de corte de ciclo em $\mathcal{R}$. Assim, valora-se, por exemplo, $\langle x_2 = \texttt{falso} \rangle$ e propaga este valor às variáveis vizinhas através das cláusulas $(x_1 \vee x_2)$, $(\neg x_2 \vee x_3)$ e $(x_2 \vee \neg x_5)$, inferindo $D_1 \leftarrow \{\texttt{verdadeiro}\}$ e $D_5 \leftarrow \{\texttt{falso}\}$. A rede resultante removendo-se estas cláusulas é acíclica e, portanto, é livre de retrocesso ao alcançar-se a consistência de arco. Nesse caso, no entanto, a consistência infere $D_3 \leftarrow \emptyset$, pelas cláusulas $(\neg x_1 \vee \neg x_3)$ e $(x_3 \vee x_5)$. Volta-se então ao conjunto de corte de ciclo e efetua-se um retrocesso na valoração de x_2, obtendo uma nova valoração parcial $\langle x_2 = \texttt{verdadeiro} \rangle$. Propaga-se essa valoração às variáveis vizinhas de maneira análoga e o algoritmo de consistência de arco

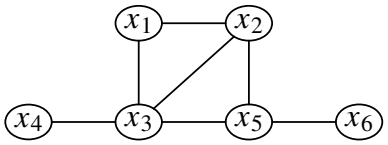


Figura 3.4: Grafo de restrições da rede $\mathcal{R}$ do Exemplo 3.23.

direcionada é novamente executado. Dessa vez, nenhum domínio vazio é gerado e uma busca sem retrocesso valora toda a rede. △

De maneira geral, o método de corte de ciclo consome tempo de processamento $O(md^{c+2})$, onde m é o número de restrições, d é o tamanho do maior domínio de variável na rede e c é a cardinalidade do conjunto de corte de ciclo, pois, no pior caso, todas as d^c valorações do conjunto de corte de ciclo são parcialmente consistentes e, para cada uma delas, o método de consistência de arco direcionada é executado, consumindo tempo $O(md^2)$. Uma rede pode ter vários conjuntos de corte de ciclo distintos, mas escolher o conjunto mínimo, isto é, de menor aridade, é um problema $\mathcal{NP}$-difícil.

No cenário não binário, o conceito de aciclicidade depende do nível de consistência que está sendo aplicado. Por exemplo, GAC direcionada é suficiente para resolver redes Berge-acíclicas. Nesse caso, o conjunto de corte de ciclo deve conter as variáveis que, ao serem removidas, tornam o hipergrafo Berge-acíclico.

Outra técnica de resolução de CSPs baseada no resultado de que redes acíclicas são resolvidas de forma eficiente é a chamada **decomposição em árvore**, também chamada de decomposição arbórea (Dechter e Pearl, 1989).

Em geral, devido à equivalência com a rede dual, uma rede $\mathcal{R}$ pode ser resolvida em tempo polinomial se o seu grafo dual é acíclico. Mais especificamente, mesmo que este grafo possua ciclos, é possível, em alguns casos, torná-lo acíclico removendo arestas redundantes, pois na rede dual toda restrição impõe uma igualdade entre determinados elementos de tuplas de variáveis duais.

Exemplo 3.24. Seja $\mathcal{R}$ uma rede com as variáveis x_1, x_2, x_3, x_4 e x_5, e as restrições R_{C_1}, R_{C_2} e R_{C_3}, onde $C_1 = \{x_1, x_2, x_3, x_4\}$, $C_2 = \{x_3, x_4, x_5\}$ e $C_3 = \{x_3, x_5\}$. Esta rede possui uma rede dual $\mathcal{R}^{dual}$ com as variáveis y_1, y_2 e y_3, uma para cada restrição de $\mathcal{R}$, e as seguintes restrições duais:

$$S_{\{y_1,y_2\}} \equiv y_1[x_3] = y_2[x_3] \text{ e } y_1[x_4] = y_2[x_4]$$
$$S_{\{y_1,y_3\}} \equiv y_1[x_3] = y_3[x_3]$$
$$S_{\{y_2,y_3\}} \equiv y_2[x_3] = y_3[x_3] \text{ e } y_2[x_5] = y_3[x_5]$$

Nesse caso, a restrição $S_{\{y_1,y_3\}}$ é redundante, pois a condição $y_1[x_3] = y_3[x_3]$ já está garantida pelas condições $y_1[x_3] = y_2[x_3]$, da restrição $S_{\{y_1,y_2\}}$, e $y_2[x_3] = y_3[x_3]$, da restrição $S_{\{y_2,y_3\}}$.

Enquanto o grafo de $\mathcal{R}^{dual}$ é cíclico, o grafo obtido removendo-se $S_{\{y_1,y_3\}}$ é acíclico e, portanto, pode ser resolvido eficientemente. △

Na prática, essa abordagem é equivalente ao conceito de árvore de junção, que caracteriza hipergrafos α-acíclicos (Definição 2.10). Assim, redes cujos hipergrafos são α-acíclicos são resolvidos em tempo polinomial no tamanho da instância, aplicando-se a consistência de arco na árvore de junção (ou ainda, aplicando a consistência por emparelhamento no hipergrafo original, conforme apresentado na Seção 3.5.2).

Por outro lado, instâncias que não são α-acíclicas podem ser decompostas em subproblemas que, juntos, constituem uma árvore de junção. Esse procedimento é chamado de decomposição em árvore, e foi proposto por Dechter e Pearl (1989), baseados em resultados prévios da comunidade de banco de dados relacionais que estabelecem a relação entre bancos acíclicos, isto é, que possuem uma árvore de junção, e processamento de consultas em tempo polinomial (Beeri et al., 1983).

Informalmente, a rede é decomposta em subconjuntos de restrições que caracterizam, cada um, uma rede independente e mais fácil de resolver. Se algum desses subproblemas não tem solução, então a rede original também não tem e a busca pode terminar. Do contrário, cada subproblema define uma nova variável cujo domínio é o conjunto de todas as soluções desta subrede. Para que essas soluções sejam combinadas em uma solução total da rede original é preciso resolver uma nova instância CSP constituída de restrições de igualdade entre subproblemas que compartilham variáveis originais, de maneira análoga ao que ocorre no grafo dual. Da mesma forma, restrições redundantes não precisam ser adicionadas nesta nova instância.

Exemplo 3.25. A Figura 3.5 apresenta uma decomposição em árvore da rede do Exemplo 3.23, composta por 4 subproblemas. A aresta tracejada é redundante e pode ser removida do grafo. Sendo z_i a variável que representa o subproblema i (de cima para baixo, e da esquerda para a direita), o domínio D_i é composto por todas as soluções de cada subproblema, resultando em uma rede de restrições acíclica com domínios

$$
\begin{aligned}
D_{z_1} &= \{(\texttt{falso}, \texttt{verdadeiro}, \texttt{verdadeiro}), (\texttt{verdadeiro}, \texttt{falso}, \texttt{falso})\} \\
D_{z_2} &= \{(\texttt{falso}, \texttt{verdadeiro}, \texttt{falso}), (\texttt{verdadeiro}, \texttt{verdadeiro}, \texttt{falso}), \\
&\qquad (\texttt{verdadeiro}, \texttt{verdadeiro}, \texttt{verdadeiro})\} \\
D_{z_3} &= \{(\texttt{falso}, \texttt{falso}), (\texttt{falso}, \texttt{verdadeiro}), (\texttt{verdadeiro}, \texttt{verdadeiro})\} \\
D_{z_4} &= \{(\texttt{falso}, \texttt{verdadeiro}), (\texttt{verdadeiro}, \texttt{falso}), (\texttt{verdadeiro}, \texttt{verdadeiro})\}
\end{aligned}
$$

△

Definição 3.25 (**Decomposição em árvore**)**.** Uma decomposição em árvore (ou decomposição arbórea) da rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ é uma tupla (T, χ, ψ) onde $T = (V, E)$ é uma árvore e χ e ψ são duas funções que associam cada vértice $v \in V$ a dois conjuntos $\chi(v) \subseteq \mathcal{X}$ e $\psi(v) \subseteq C$ satisfazendo as seguintes condições:

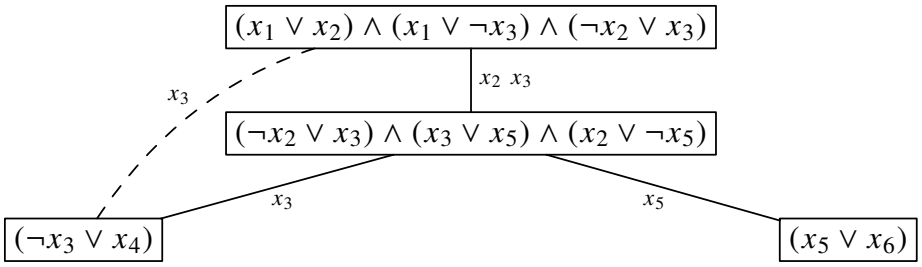


Figura 3.5: Uma decomposição em árvore da rede $\mathcal{R}$ do Exemplo 3.23. A aresta tracejada é redundante e não pertence à decomposição.

1. para todo $R_{C_i} \in C$ existe pelo menos um $v \in V$ tal que $R_{C_i} \in \psi(v)$ e $C_i \subseteq \chi(v)$;
2. para todo $x_i \in \mathcal{X}$ o conjunto $\{v \in V \mid x_i \in \chi(v)\}$ induz uma subárvore de T conexa.

A condição 2 garante que todos os vértices da decomposição associados a restrições que compartilham variáveis estão conectados em T, representando as restrições de igualdade entre os subproblemas decompostos. É importante notar que essa condição é definida em relação à função χ e não à função ψ. Em redes binárias, a função ψ pode ser omitida, adicionando-se a condição de que toda aresta $\{x_1, x_2\}$ do grafo de restrições está contida em $\chi(v)$, para algum $v \in V$; esta definição é equivalente à de árvore de junção da rede binária.

Exemplo 3.26. A decomposição em árvore apresentada na Figura 3.5 é dada por (T, χ, ψ), onde

$$
\begin{aligned}
T &= (\{z_1, z_2, z_3, z_4\}, \{\{z_1, z_2\}, \{z_2, z_3\}, \{z_2, z_4\}\}) \\
\chi(z_1) &= \{x_1, x_2, x_3\} & \psi(z_1) &= \{(x_1 \vee x_2), (x_1 \vee \neg x_3), (\neg x_2 \vee x_3)\} \\
\chi(z_2) &= \{x_2, x_3, x_5\} & \psi(z_2) &= \{(\neg x_2 \vee x_3), (x_3 \vee x_5), (x_2 \vee \neg x_5)\} \\
\chi(z_3) &= \{x_3, x_4\} & \psi(z_3) &= \{(\neg x_3 \vee x_4)\} \\
\chi(z_4) &= \{x_5, x_6\} & \psi(z_4) &= \{(x_5 \vee x_6)\}
\end{aligned}
$$

△

Uma rede pode ter várias decomposições em árvore distintas. Em particular, define-se dois principais parâmetros de complexidade de uma decomposição: largura de árvore (também chamada de largura arbórea) e largura de hiperárvore.

Definição 3.26 (Largura de árvore). (Robertson e Seymour, 1986) A largura de árvore de uma decomposição (T, χ, ψ), tal que $T = (V, E)$, é dada por $\max_{v \in V} |\chi(v)|$. A largura de árvore de um hipergrafo $\mathcal{H}$ é a menor largura de árvore dentre todas as suas decomposições.

Definição 3.27 (Largura de hiperárvore). (Gottlob et al., 2000) A largura de hiperárvore de uma decomposição (T, χ, ψ), tal que $T = (V, E)$, é dada por $\max_{v \in V} |\psi(v)|$. A largura de hiperárvore de um hipergrafo $\mathcal{H}$ é a menor largura de hiperárvore dentre todas as suas decomposições.

Exemplo 3.27. A decomposição em árvore apresentada na Figura 3.5 possui largura de árvore 3 e largura de hiperárvore 3. △

Em redes binárias, a largura de árvore é um parâmetro de generalização de ciclicidade do grafo: um grafo é acíclico se, e somente se, possui largura de árvore 1. Além disso, determinar a largura de árvore de um grafo G é um problema $\mathcal{NP}$-completo, mas decidir se essa largura é no máximo uma constante k consome tempo de processamento linear (Arnborg e Proskurowski, 1989). No caso geral, compilar uma rede de restrições em uma representação acíclica demanda, no pior caso, tempo de processamento exponencial na largura de árvore do grafo de restrições. Por outro lado, o conjunto de instâncias com largura de árvore no máximo uma constante k define uma subclasse de problemas solúveis em tempo polinomial no tamanho da instância (exponencial em k).

Em redes k-árias, a largura de hiperárvore caracteriza melhor a estrutura da decomposição do que a largura de árvore. Um hipergrafo é α-acíclico se, e somente se, possui largura de hiperárvore 1 (Gottlob et al., 2003). No entanto, diferentemente da largura de árvore, decidir se um hipergrafo possui uma largura de hiperárvore no máximo uma constante k é um problema difícil (Gottlob et al., 2005). De forma análoga, a classe de problemas com largura de hiperárvore no máximo k é resolvida em tempo polinomial.

Uma visão geral de métodos que computam decomposições em árvore pode ser encontrada no capítulo 9 de Dechter (2003) e em Gottlob et al. (2014); este também apresenta uma série de aplicações para os conceitos de largura de hiperárvore. Uma revisão de métodos de decomposição de CSPs é apresentada em Gottlob et al. (2000).

3.6 Conclusão

A programação por restrições é uma poderosa ferramenta para abordar diversos problemas do mundo real. Ao representar matematicamente as restrições de um problema, por meio de uma rede de restrições, obtém-se um problema de satisfação de restrições ou de otimização global, sobre domínios contínuos ou discretos. Embora este problema seja $\mathcal{NP}$-difícil, técnicas de processamento de restrições de propósito geral são comumente utilizadas para abordá-lo, combinando busca e inferência.

Neste capítulo, apresentou-se uma revisão das principais técnicas de processamento de restrições, sobretudo técnicas de inferência que alcançam algum nível de consistência na rede de restrições. No cenário finito, elencou-se as consistências de arco, de caminho, k-consistência, consistência de arco generalizada, consistência relacional, consistência por emparelhamento e hiper-k-consistência. Discutiu-se que enquanto algumas dessas consistências são vastamente utilizadas na prática, outras, como a k-consistência relacional por exemplo, demandam tempo e espaço exponenciais para serem alcançadas. Argumentou-se também que a consistência de arco generalizada (GAC) e a consistência de arco relacional (RAC) são consistências complementares.

No cenário contínuo, a consistência de arco generalizada é aplicada utilizando o conceito de função de projeção, que é bem definida para restrições ternárias. A consistência de envoltória é uma aproximação de GAC que visa manter a representação compacta e completa. As consistências de caixa e kB-consistência também foram apresentadas.

Este capítulo também introduziu uma revisão de métodos intervalares aplicados a NCSPs e NCOPs, citando os principais trabalhos e técnicas da área. Por fim, a consistência direcionada foi apresentada como forma mais simples de alcançar algum nível de consistência e as principais relações entre a estrutura de uma rede e o nível de consistência que garante uma solução livre de retrocesso foram discutidas.

No Capítulo 4, será estabelecida uma condição suficiente para otimização global sem retrocesso. Como nos trabalhos de Freuder (1982) e Jégou (1993), introduzidos neste capítulo, essa condição será definida sobre a relação entre estrutura do hipergrafo de restrições e o nível de consistência satisfeito pela rede; a saber, a estrutura do hipergrafo será associada ao nível de consistência relacional direcionada que garante uma busca pelo mínimo global sem encontrar conflitos.

4 CONSISTÊNCIA RELACIONAL APLICADA À OTIMIZAÇÃO GLOBAL

Nas últimas décadas, diversas relações entre a estrutura da rede de restrições e o nível de consistência que garante uma busca livre de retrocesso foram estabelecidas no cenário de CSP. Freuder (1982) estabeleceu que redes binárias com largura k são resolvidas sem retrocesso se alcançada k-consistência forte; Jégou (1993) estendeu esse resultado para o cenário não binário, mostrando que hiper-k-consistência garante uma busca livre de retrocesso em redes com largura de hipergrafo k; Faltings (1998) mostrou que GAC é suficiente para encontrar solução sem retrocesso em problemas numéricos cujo hipergrafo de restrições é Berge-acíclico. Em alguns casos, alcançar o nível de consistência que garante solução livre de retrocesso é fácil e essas relações identificam, portanto, classes polinomiais de problemas de restrições. No caso geral, no entanto, alcançar essa consistência é um problema $\mathcal{NP}$-difícil; nesse caso, a relação entre estrutura e consistência define a complexidade de subclasses de CSPs.

Neste capítulo, efetua-se uma análise semelhante no contexto de problemas de otimização global, estabelecendo uma relação entre a estrutura da instância e o nível de consistência relacional que garante uma busca pelo ótimo global em tempo polinomial; embora, no pior caso, alcançar essa consistência demande tempo de processamento e espaço de memória exponenciais no tamanho da instância, isto é, não se conhece um algoritmo eficiente para alcançá-la. Até onde essa pesquisa pôde alcançar, não há conhecimento de outro trabalho que estabeleça essa relação[1].

Considera-se o problema de otimização global (NCOP) na sua forma mais geral, como introduzido na Definição 2.20. Dessa forma, contempla-se problemas de programação não linear, programação linear, programação inteira e otimização combinatória. No entanto, o foco principal dos algoritmos que aplicam os resultados aqui apresentados são os problemas contínuos, utilizando a análise intervalar como ferramenta de processamento de domínios de variáveis.

O nível de consistência relacional como condição suficiente para otimização global sem retrocesso está relacionado com a estrutura do hipergrafo de restrições do problema, adicionada a função objetivo na forma de uma nova restrição $x_1 = f(x)$. Mais especificamente, este hipergrafo deve apresentar uma decomposição particular que é definida nesta tese sob o nome de **decomposição Epífita**. Para a introdução desta decomposição, representa-se a instância por uma rede ternária equivalente, conforme apresentado na Seção 2.2.4. Ao final do capítulo,

[1]Resultados preliminares podem ser encontrados na dissertação de mestrado que antecede essa pesquisa (Derenievicz, 2014).

estende-se o conceito de decomposição Epífita para englobar redes k-árias sem a necessidade de transformação ternária.

Este capítulo apresenta as principais contribuições desta tese e está organizado da seguinte maneira:

4.1. **Codificação ternária de problemas de otimização**: apresenta-se uma forma de representação de instâncias NCOP como redes ternárias;

4.2. **Otimização global sem retrocesso**: propõe-se um método de otimização global sem retrocesso em redes acíclicas e classifica-se instâncias NCOP em três classes principais: acíclicas, parcialmente acíclicas e cíclicas;

4.3. **Decomposições Epífitas**: propõe-se uma decomposição de hipergrafos que caracteriza uma classe de instâncias parcialmente acíclicas, nas quais RAC direcionada é condição suficiente para otimização global sem retrocesso;

4.4. **Aproximando consistência de arco relacional direcionada**: propõe-se um método de ramificação e poda para alcançar RAC direcionada de forma aproximada;

4.5. **Decomposições k-Epífitas**: estende-se o conceito de decomposição Epífita a instâncias NCOP cíclicas, estabelecendo que k-consistência relacional direcionada forte é condição suficiente para otimização global sem retrocesso de instâncias que possuem pelo menos uma decomposição k-Epífita;

4.6. **Relação com larguras e outras abordagens**: analisa-se a relação entre decomposições k-Epífitas e outras decomposições e parâmetros de ciclicidade propostos na literatura, bem como a relação entre o método para aproximar RAC direcionada proposto e outros métodos de resolução de CSPs cíclicos.

4.1 Codificação ternária de problemas de otimização

Na Seção 2.2.4, foi mostrado que restrições numéricas k-árias fatoráveis podem ser transformadas em um conjunto equivalente de restrições ternárias de forma eficiente. Do mesmo modo, problemas de otimização da forma $\min f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$, sujeito a um conjunto de restrições S, onde $f(\mathbf{x})$ e S são fatoráveis, podem ser reduzidos a uma rede ternária com a adição de uma variável $x_1 = f(\mathbf{x})$ que representa a função objetivo.

Exemplo 4.1. Considerando novamente a instância *sample*, cuja codificação ternária do conjunto de restrições foi dado no Exemplo 3.16, a adição da função objetivo resulta na seguinte representação equivalente:

$$\min \quad x_1 \tag{4.1}$$

sujeito a

$$\begin{array}{lll} x_1 = x_2 + x_3 & y_5 = y_1 + y_2 & z_4 = 0.64/x_5 \\ j_1 = x_1 + x_4 & y_6 = y_5 + y_3 & z_5 = z_1 + z_2 \\ j_2 = j_1 + x_5 & y_7 = y_6 + y_4 & z_6 = z_5 + z_3 \\ y_1 = 4/x_2 & y_7 - c_1 = 0 & z_7 = z_6 + z_4 \\ y_2 = 2.25/x_3 & z_1 = 0.16/x_2 & z_7 - c_2 = 0 \\ y_3 = 1/x_4 & z_2 = 0.36/x_3 & \\ y_4 = 0.25/x_5 & z_3 = 0.64/x_4 & \end{array} \tag{4.2}$$

△

Definição 4.1 (**Raiz de uma rede**)**.** Dada uma rede de restrições $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ que codifica uma instâncias NCOP com função objetivo $f(\mathbf{x})$, uma variável $x_1 \in \mathcal{X}$ é dita raiz de $\mathcal{R}$ se ela representa $f(\mathbf{x})$ nesta rede, isto é, se existe um subconjunto de restrições $S \subseteq C$ equivalente à expressão $x_1 = f(\mathbf{x})$.

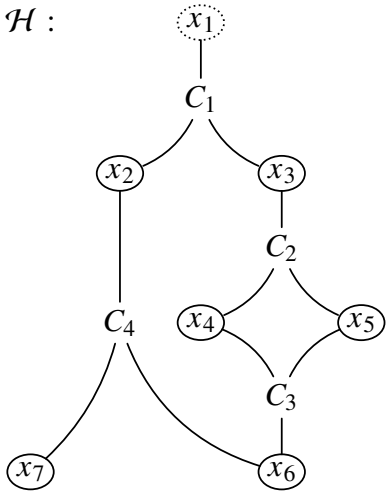
Definição 4.2 (**Solução ótima e mínimo global**)**.** Uma solução ótima de uma rede $\mathcal{R}$, em relação a uma variável raiz x_1, é uma solução de $\mathcal{R}$ que atribui a x_1 o menor valor z^* dentre todas as possíveis soluções dessa rede. Nesse caso, z^* é dito mínimo global de $\mathcal{R}$ em relação a x_1.

4.2 Otimização global sem retrocesso

Uma instância NCOP codificada como uma rede de restrições ternárias pode ser abordada utilizando-se de técnicas de consistência e retrocesso, como ocorre na resolução de CSPs. No entanto, não basta encontrar uma solução qualquer da rede de restrições, mas uma solução que seja ótima em relação à variável raiz, isto é, à variável que codifica a função objetivo.

Da mesma forma que algumas classes de CSPs podem ser resolvidas sem retrocesso se assegurado um determinado nível de consistência, deseja-se estabelecer propriedades que garantam uma busca pelo ótimo global sem encontrar conflitos. Intuitivamente, se tal busca ocorre seguindo uma ordem de variáveis que se inicia pela raiz x_1, então a atribuição inicial $\langle x_1 = \min D_1 \rangle$ garante que, se tal valoração puder ser estendida a todas as variáveis da rede, então a solução encontrada é ótima. Desta forma, levanta-se a seguinte questão: quais propriedades a rede deve manter para que essa valoração possa ser estendida à toda a rede sem a necessidade de retrocesso? Assim como nos problemas de satisfação de restrições, tais propriedades resumem-se à relação entre a estrutura do hipergrafo e o nível de consistência satisfeito pela rede.

Nesta seção, problemas de otimização são divididos em três classes principais, de acordo com a existência de ordenações específica de arestas no hipergrafo de restrições: redes



		grau de interseção
$d_1 = (C_1, C_2, C_3, C_4)$	$\rightarrow$	2
$d_2 = (C_1, C_2, C_4, C_3)$	$\rightarrow$	3
$d_3 = (C_1, C_3, C_2, C_4)$	$\rightarrow$	3
$d_4 = (C_1, C_3, C_4, C_2)$	$\rightarrow$	3
$d_5 = (C_1, C_4, C_2, C_3)$	$\rightarrow$	3
$d_6 = (C_1, C_4, C_3, C_2)$	$\rightarrow$	3

Figura 4.1: Ordenações de arestas de $\mathcal{H}$ que iniciam por C_1 e os respectivos graus de interseção.

Berge-acíclicas (Seção 4.2.1), redes "parcialmente acíclicas" (Seção 4.2.2) e redes cíclicas (Seção 4.2.3). Para isso, um parâmetro de ordenação de arestas chamado grau de interseção é definido[2].

Definição 4.3 (**Grau de interseção**)**.** Dado um hipergrafo $\mathcal{H}$ e uma ordenação de arestas $d = (C_1, \ldots, C_m)$, o grau de interseção da aresta C_i em relação a d é dado por $|C_i \cap \bigcup_{j=1}^{i-1} C_j|$. O grau de interseção da ordenação d é definido como o maior grau de interseção dentre todas as suas arestas, isto é,

$$\max\{|C_i \cap \bigcup_{j=1}^{i-1} C_j|, \text{ para } 1 \leq i \leq m \text{ na ordenação } d = (C_1, \ldots, C_m)\} \tag{4.3}$$

Para hipergrafos 3-uniformes, o grau de interseção será no máximo 3.

Exemplo 4.2. A Figura 4.1 apresenta um hipergrafo $\mathcal{H}$ e todas as suas ordenações de arestas que iniciam por C_1, com os respectivos graus de interseção. △

4.2.1 Redes Berge-acíclicas arco-consistentes

Seja $\mathcal{R}_1$ a rede representada na Figura 4.2, composta pelas restrições $R_{C_1}, \ldots, R_{C_4}$, com raiz x_1, obtida ao se codificar uma instância NCOP.

Considerando a atribuição inicial $\langle x_1 = \min D_1 \rangle$, deseja-se estender essa valoração a todas as variáveis da rede sem efetuar retrocesso, isto é, sem gerar conflitos. Cada valoração deve ser propagada sobre as variáveis restantes, o que pode ser feito utilizando técnicas de consistência direcionada (Seção 3.5.1). Por exemplo, se a restrição R_{C_1} é GAC direcionada em relação a x_1, então é possível estender a valoração $\langle x_1 = \min D_1 \rangle$ às variáveis x_2 e x_3 satisfazendo esta restrição, pois GAC direcionada garante que para todo $a_1 \in D_1$ existem $a_2 \in D_2$ e $a_3 \in D_3$ que

[2]Não confundir com o parâmetro de interseção proposto por Jégou (1993)

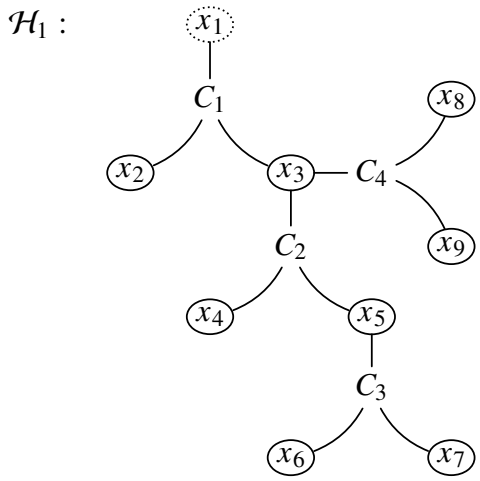


Figura 4.2: Hipergrafo de restrições da rede $\mathcal{R}_1$ composta pelas restrições $R_{C_1}, \ldots, R_{C_4}$.

satisfazem R_{C_1}. Se múltiplos valores de x_2 e x_3 existirem, então a atribuição pode ser feita de forma arbitrária, pois a valoração continuará consistente com R_{C_1}.

Do mesmo modo, uma vez definido o valor de x_2, se a restrição R_{C_2} é GAC direcionada em relação a x_2, então os valores de x_3 e x_4 são inferidos de forma consistente. Este processo continua para toda a rede, processando restrições com uma, e apenas uma, variável já valorada. Uma possível ordem de restrições processadas é $(R_{C_1}, R_{C_2}, R_{C_3}, R_{C_4})$. No caso geral, basta que a ordem escolhida inicie pela restrição cujo escopo contém a variável raiz e que esta ordenação, interpretada como uma ordem de arestas no hipergrafo de restrições, apresente um grau de interseção no máximo 1. Isto é, a ordem de restrições processadas deve satisfazer a propriedade (4.4).

$$|C_{p_i} \cap \bigcup_{j=1}^{i-1} C_{p_j}| < 2, \text{ para todo } 1 < i \leq m \text{ na ordem } (R_{C_{p_1}}, \ldots, R_{C_{pm}}) \tag{4.4}$$

Uma ordenação d satisfazendo (4.4) existe se, e somente se, o hipergrafo de restrições é Berge-acíclico, pois cada aresta nesta ordenação conecta-se com suas antecessoras em um único ponto: o vértice em comum.

Teorema 4.1. *Dado um hipergrafo $\mathcal{H}$ e uma aresta C_1, uma ordenação de arestas de $\mathcal{H}$ com grau de interseção no máximo 1 existe se, e somente se, $\mathcal{H}$ é Berge-acíclico.*

Demonstração.

($\Rightarrow$) A prova é por contradição. Seja uma ordenação de arestas d com grau de interseção no máximo 1. Supõe-se que $B = (C_1, x_1, \ldots, C_{l-1}, x_{l-1}, C_l = C_1)$, $l > 2$, é um Berge-ciclo de $\mathcal{H}$. Seja C_k a aresta de B com maior índice em d. Como $l > 2$, então existe um Berge-caminho $(C_{k-1}, x_{k-1}, C_k, x_k, C_{k+1})$ em B. Pela definição de Berge-ciclo, $x_{k-1} \in (C_{k-1} \cap C_k)$ e $x_k \in (C_k \cap C_{k+1})$. Isto é, $\{x_{k-1}, x_k\} \subseteq C_k \cap (C_{k-1} \cup C_{k+1})$. Logo, $|C_k \cap (C_{k-1} \cup C_{k+1})| \geq 2$. Como C_k é a aresta do Berge-ciclo com maior índice em d, então C_{k-1} e C_{k+1} antecedem C_k nesta ordenação e, portanto, d tem grau de interseção maior que 1. Logo, não existe um Berge-ciclo em $\mathcal{H}$.

($\Leftarrow$) Se um hipergrafo é desconexo e cada uma de suas componentes conexas maximais possui uma ordenação de arestas com grau de interseção no máximo 1, então todo o hipergrafo pode ser ordenado combinando-se as ordenações de suas componentes, que são disjuntas entre si. Dessa forma, basta provar que o teorema vale para $\mathcal{H}$ Berge-aciclico conexo. Seja $d = (C_1 \ldots, C_m)$ uma ordenação de $\mathcal{H}$ tal que $|C_1 \cap \bigcup_{j=1}^{i-1} C_j| \geq 1$, para todo $1 < i \leq m$. Essa ordenação existe sempre que $\mathcal{H}$ é conexo e pode ser obtida efetuando-se uma busca em largura a partir de C_1. Seja um conjunto ordenado de hipergrafos $(\mathcal{H}_1, \ldots, \mathcal{H}_l)$ tal que todo $\mathcal{H}_i$ é composto pelas arestas $(C_1, \ldots, C_i)$ em d e o seus respectivos vértices. É direto que $\mathcal{H}_i$ é conexo. O restante da prova é por contradição. Supõe-se que existe uma aresta C_k tal que $|C_k \cap \bigcup_{j=1}^{k-1} C_j| \geq 2$ em d. Sejam x_1 e x_2 os vértices dessa interseção. Logo, $\{x_1, x_2\} \subset C_k$ e $\{x_1, x_2\} \subset \bigcup_{j=1}^{k-1} C_j$, isto é, x_1 e x_2 pertencem ao hipergrafo $\mathcal{H}_{k-1}$. Como $\mathcal{H}_{k-1}$ é conexo, existe um caminho entre x_1 e x_2 (que não passa pela aresta C_k, pois C_k não está em $\mathcal{H}_{k-1}$). Dessa forma, em $\mathcal{H}_k$ existem dois caminhos entre x_1 e x_2 e, portanto, $\mathcal{H}$ não é Berge-acíclico. Logo, toda aresta C_k tem grau de interseção no máximo 1 em relação a d.

□

Corolário 4.1. *Consistência de arco generalizada direcionada é suficiente para resolver sem retrocesso um problema de otimização codificado se o seu hipergrafo de restrições é Berge-acíclico.*

Demonstração. Seja $\mathcal{H}$ um hipergrafo Berge-acíclico que codifica uma instância NCOP. Pelo Teorema 4.1, existe uma ordenação de arestas $d = (C_1, \ldots, C_m)$ com grau de interseção no máximo 1. Assim, seja $x_i \in C_i$ o único vértice da aresta que pode pertencer a arestas antecessoras em d. Supõe-se que toda restrição R_{C_i} da rede é GAC em relação à variável x_i. Ao processar as restrições seguindo a ordem d, cada restrição R_{C_i} tem apenas o valor de x_i definido pelo processamento das restrições anteriores (se não tiver, valora-se arbitrariamente a variável x_i). Como R_{C_i} é GAC em relação a x_i, então as variáveis em $C_i \setminus \{x_i\}$ são valoradas de forma consistente. □

O Corolário 4.1 é uma aplicação dos resultados de NCSPs acíclicos, apresentados na Seção 3.5.2, no cenário da otimização global. Mais resultados sobre a consistência de arco generalizada como condição suficiente para a solução de NCSPs acíclicos podem ser encontrados em Faltings (1998) e Cohen e Jeavons (2017).

4.2.2 Redes parcialmente acíclicas arco-consistentes relacional

Seja $\mathcal{R}_2$ a rede obtida ao se codificar uma instância NCOP, composta pelas restrições $R_{C_1}, \ldots, R_{C_5}$, com raiz x_1, cujo hipergrafo está representado na Figura 4.3. Nessa rede, não existe uma ordenação de restrições que satisfaça a propriedade (4.4), isto é, o grau de interseção mínimo

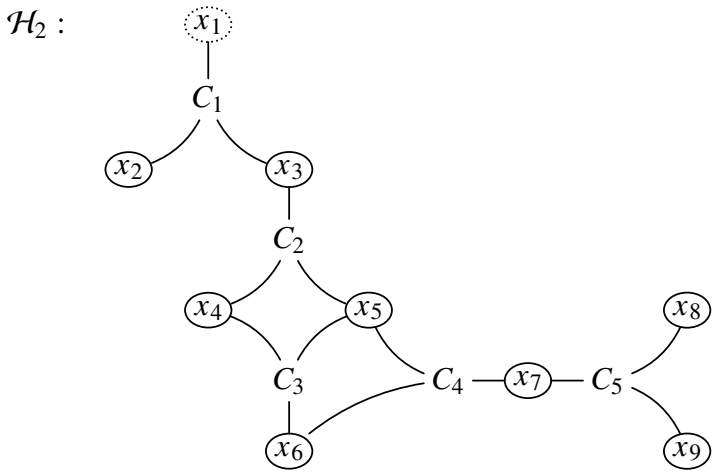


Figura 4.3: Hipergrafo de restrições da rede $\mathcal{R}_2$ composta pelas restrições $R_{C_1}, \ldots, R_{C_5}$.

de ordenações de arestas de $\mathcal{H}_2$ é maior que 1. Dessa forma, GAC direcionada não é suficiente para garantir uma solução livre de retrocesso estendendo $\langle x_1 = \min D_1 \rangle$. Neste exemplo, duas ordenações principais podem ser analisadas:

1. Seja $d_1 = (R_{C_1}, R_{C_2}, R_{C_3}, R_{C_4}, R_{C_5})$. Se a restrição R_{C_1} é GAC direcionada em relação a x_1, então a valoração $\langle x_1 = \min D_1 \rangle$ pode ser naturalmente estendida para x_2 e x_3. Da mesma forma, se a restrição R_{C_2} for GAC em relação a x_3, a valoração parcial de $\{x_1, x_2, x_3\}$ pode ser estendida a x_4 e x_5. No entanto, mesmo que a próxima restrição R_{C_3} seja GAC direcionada (ou mesmo GAC total), não há garantia de que a valoração parcial de x_4 e x_5 possa ser estendida a x_6. Por exemplo, seja R_{C_1}, R_{C_2} e R_{C_3} as seguintes restrições arco-consistentes sobre as variáveis $x_1, \ldots, x_6$, com domínios $D_1 = [-5, 1]$, $D_2 = [1, 3]$, $D_3 = [2, 6]$, $D_4 = [1, 3]$, $D_5 = [1, 2]$ e $D_6 = [2, 4]$:

$$R_{C_1} \equiv x_1 = x_2 - x_3$$
$$R_{C_2} \equiv x_3 = x_4 \cdot x_5$$
$$R_{C_3} \equiv x_6 = x_4 + x_5$$

   A valoração $\langle x_1 = -5 \rangle$ é estendida para $\langle x_2 = 1 \rangle$ e $\langle x_3 = 6 \rangle$ satisfazendo R_{C_1}. Da mesma forma, $\langle x_4 = 3 \rangle$ e $\langle x_5 = 2 \rangle$ são valoradas satisfazendo R_{C_2}. No entanto, não existe $a_6 \in [2, 4]$ tal que $a_6 = x_4 + x_5 = 5$. Por outro lado, se a restrição R_{C_3} for RAC em relação a x_6, então garantidamente existirá um valor $a_6 \in D_6$ que a satisfaça, independente dos valores assumidos por x_4 e x_5. Neste exemplo, se o domínio da variável x_6 fosse o intervalo $[2, 5]$, R_{C_3} seria RAC em relação a x_6 (pois, para quaisquer valores $a_4 \in D_4$ e $a_5 \in D_5$ existiria $a_6 \in D_6$ tal que $a_6 = a_4 + a_5$).

   Dessa forma, uma condição suficiente para a valoração de restrições R_{C_i} com duas variáveis já valoradas é estabelecida: basta que R_{C_i} seja RAC em relação à variável ainda não valorada. Na ordenação d_1, a restrição R_{C_4} também precisa ser RAC direcionada

para que o processo não encontre conflitos, enquanto que, para a restrição R_{C_5}, GAC direcionada é suficiente (ver Figura 4.3).

2. Seja $d_2 = (R_{C_1}, R_{C_5}, R_{C_4}, R_{C_3}, R_{C_2})$. Neste caso, se as restrições R_{C_1} e R_{C_5} forem GAC direcionadas e R_{C_4} e R_{C_3} forem RAC direcionadas, a atribuição $\langle x_1 = \min D_1 \rangle$ pode ser estendida às variáveis $\{x_1, \ldots, x_9\}$, satisfazendo essas quatro restrições. No entanto, não há garantia de que a restrição R_{C_2} seja satisfeita, pois todas as suas variáveis foram valoradas sem esta ter sido processada. Ou seja, mesmo que R_{C_2} seja GAC ou RAC direcionada, a valoração de variáveis seguindo a ordem de restrições d_2 pode encontrar um conflito.

É fácil verificar que o hipergrafo $\mathcal{H}_2$, ordenado de acordo com d_1, tem grau de interseção 2, enquanto a ordenação d_2 apresenta grau de interseção 3. De modo geral, RAC direcionada é suficiente para resolver redes ordenadas com um grau de interseção no máximo 2, isto é, que satisfaçam a propriedade (4.5).

$$|C_{p_i} \cap \bigcup_{j=1}^{i-1} C_{p_j}| < 3, \text{ para todo } 1 < i \leq m \text{ na ordenação } (R_{C_{p_1}}, \ldots, R_{C_{p_m}}) \tag{4.5}$$

Mais especificamente, em uma ordenação satisfazendo (4.5) as restrições $R_{C_{p_i}}$ que compartilham uma variável com as antecessoras devem satisfazer GAC direcionada, enquanto as restrições que compartilham duas variáveis devem satisfazer RAC direcionada. Em geral, uma rede de restrições pode ser ordenada de acordo com (4.5) se o seu hipergrafo tem uma estrutura "parcialmente acíclica" que é definida nesta tese através de uma decomposição chamada decomposição Epífita. Diferentemente da primeira classe de problemas, nos quais a estrutura Berge-acíclica dificilmente ocorre na prática, os resultados experimentais deste trabalho (Seção 5.4) indicam que instâncias NCOP codificadas como redes ternárias geralmente possuem tal decomposição. Decomposições Epífitas serão abordadas detalhadamente na Seção 4.3.

Teorema 4.2. *Consistência de arco relacional direcionada é suficiente para resolver sem retrocesso um problema de otimização codificado pela rede $\mathcal{R}$ com raiz x_1 se o seu hipergrafo de restrições possui uma ordenação de arestas iniciada por $C_1 \ni x_1$ com grau de interseção no máximo 2.*

Demonstração. Seja $d = (C_1, \ldots, C_m)$ uma ordenação de arestas de $\mathcal{H}$ com grau de interseção no máximo 2. Assim, seja $x_i \in C_i$ um vértice da aresta que não pertence a arestas antecessoras em d. Supõe-se que toda restrição R_{C_i} da rede é RAC em relação à variável x_i. Ao processar as restrições seguindo a ordem d, o valor de x_i, para cada restrição R_{C_i}, não tem valor definido até que essa restrição seja processada. Como R_{C_i} é RAC em relação a x_i, então qualquer valoração de $C_i \setminus \{x_i\}$ é estendida a x_i de forma consistente. □

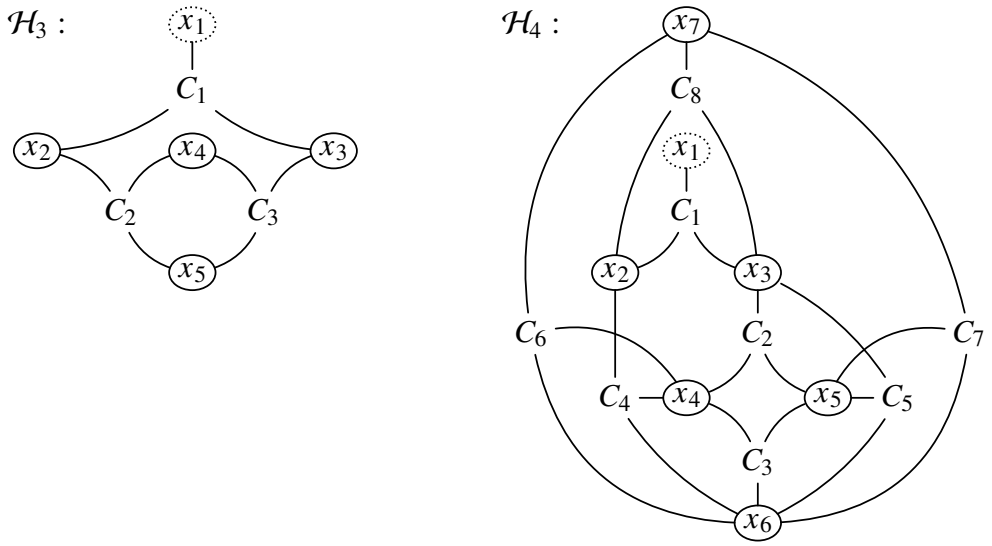


Figura 4.4: Hipergrafos de restrições das redes $\mathcal{R}_3$ e $\mathcal{R}_4$.

4.2.3 Redes cíclicas

Sejam $\mathcal{R}_3$ e $\mathcal{R}_4$ as redes cujos hipergrafos estão representados na Figura 4.4. Em ambos os casos, não existe ordenação de restrições iniciada pela raiz que satisfaça (4.4) ou (4.5), isto é, toda ordenação de arestas a partir de C_1 nos hipergrafos $\mathcal{H}_3$ e $\mathcal{H}_4$ tem grau de interseção maior que 2. Esses hipergrafos têm uma estrutura mais cíclica que os hipergrafos apresentados anteriormente (vale lembrar que o conceito de grau de ciclicidade é bastante utilizado na teoria de hipergrafos). Mais especificamente, os hipergrafos apresentados nas Figuras 4.2, 4.3 e 4.4 podem ser ordenados segundo o seu nível de ciclicidade da seguinte forma:

$$\mathcal{H}_1 \prec \mathcal{H}_2 \prec \mathcal{H}_3 \prec \mathcal{H}_4 \tag{4.6}$$

Um parâmetro que estabelece essa relação será definido na Seção 4.5, estendendo o conceito de decomposições Epífitas e classificando redes ordenadas com grau de interseção maior que 2.

4.3 Decomposições Epífitas

Na Seção 4.2.2 foi estabelecido que hipergrafos com estruturas "parcialmente acíclicas" podem ser ordenados com grau de interseção no máximo 2 e, consequentemente, RAC direcionada é suficiente para resolver redes de restrições com essa estrutura.

Nesta tese, essa estrutura parcialmente acíclica é caracterizada estendendo-se o conceito de árvores em hipergrafos, por meio de uma nova decomposição. Seguindo a tradição de nomear estruturas de dados inspirando-se na terminologia botânica, como "árvore", "floresta", "ramificação e poda", entre outros, essa decomposição é chamada decomposição Epífita.

Figura 4.5: *Bialbero di Casorzo*. Foto por Cioffi (2005).

Em botânica, epifitismo é a relação harmônica entre duas plantas na qual uma vive sobre a outra. O termo "epífito" vem do grego *epí* (sobre) e *phyto* (planta). Na Figura 4.5, o *Bialbero di Casorzo* é uma árvore epífita situada em Grana, na Itália, na qual uma cerejeira cresce sobre uma amoreira.

4.3.1 Definição

Informalmente, uma decomposição Epífita de um hipergrafo é um conjunto ordenado de árvores (hipergrafos parciais conexos e Berge-acíclicos), onde cada árvore "brota das árvores antecessoras" através de uma única aresta ou "do chão". A raiz de uma decomposição Epífita é definida como a raiz x_1 da primeira árvore e, nesse caso a decomposição é dita segundo x_1. Por exemplo, na Figura 4.6 o hipergrafo de restrições da rede $\mathcal{R}_2$ está desenhado de forma a mostrar uma decomposição Epífita segundo x_1 (as arestas direcionadas indicam a ordem das árvores).

Definição 4.4 (**Decomposição Epífita**)**.** Dado um hipergrafo $\mathcal{H} = (V, E)$ e um vértice $x_1 \in V$, uma decomposição Epífita segundo x_1 de $\mathcal{H}$ é uma tripla $(\mathcal{A}, \Omega, t)$, onde $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ é um conjunto ordenado de hipergrafos disjuntos $\mathcal{A}_i = (V_i, E_i)$ tais que $\bigcup_{i=1}^{n} V_i = V$, Ω é o conjunto de arestas de $\mathcal{H}$ que não pertencem a $\bigcup_{i=1}^{n} E_i$ (isto é, $\Omega = E \setminus \bigcup_{i=1}^{n} E_i$) e $t : \Omega \mapsto V_\Omega$ é uma função que associa cada aresta $C_i \in \Omega$ com um de seus vértices $t(C_i) \in C_i$ tal que:

1. $\mathcal{A}_1$ é uma árvore enraizada em x_1 (isto é, um hipergafo conexo e Berge-acíclico);
2. $\forall \mathcal{A}_{i>1} \in \mathcal{A}$: existe no máximo um $C_i \in \Omega$ tal que $t(C_i) \in V_i$ e $\mathcal{A}_i$ é uma árvore enraizada em $t(C_i)$ (ou qualquer outro vértice, se tal aresta C_i não existir);

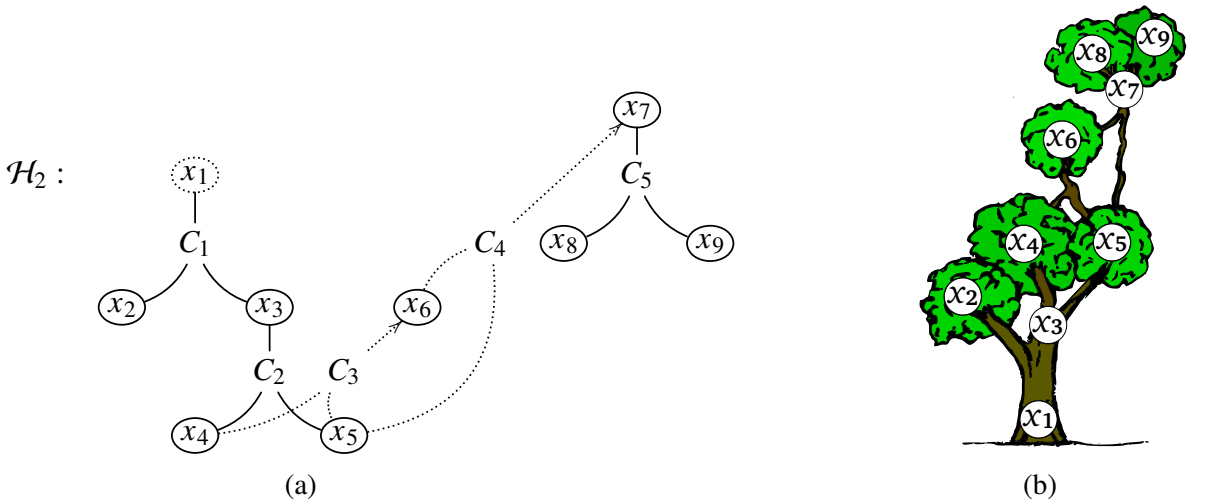


Figura 4.6: (a) Decomposição Epífita segundo x_1 composta por três árvores e (b) uma ilustração desta decomposição.

3. se $t(C_i) \in V_i$, então $C_i \setminus \{t(C_i)\} \subseteq \bigcup_{j=1}^{i-1} V_j$.

Definição 4.5 (**Altura epífita**)**.** A altura epífita de uma decomposição Epífita $(\mathcal{A}, \Omega, t)$ é a cardinalidade do conjunto Ω.

Por convenção, diz-se que uma rede $\mathcal{R}$ possui uma decomposição Epífita se o seu hipergrafo de restrições possui uma decomposição Epífita.

Exemplo 4.3. A rede da Figura 4.6 possui uma decomposição Epífita segundo x_1 dada por $((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), \Omega, t)$, onde $\mathcal{A}_1 = (\{x_1, x_2, x_3, x_4, x_5\}, \{C_1, C_2\})$, $\mathcal{A}_2 = (\{x_6\}, \emptyset)$ e $\mathcal{A}_3 = (\{x_7, x_8, x_9\}, C_5)$ são hipergrafos disjuntos, conexos e Berge-acíclicos, $\Omega = \{C_3, C_4\}$ e t é uma função satisfazendo as condições da Definição 4.4 tal que $t(C_3) = x_6$ e $t(C_4) = x_7$. A altura epífita desta decomposição é 2.

Do mesmo modo, a rede da Figura 4.7 possui uma decomposição Epífita segundo x_1 de altura epífita 5, cujas árvores são $\mathcal{A}_1 = (\{x_1, x_2, x_3\}, \{C_1\})$, $\mathcal{A}_2 = (\{x_9, x_{10}, x_{11}, x_{12}, x_{13}\}, \{C_6, C_7\})$, $\mathcal{A}_3 = (\{x_7\}, \emptyset)$, $\mathcal{A}_4 = (\{x_8\}, \emptyset)$, $\mathcal{A}_5 = (\{x_5\}, \emptyset)$, $\mathcal{A}_6 = (\{x_6\}, \emptyset)$ e $\mathcal{A}_7 = (\{x_4\}, \emptyset)$. Neste caso, a árvore $\mathcal{A}_2$ "não brota" da árvore $\mathcal{A}_1$, mas "do chão". O conjunto Ω dessa decomposição é dado por $\Omega = \{C_2, C_3, C_4, C_5, C_8\}$. △

A existência de decomposição Epífita é uma condição necessária e suficiente para que exista uma ordenação de arestas com grau de interseção no máximo 2. Deste modo, se uma instância NCOP codificada com raiz x_1 possuir uma decomposição Epífita segundo x_1 e, além disso, for RAC direcionada seguindo uma ordenação topológica das árvores da decomposição, então é possível instanciá-la sem efetuar retrocesso. Mais especificamente, não é necessário que todas as restrições da rede sejam RAC direcionadas, apenas aquelas cuja aresta esteja no conjunto Ω. Para as demais, GAC direcionada é suficiente, visto que estas compõem hipergrafos parciais Berge-acíclicos na decomposição. Dessa forma, quanto menor a cardinalidade de Ω,

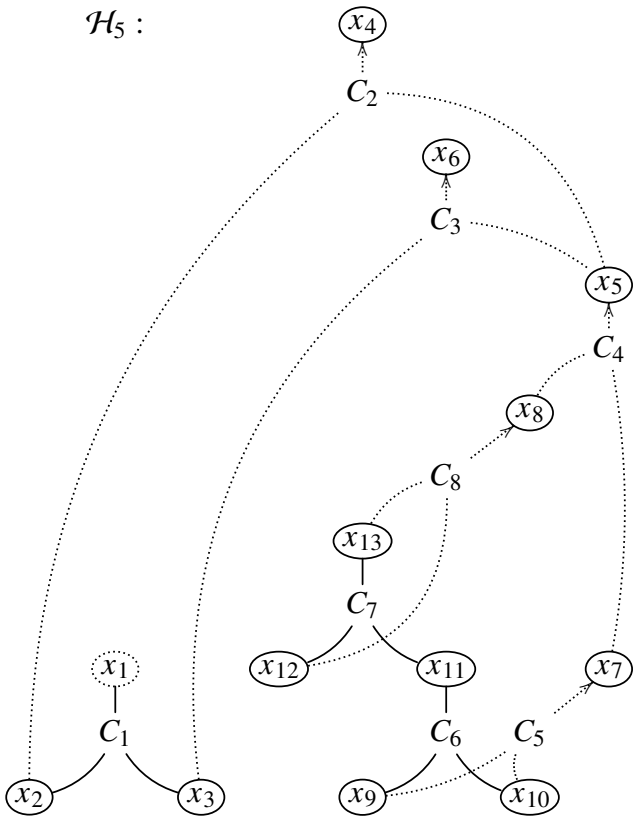


Figura 4.7: Decomposição Epífita segundo x_1.

mais "fácil" de resolver é a instância. Quando $\Omega = \emptyset$, a decomposição Epífita é, na verdade, uma única árvore, isto é, um hipergrafo conexo e Berge-acíclico.

Teorema 4.3. *Uma rede $\mathcal{R}$ possui uma decomposição Epífita segundo x_1 se, e somente se, existe uma ordenação de restrições $(R_{C_1}, \ldots, R_{C_m})$ tal que $x_1 \in C_1$ e $(C_1, \ldots, C_m)$ tem grau de interseção no máximo 2.*

Demonstração.

($\Rightarrow$) Seja $\mathcal{R}$ uma rede que possui uma decomposição Epífita $(\mathcal{A}, \Omega, t)$, onde $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$. Pelo Teorema 4.1, toda árvore $\mathcal{A}_i = (V_i, E_i)$ possui uma ordenação de arestas d_i com grau de interseção no máximo 1. Como essas árvores são disjuntas entre si, a ordenação obtida ao combinar todas as ordenações d_i, $1 \leq i \leq n$, também possui um grau de interseção no máximo 1. Basta provar que ao adicionar as arestas de Ω nesta nova ordenação o seu grau de interseção não será maior que 2. Seja $C_i \in \Omega$ e $\mathcal{A}_i \in \mathcal{A}$ a árvore tal que $t(C_i) \in V_i$. Pela condição 3 da Definição 4.4, $C_i \setminus \{t(C_i)\} \subseteq \bigcup_{j=1}^{i-1} V_j$. Ao adicionar C_i entre a última aresta da ordenação d_{i-1} e a primeira de d_i, tem-se que:

i) o grau de intercessão das arestas de $\mathcal{A}_j$, para todo $j < i$, não se altera, pois C_i sucede essas arestas na ordenação;

ii) o grau de intercessão das arestas de $\mathcal{A}_j$, para todo $j > i$, também não se altera, pois $V_j \cap C_i = \emptyset$;

iii) o grau de intercessão das arestas de $\mathcal{A}_i$ pode aumentar uma unidade, pois C_i contém apenas o vértice $t(C_i)$ que pertence a V_i, e, portanto, o grau de intercessão da ordenação passa a ser no máximo 2;

iv) o grau de intercessão de C_i é 2, pois apenas dois de seus vértices pertencem a arestas que o antecedem na ordenação.

($\Leftarrow$) Seja $d = (C_1, \ldots, C_m)$ uma ordenação de arestas com grau de interseção no máximo 2. Uma decomposição Epífita pode ser obtida a partir de d construindo iterativamente os conjuntos $\mathcal{A}$ e Ω, inicialmente vazios. Seja $d_{(\leq 1)}$ o conjunto ordenado obtido ao remover-se de d as arestas com grau de interseção 2, e seja $d_{(2)}$ o conjunto formado por estas arestas, seguindo a mesma ordem que aparecem em d. Pelo Teorema 4.1, as arestas em $d_{(\leq 1)}$ formam uma floresta. Seja A o conjunto de todas as árvores dessa floresta. Como C_1 tem grau de interseção 0, seja $\mathcal{A}_1$ a árvore de A que contém essa restrição. Adiciona-se em $\mathcal{A}$ a árvore $\mathcal{A}_1$ enraizada em $x_1 \in C_1$. Em seguida, para cada aresta C_i da ordenação $d_{(2)}$, define-se $t(C_i)$ como o vértice de C_i que não pertence a arestas antecessoras em d (esse vértice existe, pois C_i tem grau de intercessão 2 em d). Por outro lado, $t(C_i)$ pode ou não pertencer a arestas sucessoras de C_i em d. No caso positivo, existe uma árvore $\mathcal{A}_{i_3}$ em A que contém esse vértice. Do contrário, a árvore $\mathcal{A}_{i_3} = (\{t(C_i)\}, \emptyset)$ é criada e adicionada ao conjunto de árvores A. Sejam $\mathcal{A}_{i_1}$ e $\mathcal{A}_{i_2}$, não necessariamente distintas, as árvores em A que contêm os vértices de $C_i \setminus \{t(C_i)\}$ (novamente, essas árvores existem pois C_i tem grau de interseção 2). Dessa forma, adiciona-se em $\mathcal{A}$ as árvores $\mathcal{A}_{i_1}$, $\mathcal{A}_{i_2}$ e $\mathcal{A}_{i_3}$, nesta ordem, que ainda não estão nesse conjunto, onde $\mathcal{A}_{i_3}$ é enraizada em $t(C_i)$. Por fim, remove-se de $d_{(2)}$ a aresta C_i e a adiciona em Ω.

Para provar que, ao processar todas as arestas de $d_{(2)}$, a tupla $(\mathcal{A}, \Omega, t)$ é uma decomposição Epífita, basta mostrar que $\forall \mathcal{A}_{i>1} \in \mathcal{A}$ existe no máximo um $C_i \in \Omega$ tal que $t(C_i) \in V_i$. Supõe-se que existem $C_i, C_j \in \Omega$ distintos, tais que $t(C_i), t(C_j) \in V_i$. Como $t(C_i)$ e $t(C_j)$ foram escolhidos de forma a não pertencerem a antecessores em d, então todas as arestas de $\mathcal{A}_i$ sucedem C_i e C_j (se $\mathcal{A}_i$ não possui arestas, então $t(C_i) \in C_j$ e $t(C_j) \in C_i$, contradizendo o fato de que estes vértices não pertencem a antecessores em d). Como $\mathcal{A}_i$ é árvore, existe um Berge-caminho entre $t(C_i)$ e $t(C_j)$. Seja C_k a aresta desse caminho com maior índice em d. C_k tem grau de interseção 2 em d (ver Figura 4.8). Mas $\mathcal{A}_i$ possui apenas arestas do conjunto $d_{(\leq 1)}$, que não contém arestas com grau de interseção maior que 1 em d. Logo, C_i e C_j não podem ser distintos.

□

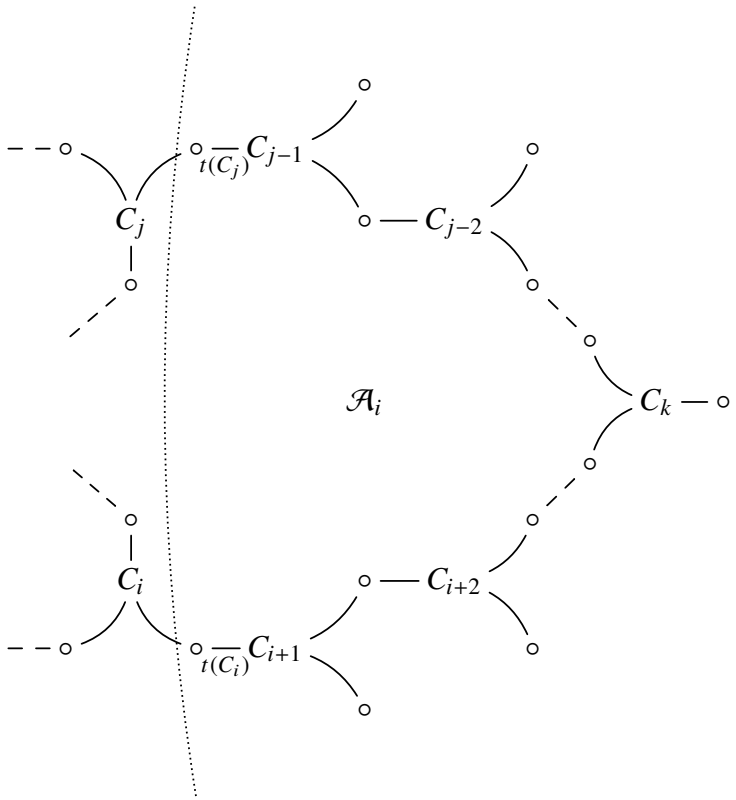


Figura 4.8: Um Berge-caminho entre os vértices $t(C_i)$ e $t(C_j)$ na árvore $\mathcal{A}_i$, no qual existe uma aresta C_k com grau de interseção 2.

Corolário 4.2. *Consistência de arco relacional direcionada é suficiente para resolver um problema de otimização codificado com raiz x_1 se o seu hipergrafo de restrições possui uma decomposição Epífita segundo x_1.*

Demonstração. Consequência direta dos Teoremas 4.2 e 4.3. □

4.3.2 Condição de existência de decomposições Epífitas

Existem instâncias NCOP que não possuem decomposições Epífitas. Dada uma rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, se existe um subconjunto de restrições $S \subseteq C$ no qual todas as variáveis em S ocorrem em mais de uma restrição nesse subconjunto, então $\mathcal{R}$ não possui essa decomposição. O Teorema 4.4 formaliza esse resultado pela equivalência entre decomposições Epífitas e ordenações de arestas com grau de interseção no máximo 2. Para isso, lembra-se que $\mathcal{H}' = (V', E')$ é um hipergrafo parcial de $\mathcal{H} = (V, E)$ se $E' \subseteq E$ e $V' = \bigcup_{C_i \in E'} C_i$, e denota-se por $\mathcal{H} - C_i$ o hipergrafo parcial com arestas $E \setminus \{C_i\}$.

Teorema 4.4. *Um hipergrafo $\mathcal{H}$ possui uma ordenação de arestas $d = (C_1, \ldots, C_m)$ com grau de interseção no máximo 2 se, e somente se, todo hipergrafo parcial $\mathcal{H}'$ de $\mathcal{H}$ com mais de uma aresta possui pelo menos um vértice $x' \notin C_1$ de grau 1.*

Demonstração.

($\Rightarrow$) A prova é por contradição. Seja $d = (C_1, \ldots, C_m)$ uma ordenação de arestas de $\mathcal{H}$ com grau de interseção no máximo 2. Supõe-se que existe um hipergrafo parcial $\mathcal{H}'$ de $\mathcal{H}$ com mais de uma aresta no qual todo vértice $x' \notin C_1$ tem grau maior que 1. Seja C_k a aresta de $\mathcal{H}'$ com maior indice em d ($k > 1$). Como todos os vértices $x' \notin C_1$ têm grau maior que 1, então todo $x' \in C_k$ também pertence a outra aresta de $\mathcal{H}'$. No entanto, todas as outras arestas de $\mathcal{H}'$ têm índice menor que k em d e, portanto, $|C_k \cap \bigcup_{j=1}^{k-1} C_j| = 3$, contradizendo o fato de d ter grau de interseção no máximo 2. Logo, todo hipergrafo parcial $\mathcal{H}'$ de $\mathcal{H}$ com mais de uma aresta possui pelo menos um vértice $x' \notin C_1$ de grau 1.

($\Leftarrow$) Seja $\mathcal{H}$ um hipergrafo com arestas $\{C_1, \ldots, C_m\}$ tal que todo hipergrafo parcial $\mathcal{H}'$ de $\mathcal{H}$ com mais de uma aresta têm pelo menos um vértice $x' \notin C_1$ de grau 1. Seja $(\mathcal{H}_1, \ldots, \mathcal{H}_m = \mathcal{H})$ uma sequência de hipergrafos parciais de $\mathcal{H}$ tal que $\mathcal{H}_1$ possui apenas a aresta C_1 e, para todo $1 < i \leq m$, o hipergrafo $\mathcal{H}_{i-1}$ é obtido removendo-se de $\mathcal{H}_i$ a aresta C_i que contém um vértice de grau 1 em $\mathcal{H}_i$. A sequência de arestas removidas é tal que C_i possui pelo menos um vértice que não pertence a nenhuma aresta C_j, $1 \leq j < i$. Isto é, $|C_i \cap \bigcup_{j=1}^{i-1} C_j| < 3$, para todo $1 < i \leq m$. Dessa forma, a ordenação $(C_1, \ldots, C_m)$ tem grau de interseção no máximo 2.

□

A prova do Teorema 4.4 motiva um método para encontrar uma ordenação de arestas com grau de interseção no máximo 2, construindo uma sequência d do último para o primeiro elemento, escolhendo em cada passo uma aresta com vértice de grau 1 no hipergrafo parcial obtido ao remover-se arestas já processadas. O Algoritmo 18 descreve esse método, recebendo como entrada um hipergrafo $\mathcal{H}$ e uma aresta C_1 e devolvendo, em caso de sucesso, uma ordenação de arestas d iniciada por C_1 com grau de interseção no máximo 2. Esse algoritmo é uma extensão natural para hipergrafos do método `min-width` apresentado em Dechter (2003), proposto originalmente por Freuder (1982), mas restrito a sempre escolher uma aresta com vértice de grau 1 ao invés da aresta com vértice de grau mínimo.

Teorema 4.5. *Dado um hipergrafo ternário $\mathcal{H}$ com m arestas e uma aresta C_1, a execução de* `ordem_de_grau_2`$(\mathcal{H}, C_1)$ *consome tempo $O(m)$.*

Demonstração. Considera-se uma estrutura de dados para representar o hipergrafo através de duas listas `V` e `E`, onde `V` é uma lista de vértices e cada vértice contém uma lista de ponteiros para arestas, e `E` é uma lista de arestas, onde cada aresta contém 3 ponteiros para seus respectivos ponteiros nas listas de vértices. Essa estrutura é construída em tempo linear e mantém uma pilha de vértices de grau no máximo 1 em tempo constante. Dessa forma, todas as operações do algoritmo são executadas em tempo $O(1)$. Como o laço da linha 2 é executado m vezes, então `ordem_de_grau_2`$(\mathcal{H}, C_1)$ consome tempo $O(m)$. □

Algoritmo 18 Ordem de grau de interseção 2 (`ordem_de_grau_2`$(\mathcal{H}, C_1)$)

Entrada: Hipergrafo $\mathcal{H} = (V, E)$ e aresta C_1.
Saída: Ordenação de arestas d iniciada por C_1 com grau de interseção no máximo 2.

```
1: d ← ∅
2: enquanto E ≠ {C_1} faça
3:     se existe x ∈ V de grau 1 tal que x ∈ C_i e i > 1 então
4:         empilha(C_i, d)
5:         H ← H − C_i
6:     senão
7:         devolve FALHA
8:     fim se
9: fim enquanto
10: empilha(C_1, d)
11: devolve d
```

Exemplo 4.4. O hipergrafo da Figura 4.6 pode ser representado computacionalmente conforme a Figura 4.9. Nesse caso, a pilha contém, inicialmente, os vértices de grau 1 em $\mathcal{H}$ que não pertencem a C_1. Escolhendo, por exemplo, a variável x_8 (linha 3 do algoritmo), sua aresta C_5 é inserida em d e removida do hipergafo. Para isso, basta remover os respectivos ponteiros das listas das variáveis de C_5. Nesse processo, empilha-se as variáveis que agora têm grau 1 (e que não pertencem a C_1), obtendo uma nova pilha (x_7). Esse processo demanda um tempo de processamento constante, pois apenas 3 variáveis devem ser analisadas para atualização da pilha. △

O Algoritmo 18 não define qual aresta deve ser escolhida no caso de mais de uma satisfazer a condição da linha 3, pois tal escolha não interfere na completude do algoritmo. Para fins práticos, duas ordenações distintas podem implicar em um mesmo fluxo de processamento da rede ao se tentar uma valoração sem retrocesso. De fato, o grau de interseção define uma relação de ordem **parcial** entre arestas. Por exemplo, em uma ordenação $d = (C_1, \ldots, C_k, \ldots, C_m)$ com grau de interseção 2, onde todas as arestas $C_1, \ldots, C_{k-1}$ têm grau de interseção 1 e C_k tem grau de interseção 2, a sub-ordem $(C_1, \ldots, C_{k-1})$ pode ser livremente permutada sem que isso altere o grau de interseção da ordenação.

Do mesmo modo, as árvores que constituem uma decomposição Epífita podem ser eventualmente embaralhadas sem alterar o processamento das restrições. Por exemplo, na rede de restrições cujo hipergrafo está representado na Figura 4.7 é irrelevante se a restrição R_{C_8} é processada antes ou depois da restrição R_{C_5}, bem como é irrelevante se as variáveis de C_1 são valoradas antes ou depois das variáveis de C_6 e C_7. Em geral, essas variantes não alteram o conjunto Ω da decomposição Epífita. Dado um hipergrafo de restrições, existe um conjunto mínimo de relações de ordem que caracterizam a estrutura principal da decomposição. No exemplo da Figura 4.7, esse conjunto mínimo é dado por $\{(C_7, C_8), (C_6, C_5), (C_8, C_4), (C_5, C_4), (C_4, C_3), (C_1, C_3), (C_3, C_2)\}$, onde o par (C_i, C_j) indica que C_i deve ser processada antes que C_j. Não há, por exemplo, uma relação de ordem estabelecida entre C_6 e C_7 ou C_1 e C_4. Nesse sentido, decomposições Epífitas distintas podem resultar no mesmo fluxo de processamento (e no mesmo conjunto Ω).

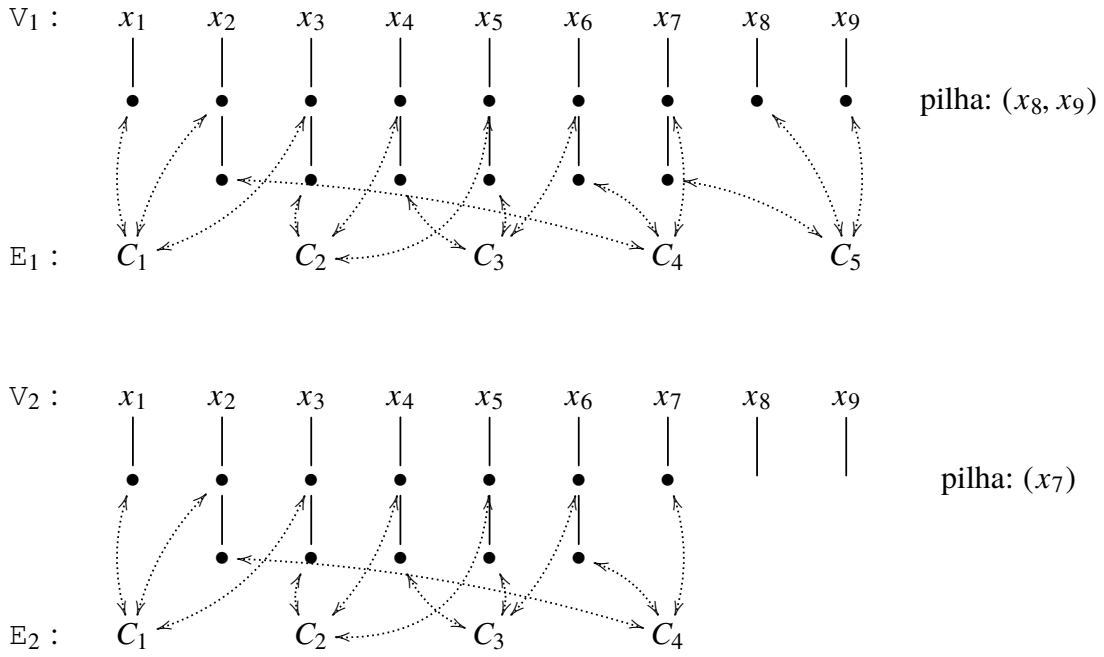


Figura 4.9: Exemplo de estrutura de dados para execução de `ordem_de_grau_2`$(\mathcal{H}, C_1)$ em tempo linear.

Por outro lado, existem ordenações de um mesmo hipergrafo $\mathcal{H}$ que caracterizam decomposições Epífitas distintas, e essas decomposições também apresentam um fluxo de processamento diferente. No exemplo da Figura 4.7, a ordenação com grau de interseção 2 dada por $(C_1, C_8, C_7, C_6, C_5, C_4, C_2, C_3)$ caracteriza uma decomposição Epífita segundo x_1 composta pelas árvores $\mathcal{A}_1 = (\{x_1, x_2, x_3\}, \{C_1\})$, $\mathcal{A}_2 = (\{x_8, x_{12}, x_{13}\}, \{C_8\})$, $\mathcal{A}_3 = (\{x_9, x_{10}, x_{11}\}, \{C_6\})$, $\mathcal{A}_4 = (\{x_7\}, \emptyset)$, $\mathcal{A}_5 = (\{x_5\}, \emptyset)$, $\mathcal{A}_6 = (\{x_4\}, \emptyset)$ e $\mathcal{A}_7 = (\{x_6\}, \emptyset)$, onde $\Omega = \{C_2, C_3, C_4, C_5, C_7\}$. Para a aplicação abordada nesta tese, conjuntos Ω distintos implicam em condições diferentes para se alcançar a consistência que garante otimização global sem retrocesso, pois altera-se o conjunto de restrições que precisam satisfazer RAC direcionada. Em geral, a aridade de Ω, isto é, a altura epífita, e o significado das restrições desse conjunto para o problema original são parâmetros que devem ser levados em consideração a fim de se propor uma heurística que vise minimizar essas condições.

Assim, uma instância NCOP possui diversas decomposições Epífitas e muitas dessas decomposições são, no sentido prático, equivalentes. No entanto, existem problemas que não possuem tal representação (como discutido na Seção 4.2.3). Essa classe de problemas será analisada na Seção 4.5, mas uma forma alternativa de abordar algumas das instâncias dessa classe é alterar a representação do problema original como rede ternária, adicionando constantes. Em um hipergrafo de restrições, constantes são sempre representadas por vértices de grau 1 e, consequentemente, possibilitam a verificação do condicional da linha 3 do Algoritmo 18.

Exemplo 4.5. Seja $\mathcal{R}_3$ uma rede composta pelas restrições $R_{C_1} \equiv x_1 = x_2 + x_3$, $R_{C_2} \equiv x_2 = x_4 \cdot x_5$ e $R_{C_3} \equiv x_3 = x_4 / x_5$, cujo hipergrafo $\mathcal{H}_3$ está representado na Figura 4.4. Essa rede não possui

uma decomposição Epífita. No entanto, existe uma rede $\mathcal{R}'_3$ equivalente cujo hipergrafo possui tal decomposição:

$$\mathcal{R}_3 = \begin{cases} x_1 = x_2 + x_3 \\ x_2 = x_4 \cdot x_5 \\ x_3 = x_4/x_5 \end{cases} \quad \Rightarrow \quad \begin{aligned} x_1 &= x_4 \cdot x_5 + x_4/x_5 \\ &= x_4 \cdot (x_5 + 1/x_5) \end{aligned} \quad \Rightarrow \quad \mathcal{R}'_3 = \begin{cases} x_1 = x_4 \cdot y_1 \\ y_1 = x_5 + y_2 \\ y_2 = 1/x_5 \end{cases}$$

△

4.4 Aproximando consistência de arco relacional direcionada

Embora o Corolário 4.2 seja significativo, pois estabelece uma relação entre a estrutura da rede e o nível de consistência que garante uma busca livre de retrocesso, alcançar RAC direcionada consome, no pior caso, tempo de processamento e espaço de memória exponenciais (Dechter, 2003). Além disso, ao se aplicar essa consistência a estrutura do hipergrafo é alterada, pois novas restrições são adicionadas à rede. Toda decomposição Epífita está associada a uma ordenação de restrições com grau de interseção no máximo 2. Ao se tentar alcançar RAC direcionada, as novas restrições adicionadas devem ser processadas **antes** que as restrições originais com as quais compartilham variáveis, pois do contrário essas restrições serão certamente violadas. Nesse sentido, uma rede que possui uma decomposição Epífita pode deixar de apresentar tal propriedade. Vale lembrar que o mesmo problema ocorre com os resultados teóricos de Freuder (1982) e Jégou (1993): para alcançar (hiper-)k-consistência é necessário adicionar novas restrições de aridade $k - 1$ na rede, alterando a largura do respectivo (hiper)grafo.

Exemplo 4.6. Seja $\mathcal{R}$ a rede composta pelas seguintes restrições:

$$\begin{aligned} R_{C_1} &\equiv x_1 = x_2 - x_3 \\ R_{C_2} &\equiv x_3 = x_4 + x_5 \\ R_{C_3} &\equiv x_6 = x_4 \cdot x_5 \\ R_{C_4} &\equiv x_7 = x_2 - x_6 \end{aligned}$$

Essa rede possui uma decomposição Epífita segundo x_1, conforme ilustrado na Figura 4.10. Sejam os seguintes domínios arco-consistentes: $D_1 = [-3, 3]$, $D_2 = [1, 3]$, $D_3 = [0, 4]$, $D_4 = [0, 2]$, $D_5 = [0, 2]$, $D_6 = [0, 4]$ e $D_7 = [-2, 2]$. As restrições R_{C_1}, R_{C_2} e R_{C_3} são RAC direcionadas em relação a x_1, x_3 e x_6, respectivamente. No entanto, a restrição R_{C_4} não é RAC direcionada em relação a x_7, pois a valoração parcial $\{\langle x_2 = 1\rangle, \langle x_6 = 4\rangle\}$ não possui extensão $a_7 = x_2 - x_6$ no domínio $D_7 = [-2, 2]$. Efetuando uma valoração da rede iniciada por $\langle x_1 = -3\rangle$, obtém-se a valoração parcial $\{\langle x_1 = -3\rangle, \langle x_2 = 1\rangle, \langle x_3 = 4\rangle, \langle x_4 = 2\rangle, \langle x_5 = 2\rangle, \langle x_6 = 4\rangle\}$, que não pode ser estendida a x_7.

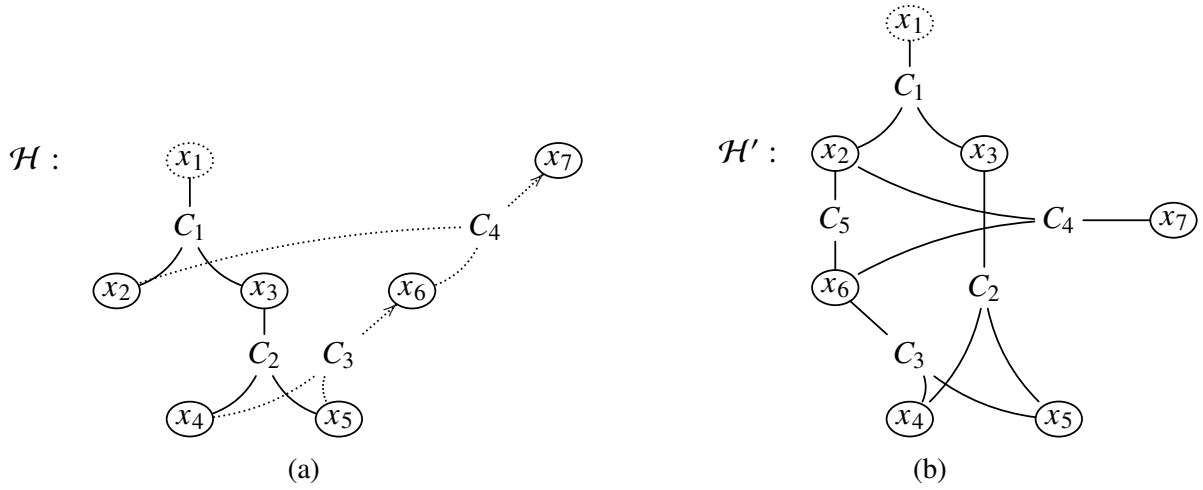


Figura 4.10: (a) Decomposição Epífita segundo x_1 composta por três árvores e (b) o hipergrafo resultante ao se adicionar uma nova restrição binária R_{C_5} sobre x_2 e x_6 na rede.

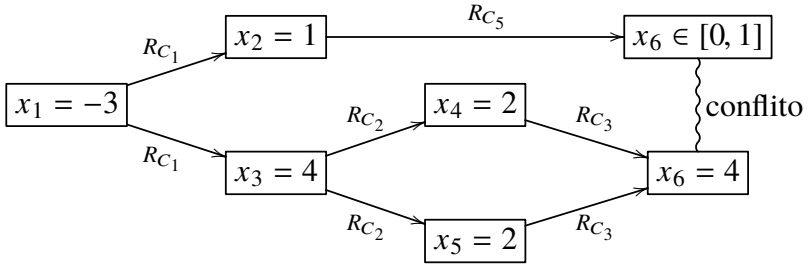


Figura 4.11: Valoração da rede do Exemplo 4.6 adicionada a restrição $R_{C_5} \equiv x_2 + x_6 \in [-2, 2]$. Um conflito ocorre na valoração da variável x_6.

Para que R_{C_4} torne-se RAC direcionada é preciso incluir na rede uma nova restrição R_{C_5} sobre as variáveis x_2 e x_6, forçando a relação $x_2 + x_6 \in [-2, 2]$. Naturalmente, ao efetuar uma valoração de variáveis, R_{C_5} deve ser analisada antes das valorações de x_2 e x_6 e, portanto, a ordenação de restrições original $(R_{C_1}, R_{C_2}, R_{C_3}, R_{C_4})$ é alterada para $(R_{C_1}, R_{C_5}, R_{C_2}, R_{C_3}, R_{C_4})$. Essa nova ordenação, no entanto, não está associada a uma decomposição Epífita (ver Figura 4.10). De fato, se a rede for valorada seguindo esta ordem, um conflito é encontrado na valoração da variável x_6 (Figura 4.11), ou seja, RAC direcionada não é suficiente para garantir uma solução ótima sem retrocesso na rede alterada. △

Nesta tese, além de caracterizar RAC direcionada como condição suficiente para otimização global sem retrocesso, um método para alcançar essa consistência sem adicionar novas restrições, embora de forma aproximada, é proposto, possibilitando aplicar o Corolário 4.2 de forma prática. De modo geral, "técnicas de consistência locais que apenas reduzem domínios, como GAC, tendem a ser mais práticas que aquelas que alteram a estrutura do hipergrafo de restrições ou das relações das restrições" (Bessiere et al., 2008, pág. 1). Desta forma, propõe-se um método de redução de domínios semelhante aos contratores das consistências $3B$ e de caixa (Seção 3.3), isto é, uma verificação de consistência seguida de um processo de bissecção de

domínios. Como as redes de interesse são aquelas que codificam problemas de otimização, essa abordagem deve ser incorporada a um método de ramificação e poda.

4.4.1 Método de ramificação e poda

Uma restrição $R_{C_i} \equiv (x_1 = x_2 \circ x_3)$ é RAC em relação a x_1 se, e somente se, para todo $a_2 \in D_2$ e $a_3 \in D_3$ existe $a_1 \in D_1$ tal que $a_1 = a_2 \circ a_3$, isto é, $D_1 \supseteq D_2 \circ D_3$.

Sem perda de generalidade, pode-se assumir que R_{C_i} é GAC, pois o contrator dessa consistência é bem definido e apenas reduz o domínio das variáveis:

$$D_1 \leftarrow D_1 \cap (D_2 \circ D_3)$$
$$D_2 \leftarrow D_2 \cap (D_1 \bullet D_3)$$
$$D_3 \leftarrow D_3 \cap (D_1 \diamond D_2)$$

Nesse caso, D_1 é tal que $D_1 \subseteq D_2 \circ D_3$ e a condição necessária para que R_{C_i} seja RAC direcionada é reduzida à equação

$$D_1 = D_2 \circ D_3 \tag{4.7}$$

Se R_{C_i} não é RAC direcionada (isto é, $D_1 \subset D_2 \circ D_3$), uma possibilidade de alcançar essa consistência de forma alternativa a adicionar uma nova restrição entre x_2 e x_3 na rede é reduzir o multi-intervalo resultante de $D_2 \circ D_3$. Pela inclusão isotônica[3] da aritmética intervalar, essa redução pode ser alcançada reduzindo algum dos domínios D_2 ou D_3. No entanto, como assumiu-se a consistência de arco em R_{C_i}, qualquer redução de seus domínios pode excluir valorações consistentes. Por exemplo, a restrição $x_1 = x_2 + x_3$, que é GAC para $D_1 = [1, 2]$ e $D_2 = D_3 = [0, 2]$, não é RAC direcionada em relação a x_1, mas toda atribuição a x_2 pode ser estendida a uma solução consistente.

A fim de percorrer todo o espaço de busca essa estratégia deve ser implementada utilizando um método de ramificação e poda com bissecção de domínios, pois existem infinitos subdomínios de D_2 e D_3 que implicam a consistência de arco relacional direcionada de R_{C_i}. Além disso, uma tolerância ε deve ser considerada na equação $D_1 = D_2 \circ D_3$, uma vez que o método de bissecção pode não encontrar domínios RAC em tempo de processamento tratável. Uma implicação desta relaxação é que uma valoração de x_1, x_2 e x_3 pode ser infactível em R_{C_i} por um fator de ε, o que será dito ε-factível.

Definição 4.6 (**Satisfazibilidade com tolerância**)**.** Uma restrição $R_{C_i} \equiv x_1 = x_2 \circ x_3$ é satisfeita com tolerância absoluta ε_A se x_1, x_2 e x_3 são valoradas com valores $a_1 \in D_1$, $a_2 \in D_2$ e $a_3 \in D_3$ tais que

$$|a_1 - (a_2 \circ a_3)| \leq \varepsilon_A \tag{4.8}$$

[3]Dados $B, B' \in \mathbb{M}^n$ e uma função intervalar $\mathcal{F} : \mathbb{M}^n \mapsto \mathbb{M}$, se $B' \subseteq B$, então $\mathcal{F}(B') \subseteq \mathcal{F}(B)$.

Alternativamente, R_{C_i} é satisfeita com tolerância relativa ε_R se

$$|a_1 - (a_2 \circ a_3)| \leq \varepsilon_R \cdot |a_2 \circ a_3| \tag{4.9}$$

Por abuso de notação, diz-se que R_{C_i} é satisfeita com tolerância ε_{AR} se R_{C_i} é satisfeita com tolerância absoluta ε_A ou tolerância relativa ε_R, isto é, ε_{AR} denota o conjunto $\varepsilon_{AR} = \{\varepsilon_A, \varepsilon_R\}$.

Definição 4.7 (ε_{AR}**-factível).** Dada uma rede $\mathcal{R}$ que possui uma decomposição Epífita $(\mathcal{A}, \Omega, t)$, uma valoração de $\mathcal{R}$ é dita ε_{AR}-factível se todas as restrições de Ω são satisfeitas com tolerância ε_{AR}, enquanto as demais restrições de $\mathcal{R}$ são satisfeitas da forma usual (tolerância 0).

Utilizando esse método de ramificação e poda, RAC direcionada é alcançada de forma aproximada. Valorar as variáveis de acordo com uma decomposição Epífita de uma rede RAC aproximada pode resultar em uma solução ε_{AR}-factível e, consequentemente, a instância do problema de otimização original não é resolvida de forma exata. O grau dessa aproximação depende das operações definidas pelas restrições, da estrutura da rede (como o conjunto Ω) e da tolerância ε_{AR} permitida.

Quando aplicado em todas as restrições que devem ser RAC direcionadas, o método proposto é uma variante da ramificação e poda intervalar para problemas de otimização (Seção 3.4.2). O Algoritmo 19 descreve esse procedimento. Uma busca em profundidade é executada, aplicando o contrator GAC para reduzir domínios localmente inconsistentes (linha 6). Se nenhum domínio vazio é gerado, tenta-se valorar toda a rede a partir da valoração inicial $\langle x_1 = \min D_1 \rangle$, construindo um conjunto ordenado I_Ω de restrições que não são satisfeitas com tolerância ε_{AR} (linha 8). Tais restrições precisam ser restringidas a fim de se alcançar RAC direcionada, o que é feito pelo procedimento `seleciona_e_bissecta` (linha 12); este procedimento também recebe como parâmetro uma precisão $\Delta > 0$ que limita a bissecção de multi-intervalos com comprimento mínimo, como ocorre nos métodos de ramificação e poda intervalares usuais. Se o conjunto I_Ω é vazio, então a valoração é uma solução ε_{AR}-factível da rede e a busca deste ramo termina. Ramos que garantidamente não têm solução melhor que a ótima z^* encontrada até a iteração corrente não são processados pela busca, pois uma restrição dinâmica $x_1 < z^*$ é adicionada à rede (linha 6). Um parâmetro de complexidade relevante para esse algoritmo é a altura epífita da rede, pois esta delimita o tamanho de I_Ω e apenas domínios de variáveis que pertencem a escopos de restrições nesse conjunto são ramificadas.

O procedimento `tenta_valoracao` valora as variáveis de cada restrição seguindo a ordem d. Variáveis de restrições R_{C_i}, tais que $C_i \notin \Omega$, são valoradas utilizando-se o contrator GAC, pois a rede é GAC direcionada em relação a d. Caso contrário, considera-se a restrição $R'_{C_i} \equiv (x_1 = x_2 \circ x_3)$ equivalente a R_{C_i}, $C_i \in \Omega$, tal que $t(C_i) = x_1$. Como x_2 e x_3 já foram valoradas, respectivamente, com valores $a_2 \in D_2$ e $a_3 \in D_3$, pois as restrições de Ω são processadas depois que as árvores que contêm as variáveis em $C_i \setminus \{t(C_i)\}$ (condição 3 da Definição 4.4), esse procedimento atribui a x_1 o valor $a_1 \in D_1$ mais próximo de $a_2 \circ a_3$. Se o erro absoluto $|a_1 - (a_2 \circ a_3)|$ dessa aproximação é maior que ε_A e o erro relativo $|a_1 - (a_2 \circ a_3)|/|a_2 \circ a_3|$

Algoritmo 19 RAC direcionada aproximada (`RAC_direcionada_aprox`$(\mathcal{R}, x_1, \varepsilon_{AR}, \Delta)$)

Entrada: Rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ que possui uma decomposição Epífita segundo x_1, conjunto de tolerâncias $\varepsilon_{AR} = \{\varepsilon_A, \varepsilon_R\}$ e precisão $\Delta > 0$.
Saída: Aproximação z^* do mínimo global de $\mathcal{R}$ em relação a x_1.

```
1: z* ← +∞
2: empilha(D, pilha)
3: seja d uma ordenação de restrições de R com grau de interseção no máximo 2, a partir de x1
4: enquanto pilha ≠ ∅ faça
5:    D ← desempilha(pilha)
6:    D ← GAC_direcionada((X, D, C ∪ {x1 < z*}), d)
7:    se ∀Di ∈ D : Di ≠ ∅ então
8:       (z, IΩ) ← tenta_valoracao((X, D, C), d, εAR)
9:       se IΩ = ∅ então
10:         z* ← z
11:      senão
12:            (D', D'') ← seleciona_e_bissecta((X, D, C), IΩ, Δ)
13:            empilha(D', pilha)
14:            empilha(D'', pilha)
15:      fim se
16:   fim se
17: fim enquanto
18: devolve z*
```

for maior que ε_R, então essa valoração não é ε-factível e R_{C_i} é colocada em I_Ω. A busca continua com a próxima restrição de d independente da valoração parcial ser ε_{AR}-factível ou não. Ao final, I_Ω contém todas as restrições que não foram satisfeitas com tolerâncias ε_{AR} e o procedimento retorna o valor z atribuído a x_1.

O procedimento `seleciona_e_bissecta` escolhe uma restrição $R_{C_i} \in I_\Omega$ e particiona o domínio de uma de suas variáveis, retornando dois subdomínios $\mathcal{D}'$ e $\mathcal{D}''$. Uma heurística para a variável a ser particionada é escolher uma variável da restrição R_{C_i} com maior índice em I_Ω. Do mesmo modo que em CSPs estruturados em árvores a consistência direcionada deve ser aplicada no sentido inverso da ordenação de restrições, reduzir as variáveis do ponto mais alto da decomposição Epífita em direção à raiz garante que a bissecção de domínios em uma determinada restrição não altere a consistência de domínios com índices mais altos na ordenação. As variáveis $x \in C_i \setminus \{t(C_i)\}$ são então ramificadas e o subdomínio que minimiza a tolerância ε_{AR} necessária para que R_{C_i} torne-se RAC direcionada, no ramo corrente, é definida como o novo domínio $\mathcal{D}'$, e seu complemento como o novo domínio $\mathcal{D}''$ (mais detalhes dessas heurísticas serão abordados na Seção 5.3.2).

Embora GAC direcionada seja suficiente para garantir a valoração das restrições cujos escopos não estão no conjunto Ω quando todas as restrições desse conjunto são RAC direcionadas, níveis mais altos de consistência podem ser utilizados para melhorar o desempenho do método ao se aproximar RAC por um fator $\varepsilon_{AR} > 0$. Por exemplo, o procedimento `GAC_direcionada` pode ser substituído por um contrator GAC total, como o algoritmo `GAC-3`, podando os domínios correntes de forma mais eficiente que utilizando apenas a abordagem direcionada.

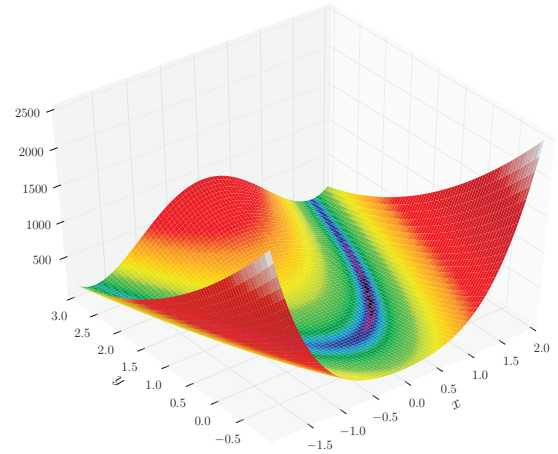


Figura 4.12: Função de Rosenbrock. Figura por Ortiz (2012a).

Exemplo 4.7. A função de Rosenbrock $f(x, y) = 100(x - y^2)^2 + (1 - y)^2$ é uma função não convexa introduzida por Rosenbrock (1960) e comumente utilizada como instância de teste para algoritmos de otimização. Essa função define graficamente um vale (ver Figura 4.12) no qual está contido o mínimo global da função, com valor $f(1, 1) = 0$. Em geral, encontrar o vale é trivial, mas convergir para o mínimo global é difícil. A função de Rosenbrock, codificada como uma rede ternária com variável raiz $z_7 = f(x, y)$ é dada por:

$$\begin{aligned} &\min \quad z_7 \\ &\text{sujeito a} \end{aligned} \tag{4.10}$$

$$\begin{aligned} &R_{C_1} \equiv z_7 = z_4 + z_6 && R_{C_4} \equiv z_2 = x - z_1 && R_{C_6} \equiv z_5 = 1 - y \\ &R_{C_2} \equiv z_4 = 100 \cdot z_3 && R_{C_5} \equiv z_6 = {z_5}^2 && R_{C_7} \equiv z_1 = y^2 \\ &R_{C_3} \equiv z_3 = {z_2}^2 \end{aligned} \tag{4.11}$$

Esta rede possui uma decomposição Epífita de altura epífita 2, apresentada na Figura 4.13 (existem outras decomposições Epífitas para essa rede). Sejam os domínios iniciais $D_x = [-10, 10]$ e $D_y = [-10, 5]$ e a execução do Algoritmo 19 sobre essa instância, com tolerância absoluta $\varepsilon_A = 0{,}1$. Primeiramente, na linha 3 do algoritmo, uma ordenação de restrições com grau de interseção no máximo 2 é definida seguindo-se a ordem da decomposição Epífita: $d = (C_1, \ldots, C_7)$. Em seguida, o contrator GAC direcionado é aplicado, obtendo os seguintes domínios GAC direcionados:

$$\begin{aligned} &D_{z_7} = [0, 1210121] && D_{z_3} = [0, 12100] && D_{z_5} = [-4, 11] \\ &D_{z_4} = [0, 1210000] && D_{z_2} = [-110, 10] && D_x = [-10, 10] \\ &D_{z_6} = [0, 121] && D_{z_1} = [0, 100] && D_y = [-10, 5] \end{aligned}$$

Como todos os domínios são não vazios, o algoritmo tenta a valoração da rede a partir da atribuição inicial $\langle z_7 = 0 \rangle$, adicionando em I_Ω as restrições de Ω (nessa decomposição Epífita

existe apenas a restrição $R_{C_7} \equiv z_1 = y^2$) que não são satisfeitas com tolerância $\varepsilon_A = 0{,}1$. Nesse caso, a seguinte valoração ocorre:

$$\begin{array}{lll} \langle z_7 = 0 \rangle & \langle z_3 = 0 \rangle & \langle z_5 = 0 \rangle \\ \langle z_4 = 0 \rangle & \langle z_2 = 0 \rangle & \langle x = 5 \rangle \\ \langle z_6 = 0 \rangle & \langle z_1 = 5 \rangle & \langle y = 1 \rangle \end{array}$$

A restrição R_{C_7} não é satisfeita com tolerância ε_A, pois $|5 - 1^2| = 4 > 0{,}1$. É importante notar que nesta valoração, ao processar-se a restrição $R_{C_4} \equiv z_2 = x - z_1$, a atribuição $\langle z_2 = 0 \rangle$ não infere os valores de z_1 e x, apenas reduz ambos os domínios ao intervalo $[0, 10]$. Uma atribuição a z_1 e x, dentro desse intervalo, é então escolhida arbitrariamente (mais especificamente, o algoritmo escolhe o ponto médio do intervalo). Embora o mínimo global dessa rede seja $z_7{}^* = 0$, uma valoração consistente total, sem retrocesso, é improvável no domínio corrente, visto que a restrição R_{C_7} não é RAC direcionada.

Como $I_\Omega = \{R_{C_7}\}$, na linha 9 do algoritmo, uma variável será selecionada e seu domínio particionado. Apenas variáveis das restrições em I_Ω podem ser selecionadas segundo a heurística proposta. Supõe-se, então, que y seja selecionada e seu domínio $D_y = [-10, 5]$ particionado em dois subdomínios $D_y{}' = [-10,\ -2{,}5]$ e $D_y{}'' = (-2{,}5,\ 5]$, que são colocados na pilha de retrocesso segundo uma heurística que visa maximizar as chances de encontrar uma solução ε_A-factível (nesse caso, o subdomínio $D_y{}''$ será processado antes que $D_y{}'$). A execução volta então à linha 4 do algoritmo e o processo se repete, obtendo um novo conjunto de domínios GAC direcionados e tentando uma nova valoração a partir da atribuição inicial de z_7. A Figura 4.14 mostra o fluxo de execução completo do algoritmo, enquanto a Tabela 4.1 apresenta os domínios das variáveis ramificadas em cada iteração. Cada linha desta tabela apresenta o respectivo nó na árvore de busca, a ramificação que gerou esse nó, os domínios das variáveis z_7, y e z_1 após a aplicação do procedimento `GAC_direcionada` (as demais variáveis foram omitidas para uma melhor visualização) e o erro de satisfazibilidade da restrição $R_{C_7} \equiv z_1 = y^2$. Quando esse erro é no máximo a tolerância $\varepsilon_A = 0{,}1$, o ótimo global z^* é atualizado; nas próximas iterações, este valor é utilizado para reduzir o domínio D_{z_7}. Em particular, se D_{z_7} é um intervalo aberto à esquerda, então o ótimo global é o extremo esquerdo desse intervalo somado a um ϵ que denota o próximo número real representado na máquina.

Neste exemplo, o método encontra o mínimo global de forma exata, mas esse não é o caso geral. Por exemplo, para a instância *sample*, o método encontra a solução 701,272735, com tolerância $\varepsilon_{AR} = 0{,}1$, após 1353 nós visitados na árvore de busca. Com tolerância $\varepsilon_{AR} = 0{,}01$, o algoritmo processa 11091 nós e encontra a solução 724,7933. O mínimo global da instância *sample*, dado no *benchmark* COCONUT, é aproximadamente 726,6367. △

Tabela 4.1: Execução do método `RAC_direcionada_aprox` (Algoritmo 19) aplicado à função de Rosenbrock no Exemplo 4.7.

iteração	nó	ramificação	domínios consistentes	erro e z^*
1	1A	-	$D_{z_7} = [0,\ 1210121]$ $D_y = [-10,\ 5]$ $D_{z_1} = [0,\ 100]$	$\lvert z_1 - y^2\rvert = 4$
2	2A	$D_y \leftarrow (-2{,}5,\ 5]$	$D_{z_7} = [0,\ 122516]$ $D_y = (-2{,}5,\ 5]$ $D_{z_1} = [0,\ 25]$	$\lvert z_1 - y^2\rvert = 4$
3	3A	$D_y \leftarrow (-2{,}5,\ 1{,}25]$	$D_{z_7} = [0,\ 26418{,}5)$ $D_y = (-2{,}5,\ 1{,}25]$ $D_{z_1} = [0,\ 6{,}25)$	$\lvert z_1 - y^2\rvert \approx 2{,}125$
4	4A	$D_y \leftarrow (-0{,}625,\ 1{,}25]$	$D_{z_7} = [0,\ 13371{,}782)$ $D_y = (-0{,}625,\ 1{,}25]$ $D_{z_1} = [0,\ 1{,}5625]$	$\lvert z_1 - y^2\rvert \approx 0{,}219$
5	5A	$D_{z_1} \leftarrow (0{,}7812,\ 1{,}5625]$	$D_{z_7} = [0,\ 13369{,}2]$ $D_y = (0{,}8839,\ 1{,}25)$ $D_{z_1} = (0{,}7812,\ 1{,}5625]$	$\lvert z_1 - y^2\rvert \approx 0{,}172$
6	6A	$D_y \leftarrow (1{,}0669,\ 1{,}25)$	$D_{z_7} = (0{,}00448,\ 13369{,}2)$ $D_y = (1{,}0669,\ 1{,}25)$ $D_{z_1} = (1{,}13836,\ 1{,}5625)$	$\lvert z_1 - y^2\rvert \approx 0{,}212$
7	7A	$D_y \leftarrow (1{,}1585,\ 1{,}25)$	$D_{z_7} = (0{,}025113,\ 13369{,}2)$ $D_y = (1{,}1585,\ 1{,}25)$ $D_{z_1} = (1{,}342,\ 1{,}5625)$	$\lvert z_1 - y^2\rvert \leq 0{,}1$ $(z^* \leftarrow 0{,}025113 + \epsilon)$
8	7B	$D_y \leftarrow (1{,}0669,\ 1{,}1585)$	$D_{z_7} = (0{,}00448,\ 0{,}025113)$ $D_y = (1{,}0669,\ 1{,}1585)$ $D_{z_1} = (1{,}138360,\ 1{,}342)$	$\lvert z_1 - y^2\rvert \leq 0{,}1$ $(z^* \leftarrow 0{,}00448 + \epsilon)$
9	6B	$D_y \leftarrow (0{,}8839,\ 1{,}0669]$	$D_{z_7} = [0,\ 0{,}00448)$ $D_y = (0{,}8839,\ 1{,}0669]$ $D_{z_1} = (0{,}87,\ 1{,}13836]$	$\lvert z_1 - y^2\rvert \leq 0{,}1$ $(z^* \leftarrow 0)$
10	5B	$D_{z_1} \leftarrow [0,\ 0{,}7812]$	$D_{z_7} = [0,\ 0) = \emptyset$	
11	4B	$D_y \leftarrow (-2{,}5,\ -0{,}625]$	$D_{z_7} = [0,\ 0) = \emptyset$	
12	3B	$D_y \leftarrow (1{,}25,\ 5]$	$D_{z_7} = [0,\ 0) = \emptyset$	
13	2B	$D_y \leftarrow [-10,\ -2{,}5]$	$D_{z_7} = [0,\ 0) = \emptyset$	

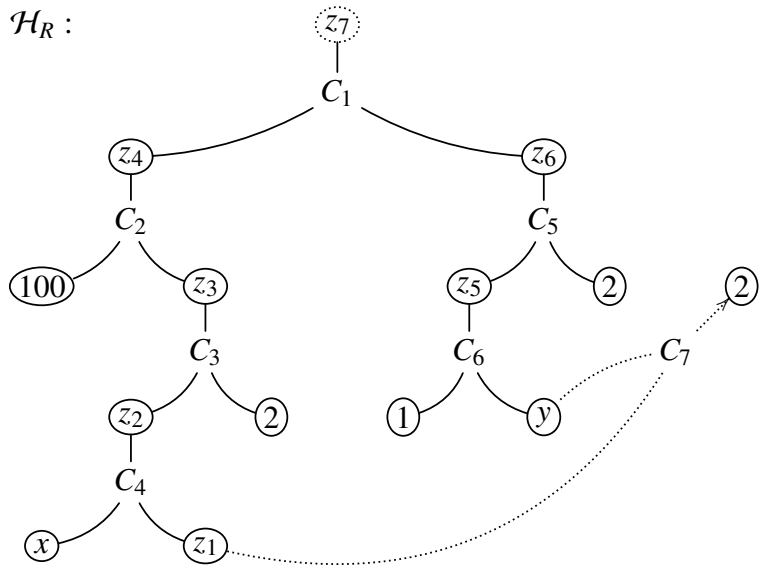


Figura 4.13: Uma decomposição Epífita da função de Rosenbrock codificada como uma rede ternária com raiz z_7.

4.4.2 Comparação com o método de ramificação e poda intervalar

Nesta seção, o Algoritmo 19 é comparado com o método de ramificação e poda intervalar para problemas de otimização, apresentado na Seção 3.4.2. Ambos os métodos apresentam uma estrutura similar de busca em profundidade com retrocesso, aplicando um contrator de consistência como método de poda local. As principais diferenças entre as duas abordagens são a forma como soluções (não necessariamente ótimas) são utilizadas para guiar a busca e a heurística de escolha de variáveis na fase de ramificação.

Primeiramente, o procedimento `tenta_valoracao` (linha 8 do Algoritmo 19) é uma especialização do procedimento `limitante_superior` (linha 7 do Algoritmo 15), que provê uma valoração factível no espaço de busca corrente, não necessariamente ótima. Métodos de busca local são geralmente utilizados para encontrar tais soluções, como o Método de Newton. Alternativamente, o procedimento `tenta_valoracao` efetua uma valoração natural da rede, assim como ocorre em Araya et al. (2014) e Ninin et al. (2015). No entanto, essa solução é garantidamente ótima no subdomínio corrente, pois estende a valoração inicial $\langle x_1 = \min D_1 \rangle$. A principal diferença entre essa abordagem e o procedimento `limitante_superior` usual é que aqui uma solução pode não ser encontrada no ramo atual, mas se for, então a solução é mínima no domínio corrente e não há necessidade de continuar a busca em subdomínios do ramo atual. Vale mencionar que outros resolvedores intervalares também relaxam restrições na fase de valoração (Ninin et al., 2015; Trombettoni et al., 2011), como ocorre ao considerar-se RAC direcionada aproximada.

Além disso, ambos os métodos apresentam o procedimento `escolhe_e_bissecta`. A escolha de qual variável e partição de seu domínio será o próximo ramo do espaço de busca é crucial para uma execução mais rápida de buscas com retrocesso. Várias heurísticas foram

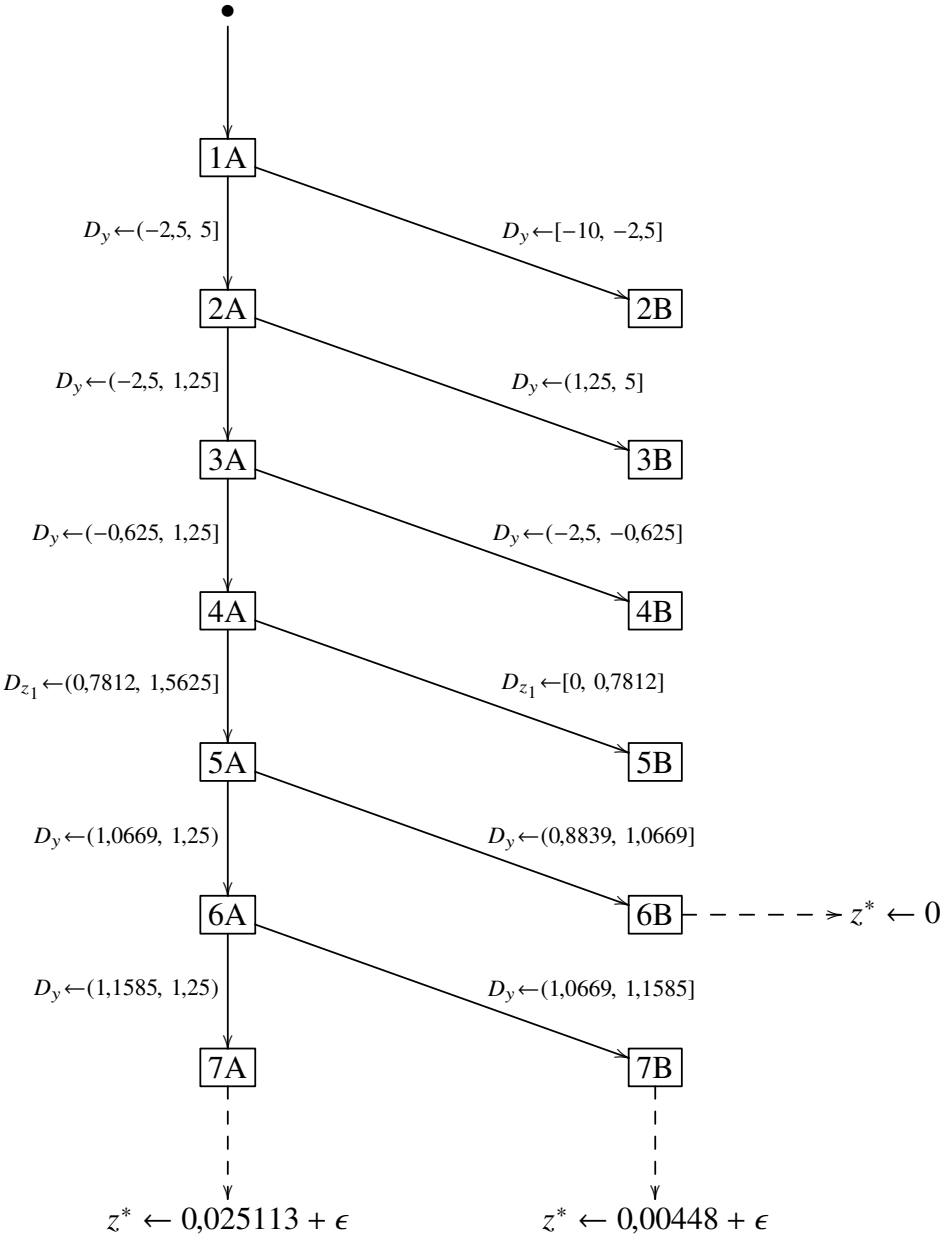


Figura 4.14: Fluxo de execução do método `RAC_direcionada_aprox` (Algoritmo 19) aplicado à função de Rosenbrock no Exemplo 4.7.

propostas nas últimas décadas, como escolher a variável com o maior domínio (multi-intervalo de maior comprimento) ou esquemas baseado em Smear (Csendes e Ratz, 1997). Nesse sentido, o método proposto provê uma heurística livre de domínio para a escolha de variáveis na fase de ramificação: variáveis de restrições do conjunto Ω que não são ε_{AR}-factíveis e que maximizam a aproximação da consistência de arco relacional direcionada da rede no ramo corrente. Tal heurística é uma importante contribuição deste trabalho, pois encontrar mínimos locais é uma boa estratégia para reduzir o espaço de busca em métodos de ramificação e poda, podendo ser implementada em outros resolvedores intervalares.

4.5 Decomposições $k-$Epífitas

Na Seção 4.2, redes ternárias de problemas de otimização foram classificadas em três classes principais, de acordo com a existência de ordenação de restrições com grau de interseção k, para $k < 2$, $k < 3$ e $k = 3$. No primeiro caso, mostrou-se que se o hipergrafo de restrições é Berge-acíclico então GAC direcionada é suficiente para encontrar a solução ótima do problema. Para a segunda classe, o hipergrafo de restrições deve possuir uma decomposição Epífita, e uma combinação de GAC direcionada e RAC direcionada garantem um conjunto de domínios no qual uma busca sem retrocesso resolve o problema de forma ótima, embora aplicar tais consistências de forma exata seja, em geral, intratável. Nesta seção, os resultados estabelecidos para decomposições Epífitas são estendidos à terceira classe de problemas, na qual todas as instâncias apresentam ordenações de arestas com grau de interseção maior que 2.

Como apontado na Seção 4.2.3, os hipergrafos $\mathcal{H}_3$ e $\mathcal{H}_4$, apresentados na Figura 4.4, não possuem decomposições Epífitas. Além disso, foi dito que o hipergrafo $\mathcal{H}_4$ apresenta uma estrutura mais cíclica que $\mathcal{H}_3$. Embora ambos os hipergrafos pertençam à terceira classe de problemas, estes também podem ser classificados quanto ao seu nível de ciclicidade. Assim, o conjunto de instâncias que não possuem decomposições Epífitas é dividido em subclasses. Para isso, um parâmetro estrutural de hipergrafos chamado **largura epífita** é introduzido, considerando ordenações de vértices ao invés de ordenações de arestas.

Várias definições de largura de hipergrafos foram propostas a fim de caracterizar classes de CSPs, conforme apresentado na Seção 3.5. Em particular, Robertson e Seymour (1986) e Gottlob et al. (2000) introduziram as noções de largura de árvore (largura arbórea) e largura de hiperárvore definidas sobre decomposições em árvores (Dechter e Pearl, 1989); no entanto, encontrar tais decomposições e, consequentemente, essas larguras, é um problema $\mathcal{NP}$-difícil (Gottlob et al., 2000). Alternativamente, a largura epífita é uma extensão natural da largura de grafos, proposta por Freuder (1978) para CSPs binários.

A largura de um vértice x_i em um grafo G com uma ordenação de vértices b (de acordo com Freuder (1978)) é dado pelo número de pais de x_i em b. Em hipergrafos, o conceito de pai de um vértice não é bem definido. Neste trabalho, interpreta-se essa largura, de forma equivalente, pelo número de arestas às quais x_i pertence e que ainda não foram contabilizadas nas larguras de

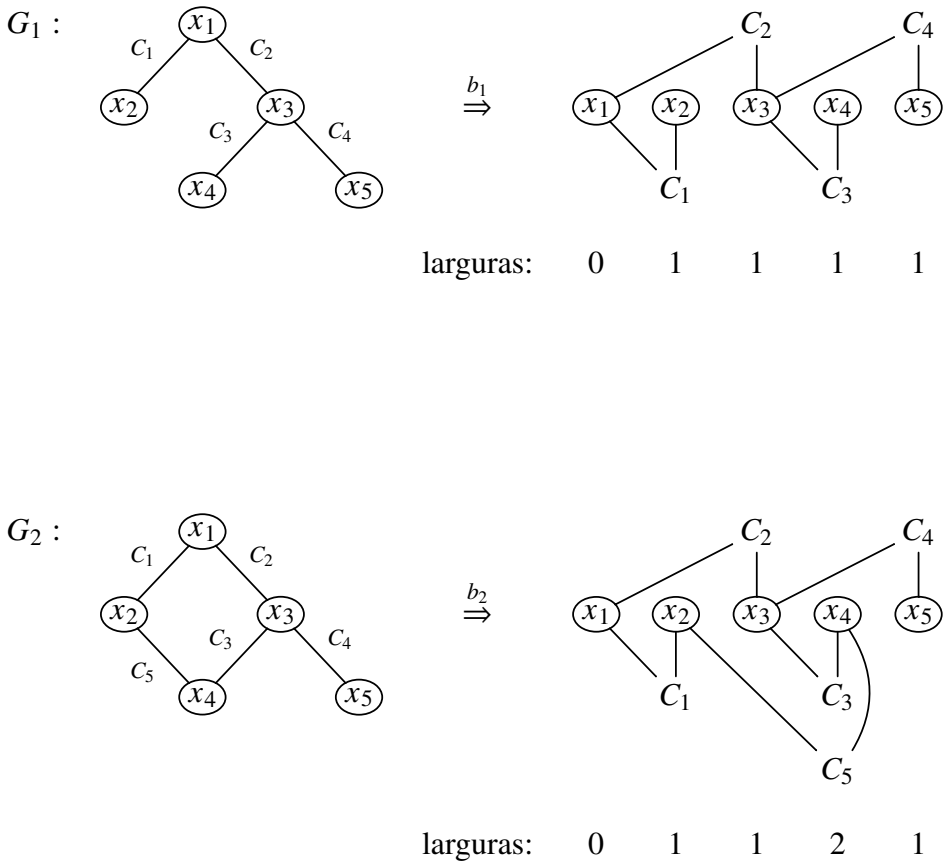


Figura 4.15: Grafos G_1 e G_2 ordenados de acordo com d_1 e d_2, e a largura de cada vértice na respectiva ordenação. A largura da ordenação d_1 é 1, enquanto a largura da ordenação d_2 é 2.

outros vértices. Isto é, dada uma ordenação de vértices b, a largura de um vértice x_i é o número de arestas $C_1, \ldots, C_k$ tais que x_i é o último vértice de qualquer $C_1, \ldots, C_k$ em b. A Figura 4.15 exemplifica essa interpretação com dois grafos ordenados G_1 e G_2.

Essa definição é naturalmente estendida a hipergrafos. A Figura 4.16 apresenta os três hipergrafos $\mathcal{H}_2$, $\mathcal{H}_3$ e $\mathcal{H}_4$ vistos na Seção 4.2, onde apenas $\mathcal{H}_2$ possui uma decomposição Epífita. Nas ordenações representadas na figura, $\mathcal{H}_2$ apresenta largura 1, $\mathcal{H}_3$ largura 2 e $\mathcal{H}_4$ largura 3.

Definição 4.8 (Largura Epífita). A largura epífita de um vértice x_i na ordenação $b = (x_1, \ldots, x_n)$ é o número de arestas $C_1, \ldots, C_k$ tais que x_i é o último vértice de qualquer $C_1, \ldots, C_k$ em b. A largura epífita de uma ordenação b é a maior largura epífita dentre todos os vértices em b. A largura epífita de um hipergrafo $\mathcal{H}$ em relação a x_1 é a menor largura epífita dentre todas as ordenações de vértices de $\mathcal{H}$ iniciadas por x_1.

Teorema 4.6. *Um hipergrafo possui uma decomposição Epífita segundo x_1 se, e somente se, tem largura epífita* 1 *em relação a x_1.*

Demonstração. Essa equivalência é provada mostrando que é possível converter uma ordenação de arestas $d = (C_1, \ldots, C_m)$ com grau de interseção no máximo 2 em uma ordenação de vértices $b = (x_1, \ldots, x_n)$ com largura epífita 1, tal que $x_1 \in C_1$, e vice-versa.

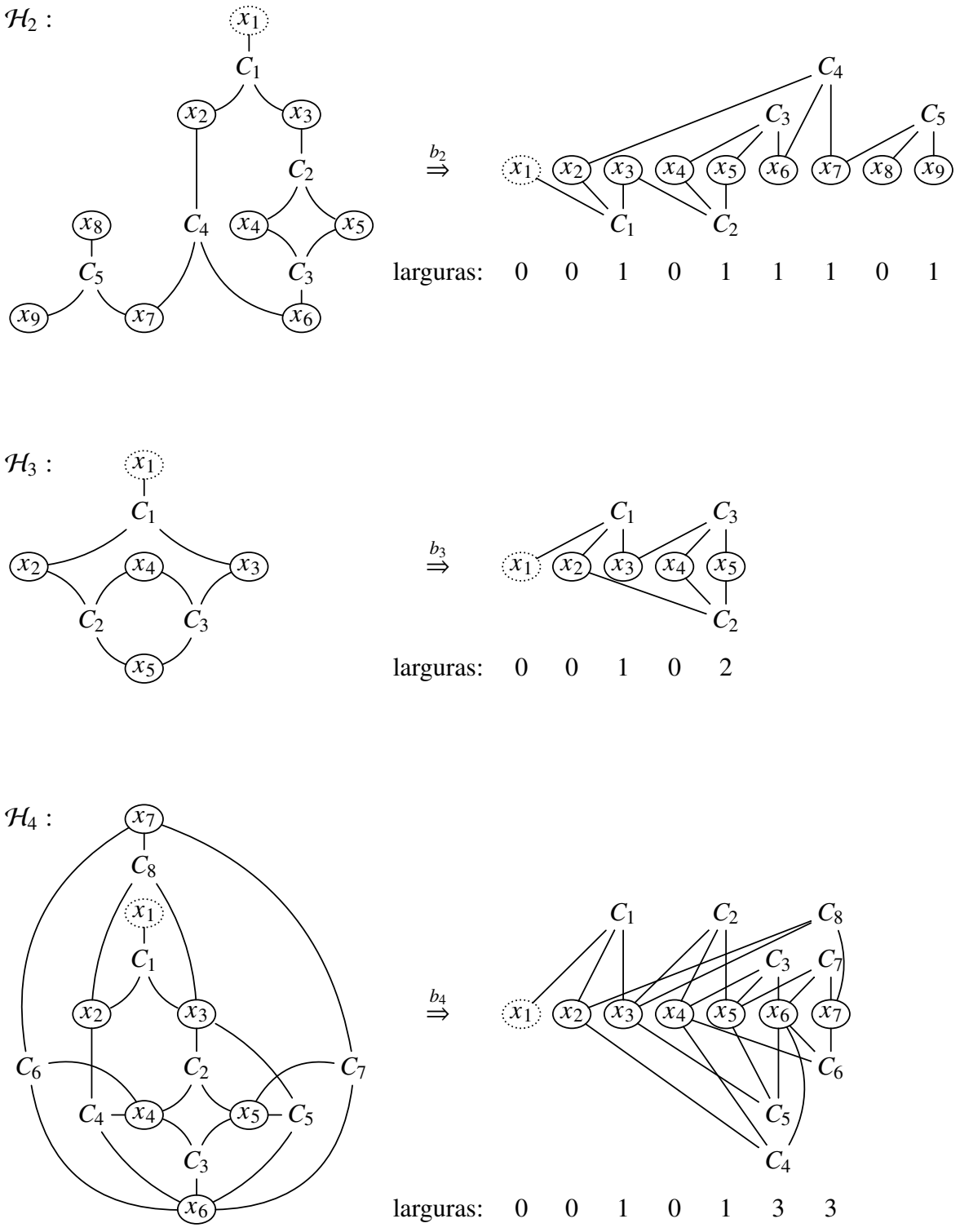


Figura 4.16: Hipergrafos $\mathcal{H}_2$, $\mathcal{H}_3$ e $\mathcal{H}_4$ ordenados e a largura epífita de cada vértice na respectiva ordenação.

(⇒) Para cada C_k em d, coloca-se em b os vértices de C_k que ainda não estão em b, de forma que o último vértice inserido é o vértice de grau 1 no hipergrafo parcial com arestas $\{C_1, \ldots, C_k\}$ (tal vértice existe pois C_k tem grau de interseção no máximo 2). Os dois primeiros vértices, se inseridos em b, têm largura epífita 0, pois nenhum deles é o último vértice de C_k. Pelo mesmo motivo, se eles já estão em b, então a sua largura epífita não se altera. Por outro lado, o terceiro vértice x está sendo inserido em b pela primeira vez, pois x tem grau 1 nesta sub-ordem. Como x é o último vértice de C_k em b, então a sua largura epífita é 1.

(⇐) Para cada x_k em b, marca-se x_k como processado. Se todos os vértices de uma aresta C_i que ainda não está em d são marcados, C_i é colocada em d. Cada aresta C_i inserida em d possui pelo menos um vértice x de grau 1 nesta sub-ordem, pois, do contrário, x já deveria estar marcado. Logo, $|C_i \cap \bigcup_{j=1}^{i-1} C_j| < 3$.

□

Definição 4.9 (Decomposição k-Epífita). Dado um hipergrafo $\mathcal{H} = (V, E)$ e um vértice $x_1 \in V$, uma decomposição k-Epífita segundo x_1 de $\mathcal{H}$ é uma quadrupla $(\mathcal{A}, \Omega, t, k)$, onde $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ é um conjunto ordenado de hipergrafos disjuntos $\mathcal{A}_i = (V_i, E_i)$ tais que $\bigcup_{i=1}^{n} V_i = V$, $\Omega = E \setminus \bigcup_{i=1}^{n} E_i$ é um conjunto de arestas e $t : \Omega \mapsto V_\Omega$ é uma função que associa cada aresta $C_i \in \Omega$ com um de seus vértices $t(C_i) \in C_i$ tal que:

1. $\mathcal{A}_1$ é uma árvore enraizada em x_1;
2. $\forall \mathcal{A}_{i>1} \in \mathcal{A}$: existe no máximo k arestas $\{C_{i_1}, C_{i_2}, \ldots, C_{i_{k'}}\} \subseteq \Omega$, $k' \leq k$, tais que $t(C_{i_1}) = t(C_{i_2}) = \cdots = t(C_{i_{k'}}) \in V_i$ e $\mathcal{A}_i$ é uma árvore enraizada em $t(C_{i_1})$ (ou qualquer outro vértice, se tais arestas não existirem);
3. $\forall C_i \in \Omega$: se $t(C_i) \in V_i$, então $C_i \setminus \{t(C_i)\} \subseteq \bigcup_{j=1}^{i-1} V_j$.

Em particular, para $k = 1$ as Definições 4.4 e 4.9 são idênticas. Isto é, decomposição Epífita e decomposição 1-Epífita são conceitos equivalentes.

Teorema 4.7. *Um hipergrafo $\mathcal{H}$ possui uma decomposição k-Epífita segundo x_1 se, e somente se, tem largura epífita no máximo k em relação a x_1.*

Demonstração.

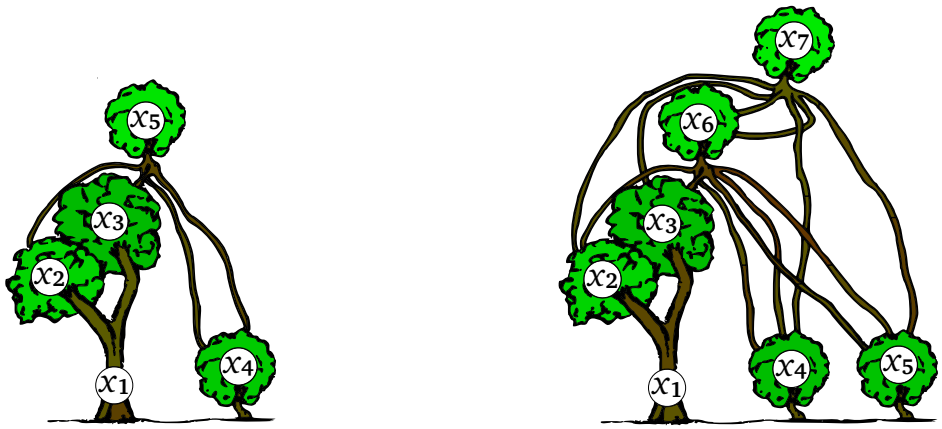
(⇒) Seja $(\mathcal{A}, \Omega, t, k)$ uma decomposição k-Epífita, onde $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$. Como toda árvore $\mathcal{A}_i = (V_i, E_i)$ possui uma decomposição Epífita (Teorema 4.3), então, pelo Teorema 4.6, possui também uma ordenação de vértices b_i com largura epífita 1 (basicamente, é a ordenação topológica da árvore), onde o primeiro vértice na ordenação b_i tem largura epífita 0. Como essas árvores são disjuntas entre si, a ordenação obtida

ao combinar todas as ordenações b_i, $1 \leq i \leq n$, também possui largura epífita 1. Basta provar que ao considerar as arestas de Ω nesta nova ordenação a sua largura epífita não será maior que k. Seja $\{C_{i_1}, \ldots, C_{i_{k' \leq k}}\} \subseteq \Omega$ e $\mathcal{A}_i \in \mathcal{A}$ a árvore tal que $t(C_{i_1}) \in V_i$. O vértice $t(C_{i_1})$ tem largura epífita 0 em b_i, pois é o primeiro vértice da ordenação seguindo a ordem topológica de $\mathcal{A}_i$. Pela condição 3 da Definição 4.9, $C_i \setminus \{t(C_i)\} \subseteq \bigcup_{j=1}^{i-1} V_j$, para todo $C_i \in \Omega$. Logo, ao considerar as arestas $\{C_{i_1}, \ldots, C_{i_{k' \leq k}}\}$, o vértice $t(C_i)$ passa a ter largura epífita no máximo k, pois é o (único) último vértice dessas restrições. E portanto, a ordenação tem largura epífita no máximo k.

($\Leftarrow$) Seja $\mathcal{H}$ um hipergrafo e $b = (x_1, \ldots, x_n)$ uma ordenação de vértices de $\mathcal{H}$ com largura epífita no máximo k. Ao desconsiderar as arestas de $\mathcal{H}$ que causam a um vértice ter largura epífita maior que 1, b torna-se uma ordenação com largura epífita no máximo 1 e, pelo Teorema 4.6, garante a existência de uma decomposição Epífita $(\mathcal{A}, \Omega, t)$, onde $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$. É possível obter uma decomposição k-Epífita a partir desta decomposição Epífita. Seja x_i um vértice que possui arestas que foram desconsideradas, isto é, x_i é o último vértice de mais de uma aresta em $\mathcal{H}$, segundo b. Sejam $C_{i_2}, \ldots, C_{i_{k' \leq k}}$ as arestas de x_i desconsideradas. Se existe uma aresta $C_i \in \Omega$ tal que $t(C_i) = x_i$, então basta acrescentar as arestas $C_{i_2}, \ldots, C_{i_{k' \leq k}}$ ao conjunto Ω e definir que $t(C_i) = t(C_{i_2}) = \cdots = t(C_{i_{k'}})$. Como $k' \leq k$, essa alteração define uma decomposição k-Epífita. Por outro lado, se não existe $C_i \in \Omega$ tal que $t(C_i) = x_i$, então x_i não é raiz da árvore $\mathcal{A}_i$ à qual pertence e é o último vértice de uma aresta C_{i_1} em $\mathcal{A}_i$. Seja $(\mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_l})$ um conjunto ordenado de árvores obtidas ao remover-se a aresta C_{i_1} de $\mathcal{A}_i$, onde $C_{i_1} = \{x_{i_1}, \ldots, x_{i_l}\}$ tem aridade l, $x_{i_j} \in V_{i_j}$, para todo $1 \leq j \leq l$ e a ordenação dessas árvores é definida segundo a ordenação de $x_{i_1}, \ldots, x_{i_l}$ em b (como x_i é o último vértice de C_{i_1}, então $x_i = x_{i_l}$). Além disso, considera-se que cada árvore $\mathcal{A}_{i_j}$ está enraizada em x_{i_j}, para todo $1 < j \leq l$. Seja o conjunto ordenado de árvores $(\mathcal{A}_1, \ldots, \mathcal{A}_{i-1}, \mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_l}, \mathcal{A}_{i+1}, \ldots, \mathcal{A}_n)$ e o conjunto Ω acrescido das arestas $C_{i_1}, \ldots, C_{i_{k'<k}}$ de x_i (incluindo as desconsideradas em b), tais que $t(C_{i_1}) = \cdots = t(C_{i_{k'}}) = x_i$. Para verificar que esses conjuntos constituem uma decomposição k-Epífita, basta observar que as árvores $\mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_{l-1}}$ não possuem arestas no conjunto Ω e que, para $\mathcal{A}_{i_l}$, existem $C_{i_1}, \ldots, C_{i_{k'<k}}$ arestas em Ω tais que $t(C_{i_1}) = \cdots = t(C_{i_{k'}}) = x_i \in V_{i_l}$ e, como x_i é o último vértice destas arestas em b, então $C_{i_j} \setminus \{t(C_{i_j})\} \subseteq \bigcup_{k=1}^{i-1} V_k \cup \bigcup_{k=1}^{l-1} V_{i_k}$, para todo $1 \leq j \leq k' \leq k$.

□

Os Teoremas 4.6 e 4.7 estabelecem que decomposições k-Epífitas são uma generalização de decomposições Epífitas, e podem ser utilizadas para classificar instâncias NCOP de acordo com a sua largura epífita.



(a) (b)

Figura 4.17: Ilustrações de decomposições 2-Epífita (a) e 3-Epífita (b). De forma ilustrativa, em uma decomposição k-Epífita existem $2k$ raizes que brotam de árvores antecessoras.

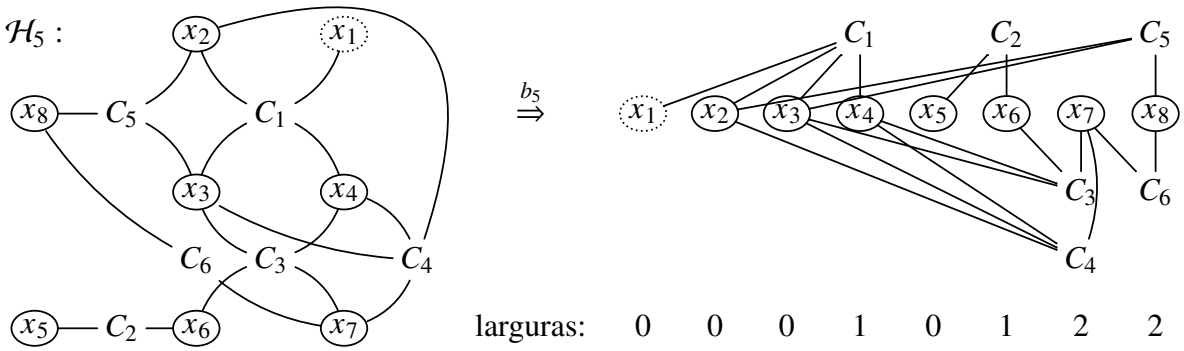
Exemplo 4.8. Os hipergrafos $\mathcal{H}_2$, $\mathcal{H}_3$ e $\mathcal{H}_4$ da Figura 4.16 possuem decomposições, respectivamente, 1-Epífita, 2-Epífita e 3-Epífita segundo x_1. Por exemplo, $\mathcal{H}_3$ possui a decomposição $((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3), \Omega, t, 2)$, tal que $\mathcal{A}_1 = (\{x_1, x_2, x_3\}, \{C_1\})$, $\mathcal{A}_2 = (\{x_4\}, \emptyset)$ e $\mathcal{A}_3 = (\{x_5\}, \emptyset)$, onde $\Omega = \{C_2, C_3\}$ e $t(C_2) = t(C_3) = x_5$. A Figura 4.17 ilustra a estrutura das decomposições 2-Epífita e 3-Epífita dos hipergrafos $\mathcal{H}_3$ e $\mathcal{H}_4$. △

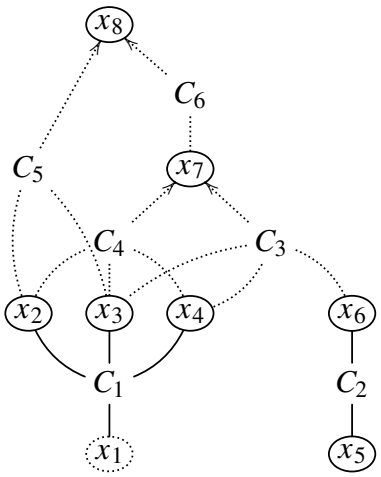
Nota-se que o conceito de largura epífita, que garante a existência de uma decomposição k-Epífita, é bem definido mesmo para restrições k-árias, para qualquer $k \geq 2$. Do mesmo modo, a definição de decomposição k-Epífita é válida para redes k-árias. Assim, essa forma de decomposição não depende da representação ternária da instância original, apenas da adição da restrição $x_1 = f(x)$ representando a função objetivo. Por exemplo, a Figura 4.18 apresenta o hipergrafo $\mathcal{H}_5$ de uma rede quaternária de largura epífita 2, juntamente com uma decomposição 2-Epífita de altura epífita $|\Omega| = 4$ desse hipergrafo.

Teorema 4.8. *Uma rede $\mathcal{R}$ possui uma decomposição k-Epífita $\mathcal{H}$ segundo $x_1 \in C_1$ se, e somente se, todo hipergrafo parcial $\mathcal{H}'$ de $\mathcal{H}$ com mais de uma aresta possui pelo menos um vértice $x' \notin C_1$ de grau no máximo k.*

Demonstração.

(⇒) A prova é por contradição. Seja $\mathcal{H}$ um hipergrafo que possui uma decomposição k-Epífita segundo $x_1 \in C_1$ e seja $b = (x_1, \ldots, x_n)$ uma ordenação de vértices de $\mathcal{H}$ de largura epífita no máximo k. Supõe-se que existe um hipergrafo parcial $\mathcal{H}'$ de $\mathcal{H}$ com mais de uma aresta no qual todo vértice $x' \notin C_1$ tem grau maior que k. Seja x_i o vértice de $\mathcal{H}'$ com maior índice em b ($x_i \notin C_1$). Como todos os vértices $x' \notin C_1$ têm





$$\mathcal{A}_1 = (\{x_1, x_2, x_3, x_4\}, \{C_1\})$$
$$\mathcal{A}_2 = (\{x_5, x_6\}, \{C_2\})$$
$$\mathcal{A}_3 = (\{x_7\}, \emptyset)$$
$$\mathcal{A}_4 = (\{x_8\}, \emptyset)$$
$$\Omega = \{C_3, C_4, C_5, C_6\}$$
$$t(C_3) = t(C_4) = x_7$$
$$t(C_5) = t(C_6) = x_8$$

Figura 4.18: Hipergrafo $\mathcal{H}_5$ de uma rede quaternária, uma ordenação de vértices de largura epífita 2 e uma decomposição 2-Epífita de altura epífita 4 deste hipergrafo.

grau maior que k em $\mathcal{H}'$, então x_i pertence a $l > k$ arestas nesse hipergrafo parcial e, além disso, é o último vértice dessas l arestas em d. Logo, x_i tem largura epífita $l > k$, contradizendo o fato de que $\mathcal{H}$ possui uma decomposição k-Epífita.

($\Leftarrow$) Seja $\mathcal{H}$ um hipergrafo com vértices $\{x_1, \ldots, x_n\}$ tal que todo hipergrafo parcial $\mathcal{H}'$ de $\mathcal{H}$ com mais de uma aresta tem pelo menos um vértice $x' \notin C_1$ de grau no máximo k. Seja $(\mathcal{H}_1, \ldots, \mathcal{H}_n = \mathcal{H})$ uma sequência de hipergrafos tal que $\mathcal{H}_1$ possui apenas o vértice x_1 e, para todo $1 < i \leq n$, o hipergrafo $\mathcal{H}_{i-1}$ é obtido removendo-se de $\mathcal{H}_i$ um vértice x_i de grau 0 ou, se este vértice não existir, um vértice x_i de grau no máximo k em $\mathcal{H}_i$ (este vértice garantidamente existe, pois, nesse caso, $\mathcal{H}_i$ é um hipergrafo parcial de $\mathcal{H}$). A sequência de vértices removidos é tal que x_i é o último vértice de no máximo k arestas em $\mathcal{H}_i$ e, consequentemente, também em $\mathcal{H}$, pois as demais arestas às quais x_i pertence também possuem outros vértices em $\mathcal{H}_j$, para algum $i < j \leq n$. Logo, a sequência $(x_1, \ldots, x_n)$ tem largura epífita k e, portanto, $\mathcal{H}$ possui uma decomposição k-Epífita.

□

O Teorema 4.8 é uma generalização do Teorema 4.4. Assim, o método `ordem_de_grau_2` (Algoritmo 18) pode ser facilmente modificado para encontrar decomposições k-Epífitas, alterando a condição da linha 3 para que verifique a existência de um vértice de grau no máximo k. De forma geral, ao escolher sempre o vértice de grau mínimo que não pertence à aresta C_1, obtém-se uma ordenação de largura epífita mínima l, onde l é o maior grau dentre os vértices escolhidos em cada passo do algoritmo. Essa generalização é o próprio algoritmo `min-width` proposto em Freuder (1978), mas aplicado em hipergrafos e com a restrição de que vértices da aresta que contém a variável raiz da rede não devem ser escolhidos na construção da ordenação. Essa forte relação entre os dois algoritmos surge do fato de que a largura epífita é a extensão natural da largura de grafos. O Algoritmo 20 descreve a extensão do Algoritmo 18, devolvendo uma ordenação de vértices de largura epífita mínima k e, se esta largura for 1, uma ordenação de arestas com grau de interseção no máximo 2.

Teorema 4.9. *Um problema de otimização codificado por uma rede $\mathcal{R}$ que possui uma decomposição k-Epífita é resolvido sem retrocesso se $\mathcal{R}$ é fortemente k-consistente relacional direcionado.*

Demonstração. Seja $b = (x_1, \ldots, x_n)$ uma ordenação de variáveis de $\mathcal{R}$ com largura epífita k. Toda variável x_l é a última variável de no máximo k restrições $R_{C_1}, \ldots, R_{C_{k'}}$, $k' \leq k$. Uma valoração consistente de $(x_1, \ldots, x_{l-1})$ pode ser estendida a x_l se toda valoração consistente de $\bigcup_{j=1}^{k'} C_j \setminus \{x_l\} \subseteq \{x_1, \ldots, x_{l-1}\}$ puder ser estendida a x_l, o que é satisfeito por k-consistência relacional direcionada forte. □

Algoritmo 20 Ordem de largura epífita k (`ordem_de_largura_epifita_k`$(\mathcal{H}, C_1, x_1)$)

Entrada: Hipergrafo $\mathcal{H} = (V, E)$, aresta C_1 e vértice $x_1 \in C_1$.

Saída: Ordenação de vértices b iniciada por x_1 com largura epífita k e, caso k seja igual a 1, ordenação de arestas d iniciada por C_1 com grau de interseção no máximo 2.

1: $d \leftarrow \emptyset$
2: $b \leftarrow \emptyset$
3: $k \leftarrow 1$
4: **enquanto** $E \neq \{C_1\}$ **faça**
5: seja $x \in V$ um vértice de grau mínimo l em $\mathcal{H}$ tal que $x \notin C_1$
6: **se** $l > k$ **então**
7: $k \leftarrow l$
8: **fim se**
9: **para todo** $C_i \in E$ tal que $x \in C_i$ **faça**
10: `empilha`(C_i, d)
11: $E \leftarrow E \setminus \{C_i\}$
12: **fim para**
13: **para todo** $x \in V$ de grau 0 em $\mathcal{H}$ **faça**
14: `empilha`(x, b)
15: $V \leftarrow V \setminus \{x\}$
16: **fim para**
17: **fim enquanto**
18: `empilha`(C_1, d)
19: **para todo** $x_i \in C_1 \setminus \{x_1\}$ **faça**
20: `empilha`(x_i, b)
21: **fim para**
22: `empilha`(x_1, b)
23: **devolve** (d, b, k)

O Teorema 4.9 estabelece uma condição suficiente para otimização global sem retrocesso, pois toda instância NCOP possui uma decomposição k-Epífita, para algum $k \geq 1$. Além disso, classifica tais problemas em níveis de complexidade com base na estrutura do hipergrafo de restrições (largura epífita).

No caso específico de instâncias representadas por redes ternárias, o método de ramificação e poda proposto para resolver instâncias que possuem decomposições Epífitas (Algoritmo 19) pode ser facilmente adaptado para resolver instâncias com decomposições k-Epífitas. Nesse caso, existem subconjuntos de restrições em Ω que possuem o mesmo valor na função t e, para essas restrições, RAC direcionada deve ser alcançada simultaneamente. Vale lembrar que alcançar RAC direcionada de forma exata exige a adição de novas restrições na rede e, consequentemente, altera a sua largura epífita. Na prática, isso implica que o nível de consistência relacional direcionada que garante solução sem retrocesso é dinâmica e aumenta conforme essa consistência é alcançada. Por exemplo, na Figura 4.10 apresentou-se uma decomposição 1-Epífita de $\mathcal{H}$ que não é RAC direcionada sob os domínios dados no Exemplo 4.6; no entanto, a adição de uma restrição binária R_{C_5} na rede, a fim de alcançar RAC direcionada, altera a estrutura do hipergrafo e, consequentemente, a sua largura epífita. A saber, o hipergrafo $\mathcal{H}'$ da Figura 4.10 possui uma decomposição 2-Epífita dada por $((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4), \Omega, t, 2)$, onde $\mathcal{A}_1 = (\{x_1, x_2, x_3, x_6\}, \{C_1, C_5\})$, $\mathcal{A}_2 = (\{x_5\}, \emptyset)$, $\mathcal{A}_3 = (\{x_4\}, \emptyset)$, $\mathcal{A}_4 = (\{x_7\}, \emptyset)$, $\Omega = \{C_2, C_3, C_4\}$, $t(C_2) = t(C_3) = x_4$ e $t(C_4) = x_7$. Nesse caso, RAC direcionada não é mais condição suficiente para encontrar a solução ótima sem retrocesso, sendo necessário alcançar 2-consistência relacional direcionada forte. Lembra-se que o mesmo problema ocorre com os resultados de Freuder (1982) e Jégou (1993). Por outro lado, evita-se a adição de novas restrições na rede ao aproximar a consistência por um fator $\varepsilon_{AR} > 0$, utilizando um método de redução de domínios como o Algoritmo 19.

4.6 Relação com larguras e outras abordagens

Sam-Haroud e Faltings (1996) estabeleceram uma relação entre redes numéricas ternárias e busca sem retrocesso: uma rede ternária convexa $(3, 2)$-consistente relacional é livre de retrocesso. Além da condição de convexidade, que o método aqui proposto não exige, o principal objetivo do trabalho de Sam-Haroud e Faltings era construir uma descrição compacta de todas as soluções da rede, ao invés da valoração ótima da rede.

O método de corte de ciclo, proposto por Dechter e Pearl (1987a), é semelhante ao método de otimização relaxada aqui proposto. Em particular, em uma decomposição Epífita $(\mathcal{A}, \Omega, t)$, as restrições do conjunto Ω são aquelas que, ao serem removidas da rede, resultam em um hipergrafo de restrições Berge-acíclico. Do mesmo modo, a busca com retrocesso ocorre apenas em variáveis de restrições desse conjunto, enquanto as demais são valoradas, quando possível, por GAC direcionada. Por um lado, o método aqui proposto pode ser visto como uma

aplicação do método de corte de ciclo para redes não binárias no cenário contínuo. Por outro lado, existem algumas diferenças fundamentais entre as duas abordagens:

1. no cenário contínuo, o método de ramificação e poda encontra subdomínios consistentes ao invés de valorações de variáveis. Assim, enquanto no método de corte de ciclo a tentativa de valoração da parte acíclica da rede ocorre sempre que uma valoração consistente no conjunto de corte de ciclo é encontrada, no método `RAC_direcionada_aprox` essa tentativa de valoração ocorre em toda iteração da busca;

2. no método de corte de ciclo, as variáveis são valoradas do centro da rede (variáveis que compõem um ciclo) às bordas. No método aqui proposto, essa estratégia não é eficiente, pois o algoritmo baseia-se em iniciar a valoração pela atribuição $\langle x_1 = \min D_1 \rangle$, onde x_1 é a raiz da decomposição Epífita. Dessa forma, a busca ocorre seguindo uma ordenação fixa de variáveis que atravessa o conjunto Ω, isto é, as variáveis que compõem os ciclos da rede, diversas vezes;

3. como o problema aqui abordado é de otimização global, não basta encontrar a primeira solução da rede, mas aquela que é ótima segundo a função objetivo. Nesse sentido, o método de corte de ciclo usual efetuaria uma simples enumeração de todas as soluções da rede, sem tomar vantagem de cortes no espaço de busca através de limitantes superiores à função objetivo ou heurísticas de solução ótima, como ocorre no método `RAC_direcionada_aprox`.

A decomposição Epífita também assemelha-se à decomposição em árvore, apresentada na Seção 3.5.3: cada árvore do conjunto $\mathcal{A}$ pode ser vista como um subproblema, e estes são conectados por restrições do conjunto Ω. No entanto, diferentemente da decomposição em árvore, estas restrições não são de igualdade, mas restrições numéricas arbitrárias. Além disso, a estrutura obtida ao representar cada subproblema por uma variável não é necessariamente uma árvore (e remover arestas redundantes não é trivial, já que essas arestas representam restrições numéricas).

Em geral, pode-se comparar a estrutura de decomposições k-Epífitas com outras definições de largura, visto que a largura epífita também é um parâmetro de ciclicidade do hipergrafo de restrições. Em particular, a definição de largura de Freuder aplica-se apenas a grafos, enquanto outras noções como largura de hipergrafo, largura de árvore e largura de hiperárvore são definidas também para hipergrafos (ver Seção 3.5.2).

Teorema 4.10. *A largura do grafo primal ou do grafo dual de uma rede de restrições $\mathcal{R}$ não caracteriza a largura epífita da rede.*

Demonstração. A Figura 4.19 apresenta os grafos primais de dois hipergrafos $\mathcal{H}_1$ e $\mathcal{H}_2$, e os grafos duais de dois hipergrafos $\mathcal{H}_3$ e $\mathcal{H}_4$. Enquanto $\mathcal{H}_1$ tem largura epífita 1 e $\mathcal{H}_2$ tem largura epífita 2, ambos possuem o mesmo grafo primal. Da mesmo modo, $\mathcal{H}_3$ tem largura epífita 1 e $\mathcal{H}_4$ tem largura epífita 2, mas possuem o mesmo grafo dual. □

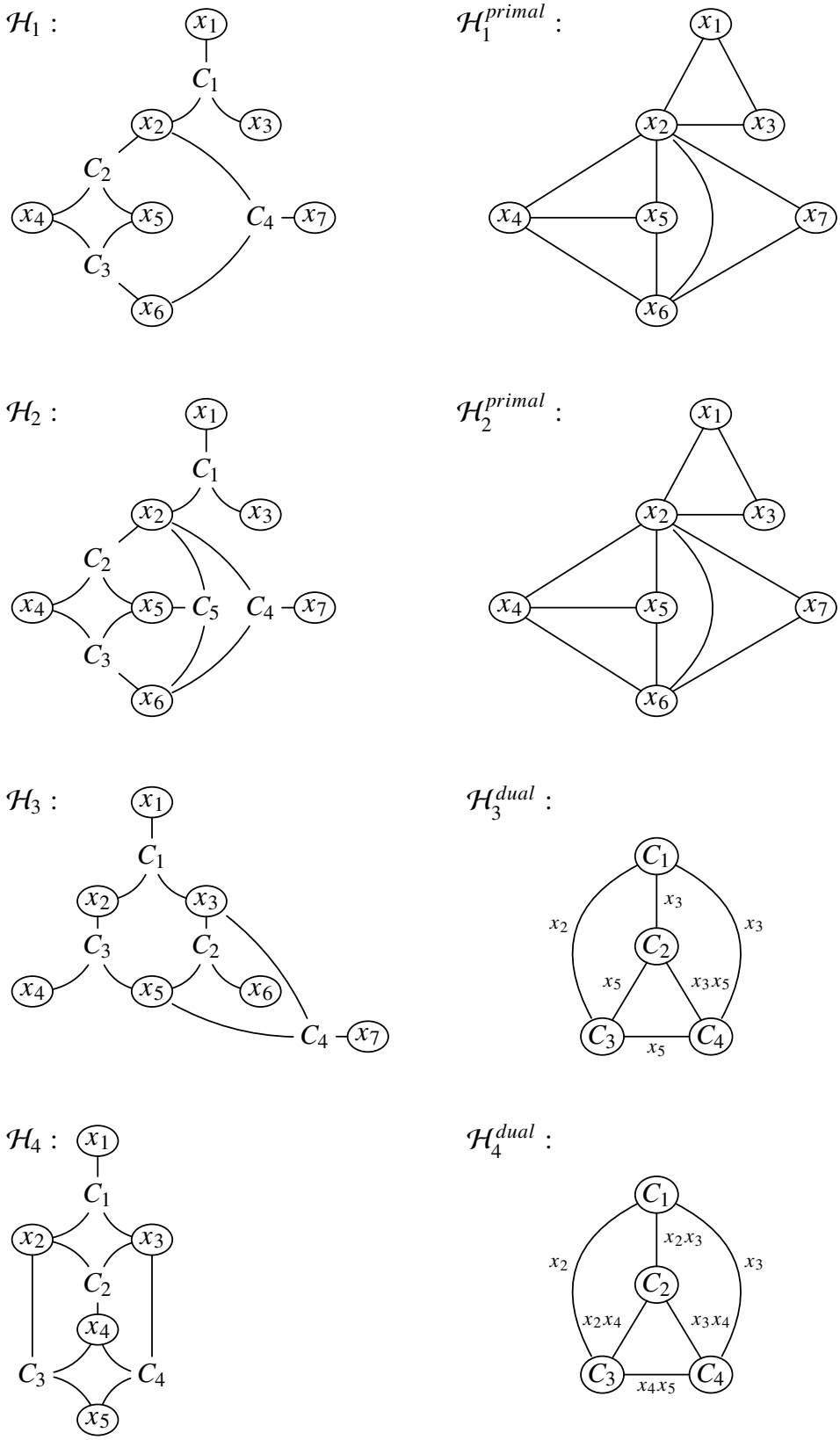


Figura 4.19: Hipergrafos $\mathcal{H}_1$ e $\mathcal{H}_3$ com larguras epífita 1 e hipergrafos $\mathcal{H}_2$ e $\mathcal{H}_4$ com larguras epífita 2. Os hipergrafos $\mathcal{H}_1$ e $\mathcal{H}_2$ possuem o mesmo grafo primal, enquanto os hipergrafos $\mathcal{H}_3$ e $\mathcal{H}_4$ possuem o mesmo grafo dual.

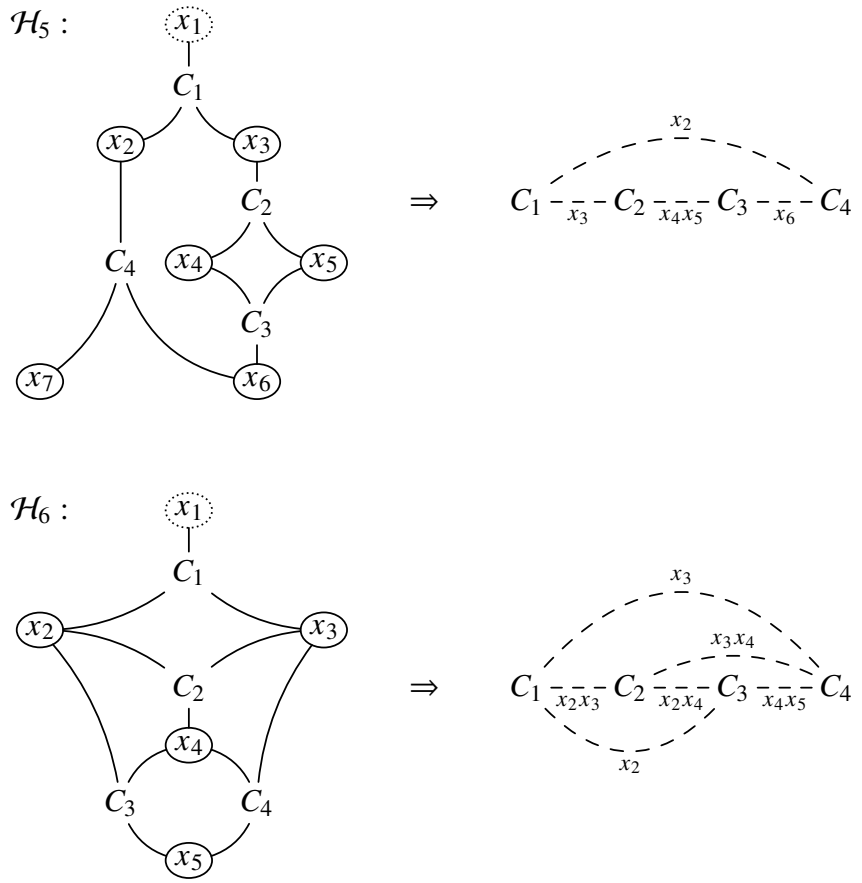


Figura 4.20: Hipergrafo $\mathcal{H}_5$ com largura epífita 1 e hipergrafo $\mathcal{H}_6$ com largura epífita 2, e as interseções de cada aresta no respectivo hipergrafo. A única aresta com um número de interseções maximais menor que 2 é a aresta C_1 do hipergrafo $\mathcal{H}_6$; mas, após ela, toda aresta deste hipergrafo tem 2 interseções maximais.

Teorema 4.11. *A largura de hipergrafo de uma rede $\mathcal{R}$ não caracteriza a largura epífita da rede.*

Demonstração. A Figura 4.20 apresenta um hipergrafo $\mathcal{H}_5$ com largura epífita 1 e um hipergrafo $\mathcal{H}_6$ com largura epífita 2, e as respectivas interseções de arestas de cada hipergrafo, representadas por arestas tracejadas (sem considerar uma ordenação de arestas fixa). A largura de hipergrafo de uma aresta, em uma ordenação d, é o número de interseções maximais desta aresta com antecessoras em d. Pode-se notar que, exceto por C_1 em $\mathcal{H}_6$, todas as arestas, em ambos os hipergrafos, têm pelo menos duas interseções maximais. Logo, para qualquer ordenação desses hipergrafos, a última aresta tem largura pelo menos 2, exceto se esta for a aresta C_1 em $\mathcal{H}_6$. Para este caso, no entanto, a penúltima aresta garantidamente tem largura 2. Dessa forma, as larguras de hipergrafo de $\mathcal{H}_5$ e $\mathcal{H}_6$ são ambas 2. □

Teorema 4.12. *As larguras de árvore e de hiperárvore de uma rede $\mathcal{R}$ não caracterizam a largura epífita da rede.*

Demonstração. Os hipergrafos $\mathcal{H}_3$ e $\mathcal{H}_4$ da Figura 4.19 possuem o mesmo grafo dual, com exceção dos rótulos de arestas que compartilham variáveis. No entanto, é fácil verificar que

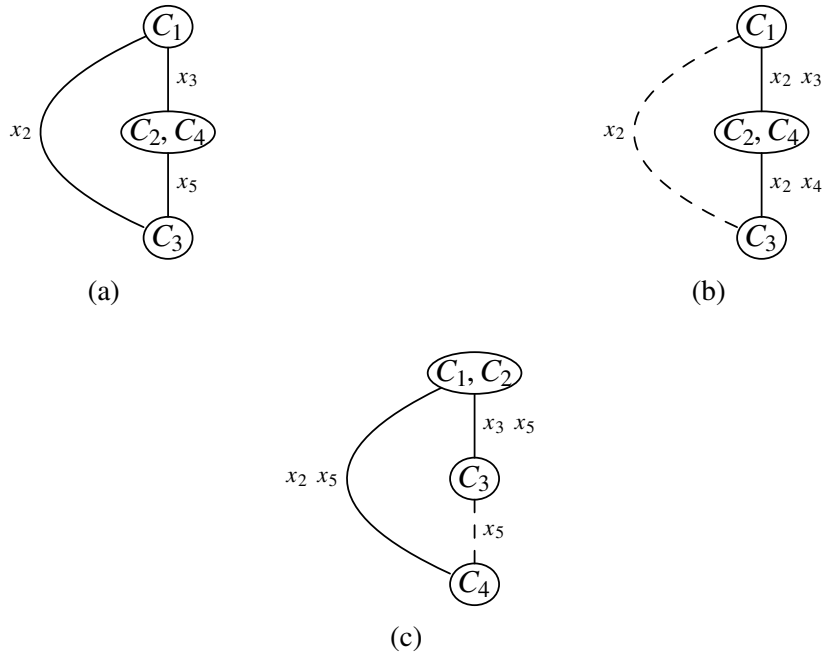


Figura 4.21: Decomposições (a) e (c) do hipergrafo $\mathcal{H}_3$, e decomposição (b) do hipergrafo $\mathcal{H}_4$, ambos representados na Figura 4.19. Apenas (b) e (c) são decomposições em árvore, com larguras de árvore 4 e 2, e larguras de hiperárvore 5 e 2, respectivamente.

em nenhum caso é possível obter uma árvore removendo arestas redundantes. Portanto, ambos os hipergrafos são α-cíclicos (logo, α-ciclicidade não caracteriza a largura epífita de um hipergrafo). Além disso, pode-se considerar também as decomposições em árvore de cada hipergrafo, lembrando que a largura de árvore de uma decomposição é o maior número de variáveis associados a um vértice da árvore ($\max_{v \in V} |\chi(v)|$), enquanto a largura de hiperárvore é o maior número de restrições associadas ($\max_{v \in V} |\psi(v)|$). Como estes hipergrafos não possuem árvores de junção, é preciso agrupar pelo menos duas arestas em um subproblema. A saber, o agrupamento que minimiza a largura de árvore em ambos os hipergrafos é $\{C_2, C_4\}$, pois são as arestas que compartilham o maior número de vértices. A Figura 4.21 apresenta as decomposições dos hipergrafos $\mathcal{H}_3$ (a) e $\mathcal{H}_4$ (b) com esse agrupamento. Enquanto a decomposição do primeiro hipergrafo não é uma árvore, para $\mathcal{H}_4$ obtém-se uma decomposição em árvore com largura de árvore 4 e largura de hiperárvore 2. Com um agrupamento distinto $\{C_1, C_2\}$, por exemplo, obtém-se uma decomposição em árvore de $\mathcal{H}_3$ com largura de árvore 5 e largura de hiperárvore 2 (c). Dessa forma, as larguras de árvore e de hiperárvore não caracterizam a largura epífita de uma rede, pois $\mathcal{H}_3$ tem largura epífita 1 e $\mathcal{H}_4$ tem largura epífita 2. Curiosamente, $\mathcal{H}_4$ possui uma largura de árvore menor que $\mathcal{H}_3$, mas esse não é o caso geral. □

4.7 Conclusão

Problemas de otimização global que possuem decomposições k-Epífitas são resolvidos sem a necessidade de retrocesso se k-consistência relacional direcionada forte for satisfeita. Toda

instância NCOP possui uma decomposição k-Epífita, para algum $k \geq 1$. Em particular, muitas instâncias possuem decomposições 1-Epífitas, nas quais RAC direcionada é condição suficiente para otimização global sem retrocesso. Embora alcançar RAC direcionada seja impraticável no caso geral, pode-se aproximar esta consistência utilizando um método de ramificação e poda. Ao aplicar este método, encontra-se uma aproximação do mínimo global de instâncias codificadas como redes ternárias.

Este capítulo apresentou as principais contribuições desta tese, as quais serão recapituladas na Seção 6.1. De forma geral, apresentou-se a codificação de problemas de otimização como redes ternárias, introduzindo o conceito de raiz de uma rede, e estendeu-se a ideia de busca sem retrocesso de CSPs ao cenário de otimização global. Mostrou-se a relação entre a estrutura de uma instância, caracterizada pelos parâmetros de grau de interseção e largura epífita, e o nível de consistência que garante otimização global sem retrocesso. Enquanto hipergrafos de restrições com grau de interseção no máximo 1 são Berge-acíclicos, e as instâncias representadas por estes hipergrafos podem ser resolvidas alcançando-se GAC direcionada, hipergrafos com grau de interseção 2 possuem largura epífita 1 e possuem, portanto, decomposições Epífitas, nas quais uma combinação de GAC direcionada e RAC direcionada é suficiente para otimização global sem retrocesso. De modo geral, hipergrafos com largura epífita k caracterizam a classe de problemas que possuem decomposições k-Epífitas. Esse parâmetro de largura difere de outros parâmetros propostos na literatura.

O método `RAC_direcionada_aprox` para aproximar RAC direcionada foi proposto, evitando adicionar novas restrições à rede. Esse método é uma variante da ramificação e poda intervalar para resolver NCOP, caracterizando um conjunto de heurísticas de escolha de variáveis na fase de ramificação. No Capítulo 5, serão apresentados detalhes da implementação desse método no otimizador `OGRe` e resultados experimentais de sua aplicação em um conjunto de instâncias de problemas de otimização global.

5 IMPLEMENTAÇÃO E RESULTADOS EXPERIMENTAIS

Neste capítulo, descreve-se a implementação preliminar de um otimizador chamado `OGRe` - Otimização Global Relaxada, fundamentado nos resultados teóricos apresentados no Capítulo 4. O objetivo primário deste otimizador é avaliar a aplicação prática da decomposição Epífita e da consistência de arco relacional direcionada como condição suficiente de otimização global sem retrocesso, embora essa consistência seja alcançada de maneira aproximada considerando uma tolerância ε_{AR} (justificando o termo "relaxada" na nomenclatura do otimizador).

Esse otimizador está escrito em linguagem de programação C, e é dividido em dois módulos principais: uma biblioteca multi-intervalar que implementa operações usuais sobre multi-intervalos, e um módulo de busca que implementa contratores da consistência de arco generalizada e o algoritmo de ramificação e poda para alcançar RAC aproximada, proposto na Seção 4.4.1. Apesar de existir uma série de bibliotecas intervalares no mercado, que incluem a extensão *Intlab* (Rump, 1999) do *software* Matlab, as bibliotecas C++ *Boost* e *Gaol* e, mais recentemente, o pacote de álgebra intervalar do resolvedor *GNU Octave* (Eaton et al., 2017), optou-se pelo desenvolvimento de uma biblioteca própria permitindo a definição de operações específicas à aplicação de contratores GAC de forma simplificada, conforme será abordado na Seção 5.2.2. De modo geral, o otimizador `OGRe`, disponível com código-fonte aberto e licença GNU GPL, em Derenievicz (2018), caracteriza uma contribuição prática desta pesquisa, adicional às contribuições teóricas apresentadas no Capítulo 4.

Este capítulo está divido em quatro seções:

5.1. **Objetivo e limitações**: visão geral do otimizador `OGRe`;

5.2. **Biblioteca multi-intervalar**: detalhes da implementação da biblioteca multi-intervalar;

5.3. **O otimizador `OGRe`**: detalhes da implementação do otimizador;

5.4. **Resultados experimentais**: alguns resultados da execução do otimizador em um conjunto de instâncias de problemas reais de otimização.

5.1 Objetivo e limitações

O objetivo desta implementação é avaliar a aplicação prática dos resultados apresentados no Capítulo 4. Em particular, deseja-se verificar a decomposição Epífita de instâncias de problemas reais e a eficiência do método proposto para aproximar a consistência de arco relacional direcionada. De modo geral, o otimizador `OGRe` implementa o método `RAC_direcionada_aprox` (Algoritmo 19), sem considerar técnicas adicionais para acelerar a busca ou podar o espaço de busca de maneira mais eficiente, como técnicas de busca local, aprendizado por conflitos, retrocesso não cronológico, heurísticas, entre outros, usualmente implementados em otimizadores intervalares. Dessa forma, não é objetivo desta implementação competir com otimizadores do estado da arte, seja em relação a tempo de execução ou exatidão dos resultados obtidos. Por outro lado, espera-se que os métodos aqui propostos possam ser utilizados futuramente para incrementar resolvedores intervalares em relação a pré-processamento, busca local ou limitante inferior de solução ótima.

Além das limitações supracitadas, a implementação da biblioteca multi-intervalar considera apenas os operadores $+$, $-$, $\cdot$, $/$, $\hat{}$ e $\sqrt{}$, sendo estes últimos limitados, respectivamente, a expoentes e índices inteiros. Os parâmetros de aproximação ε_{AR} e Δ utilizados no Algoritmo 19 não são trivialmente definidos no caso geral. Nos experimentos realizados, considerou-se um conjunto de execuções com parâmetros distintos para cada instância.

5.2 Biblioteca multi-intervalar

Nesta seção são apresentados detalhes da implementação de uma biblioteca com as principais operações sobre multi-intervalos utilizadas no `OGRe`. De modo geral, as definições seguem de acordo com o introduzido na Seção 2.3, exceto por alguns casos específicos de operações intervalares. Algumas limitações e detalhes de implementação que não são facilmente encontrados na literatura também são elencados.

É importante ressaltar que as operações aqui definidas têm o propósito de manter as técnicas intervalares (sobretudo a consistência de arco generalizada) completas e, de modo algum, caracterizam uma completa extensão da aritmética usual. Para esse contexto, no entanto, essas definições apresentam um funcionamento correto.

5.2.1 Números reais

O padrão IEEE 754 normaliza a representação, arredondamento e aritmética de números reais na forma de **pontos flutuantes**. Em particular, define operações aritméticas sobre o conjunto estendido dos números reais, conforme especificado na equação (2.6); isto é, os símbolos $-\infty$ e $+\infty$ também são representados. Em particular, a linguagem C adota a representação de pontos flutuantes nos tipos `float` (32 bits) e `double` (64 bits) do formato IEEE 754, bem como a sua

padronização de aritmética em $\overline{\mathbb{R}}$. Na implementação do `OGRe` define-se um número real como um ponto flutuante do tipo `double`.

Devido à finitude de representação de números reais, utiliza-se uma tolerância ϵ na comparação de pontos flutuantes. Assim, dois números a e b são considerados iguais se $|a-b| \leq \epsilon$. Além disso, a distribuição dos números em ponto flutuante não é uniforme: quanto mais longe de zero, seja em direção a $-\infty$ ou $+\infty$, menos números são representados e, consequentemente, maior a distância entre eles. Dessa forma, uma tolerância ϵ que seja adequada para números a e b próximos de zero não satisfaz a condição $|a - b| \leq \epsilon$ para valores grandes. Alternativamente, define-se também uma tolerância relativa ϵ_R na comparação (nota-se que, da mesma forma, uma tolerância relativa adequada para números de alta magnitude não serve para valores próximos de zero). Mais detalhes de implementação da comparação entre pontos flutuantes podem ser encontrados em Dawson (2012).

De modo geral, define-se que:

$$a \overset{\epsilon}{=} b \Leftrightarrow \begin{cases} a = -\infty \text{ e } b = -\infty \textbf{ ou} \\ a = \infty \text{ e } b = \infty \textbf{ ou} \\ |a - b| \leq \epsilon \textbf{ ou} \\ |a - b| \leq \epsilon \cdot \max\{|a|, |b|\} \end{cases} \qquad a \overset{\epsilon}{<} b \Leftrightarrow \begin{cases} a = -\infty \text{ e } b = \infty \textbf{ ou} \\ a \overset{\epsilon}{\neq} b \text{ e } a < b \end{cases} \tag{5.1}$$

Por simplificação, a equação (5.1) utiliza ϵ para indiciar tanto a tolerância absoluta quanto a relativa, mas estes valores podem ser distintos. Na implementação do `OGRe`, define-se ϵ através da constante `DBL_EPSILON`, que fornece a diferença entre o ponto flutuante 1.0 e o próximo número real representado.

Além das operações sobre $\overline{\mathbb{R}}$ usuais, define-se as seguintes operações:

$$-\infty + (+\infty) = +\infty + (-\infty) = 0 \tag{5.2}$$

$$-\infty - (-\infty) = +\infty - (+\infty) = 0 \tag{5.3}$$

$$\pm\infty \cdot 0 = 0 \cdot (\pm\infty) = 0 \tag{5.4}$$

$$x/0 \in \{-\infty, +\infty\} \qquad \text{se } x \neq 0 \tag{5.5}$$

$$0/x = 0 \qquad \text{para todo } x \in \overline{\mathbb{R}} \tag{5.6}$$

$$x/(\pm\infty) = 0 \qquad \text{para todo } x \in \overline{\mathbb{R}} \tag{5.7}$$

As definições (5.2) e (5.3) são utilizadas para computar a distância entre os extremos de dois intervalos X_1 e X_2 tais que $\underline{X_1} = \underline{X_2} = -\infty$ ou $\overline{X_1} = \overline{X_2} = +\infty$. A definição (5.4), como citado na Seção 2.3.1, é utilizada para efetuar a multiplicação intervalar. A definição (5.5) continua definindo parcialmente a divisão por zero; no padrão IEEE 754 esta expressão é definida como $x/0 = +\infty$, mas há casos em que deseja-se que o resultado seja negativo[1].

[1]Em Hansen e Walster (2004), define-se, através do conceito de *cset*, que $x/0 = \{-\infty, +\infty\}$.

As definições (5.6) e (5.7) são utilizadas para simplificar a implementação da divisão intervalar, conforme será abordado na Seção 5.2.2. Em particular, nota-se que essas definições implicam $\pm\infty/\pm\infty = 0$ e $0/0 = 0$. Embora não sejam usuais[2], estas definições derivam diretamente da divisão intervalar introduzida na Seção 2.3.2.

5.2.2 Intervalos

Um intervalo X é representado pelos seus extremos $\underline{X}$ e $\overline{X}$, e pelos valores $\tau_e(X)$ e $\tau_d(X)$. No `OGRe`:

```
1  #ifndef __INTERVALO__
2  #define __INTERVALO__
3
4  typedef struct t_intervalo *intervalo;
5  struct t_intervalo {
6    int T_e, T_d;
7    real ext_e, ext_d;
8  };
9
10 #endif
```

A aritmética intervalar é utilizada, principalmente, para computar o contrator GAC (Algoritmo 11). Assim, implementa-se estas operações de forma a manter a seguinte equivalência:

$$X_1 \subseteq (X_2 \circ X_3) \Leftrightarrow X_2 \subseteq (X_1 \bullet X_3) \Leftrightarrow X_3 \subseteq (X_1 \diamond X_2) \tag{5.8}$$

Para isso, algumas definições usuais, como as apresentadas na Seção 2.3.2, devem ser ligeiramente modificadas.

Multiplicação

Quando dois intervalos não fechados X_1 e X_2 são operados, o intervalo resultante $Y_1 = X_1 \circ X_2$ possui os valores $\tau_e(Y_1)$ e $\tau_d(Y_1)$ definidos de acordo com os extremos de X_1 e X_2 que computaram Y_1. Por exemplo, $[1, 2) \cdot [1, 4] = [1, 8)$ e $(0, 2] - (1, 4] = (-4, 1)$. No entanto, existe uma exceção para o caso da multiplicação de um intervalo que contém 0: se X_1 e X_2 são dois intervalos não vazios e 0 pertence a $X_1 \cup X_2$, então 0 também deve pertencer a $X_1 \cdot X_2$, pois existem $a_1 \in X_1$ e $a_2 \in X_2$ tais que $a_1 \cdot a_2 = 0$. Por exemplo, $[0, 2] \cdot (1, 3]$ deve resultar no intervalo $[0, 6]$, e não no intervalo $(0, 6]$.

[2]Em Hansen e Walster (2004), por exemplo, define-se que $0/0 = (-\infty, +\infty)$.

Divisão

A divisão intervalar possui diversas exceções, devido às várias indefinições envolvendo esta operação, como $0/0$ e $\pm\infty/\pm\infty$. Alguns exemplos ajudam a identificar estas exceções.

Exemplo 5.1. Seja a restrição $R_C \equiv x_1 = x_2 \cdot x_3$, tal que $D_1 = [0, 0]$, $D_2 = [0, 1]$ e $D_3 = [0, 1]$. A restrição R_C é claramente GAC, pois para qualquer valor tomado em um dos intervalos D_1, D_2 ou D_3, existem valores nos demais domínios para os quais a restrição é satisfeita. No entanto, a aplicação do contrator obtém a seguinte redução:

$$D_1 \leftarrow D_1 \cap D_2 \cdot D_3 = [0, 0] \cap \underbrace{[0, 1] \cdot [0, 1]}_{[0,1]} = [0, 0] \tag{5.9}$$

$$D_2 \leftarrow D_2 \cap D_1/D_3 = [0, 1] \cap \underbrace{[0, 0] \cdot [1, +\infty)}_{[0,0]} = [0, 0] \tag{5.10}$$

$$D_3 \leftarrow D_3 \cap D_1/D_2 = [0, 1] \cap \underbrace{[0, 0]/[0, 0]}_{\emptyset} = \emptyset \tag{5.11}$$

Na equação (5.10), a extensão intervalar de x_1/x_3 resultou no intervalo $[0, 0]/[0, 1] = [0, 0]$ e, consequentemente, excluiu valores consistentes do domínio de x_2, resultando no intervalo vazio no terceiro passo do contrator. △

Exemplo 5.2. Seja a restrição $R_C \equiv x_1 = x_2 \cdot x_3$, tal que D_1 contém 0, D_2 é um intervalo qualquer e $D_3 = [0, 0]$. Como o produto $x_2 \cdot x_3$ é nulo, pois x_3 só pode assumir o valor 0, então o contrator deve reduzir D_1 ao intervalo $[0, 0]$. Por outro lado, D_2 não deve sofrer redução, pois para qualquer valor de x_2 a restrição $0 = x_2 \cdot 0$ é satisfeita.

$$D_1 \leftarrow D_1 \cap D_2 \cdot D_3 = D_1 \cap \underbrace{D_2 \cdot [0, 0]}_{[0,0]} = [0, 0]$$

$$D_2 \leftarrow D_2 \cap D_1/D_3 = D_2 \cap \underbrace{[0, 0]/[0, 0]}_{\emptyset} = \emptyset$$

$$D_3 \leftarrow D_3 \cap D_1/D_2 = [0, 0] \cap \underbrace{[0, 0]/\emptyset}_{\emptyset} = \emptyset$$

△

Os Exemplos 5.1 e 5.2 mostram que a definição de divisão intervalar apresentada na Seção 2.3.2 não satisfaz a equivalência (5.8). Resolve-se este problema definindo a seguinte exceção:

$$0 \in (X_1 \cap X_2) \text{ ou } X_2 = [0, 0] \Rightarrow X_1/X_2 = (-\infty, +\infty) \tag{5.12}$$

Outro problema da divisão intervalar apresentada é que ela faz uso da multiplicação intervalar como parte do processamento, calculando o produto dos inversos dos extremos do

intervalo denominador, podendo ampliar erros de representação numérica e arredondamento. Uma versão mais eficiente seria:

$$X/Y = [\min\{\underline{X}/\underline{Y}, \underline{X}/\overline{Y}, \overline{X}/\underline{Y}, \overline{X}/\overline{Y}\}, \max\{\underline{X}/\underline{Y}, \underline{X}/\overline{Y}, \overline{X}/\underline{Y}, \overline{X}/\overline{Y}\}] \tag{5.13}$$

Aqui, podem aparecer as expressões $x/0$, $0/0$ e $\pm\infty/\pm\infty$. As definições apresentadas na seção anterior (equações (5.5), (5.6) e (5.7)) mantêm a equivalência entre essas duas definições. Mais especificamente, na equação $X/Y = X \cdot (1/Y)$ os conjuntos $\{\underline{X}/\underline{Y}, \underline{X}/\overline{Y}, \overline{X}/\underline{Y}, \overline{X}/\overline{Y}\}$ e $\{\underline{X} \cdot (1/\underline{Y}), \underline{X} \cdot (1/\overline{Y}), \overline{X} \cdot (1/\underline{Y}), \overline{X} \cdot (1/\overline{Y})\}$ devem ser idênticos.

Exemplo 5.3. Sejam dois intervalos $X_1 = [-2, 0)$ e $X_2 = (0, 4]$. Como $0 \notin X_2$, a operação X_1/X_2 é computada pela multiplicação de X_1 com o inverso de X_2:

$$X_1/X_2 = [-2, 0) \cdot [1/4, +\infty)$$

A multiplicação intervalar, por sua vez, computa o intervalo resultante com base nos máximos e mínimos do conjunto $\{\underline{X_i}\underline{X_j}, \underline{X_i}\overline{X_j}, \overline{X_i}\underline{X_j}, \overline{X_i X_j}\}$, que nesse caso é:

$$\{\underbrace{-2 \cdot (1/4)}_{-(1/2)}, \underbrace{-2 \cdot (+\infty)}_{-\infty}, \underbrace{0 \cdot (1/4)}_{0}, \underbrace{0 \cdot (+\infty)}_{0}\}$$

Assumindo que X_1/X_2 pode ser computado equivalentemente efetuando-se diretamente a operação de divisão nos extremos dos intervalos, obtém-se o conjunto $\{-2/0, -2/4, 0/0, 0/4\}$. Logo, a equivalência é mantida se, e somente se,

$$-2/0 = -\infty \qquad 0/0 = 0$$
$$-2/4 = -(1/2) \qquad 0/4 = 0$$

Alternativamente, como nesse caso define-se o inverso de 0 por $1/0 = +\infty$ e, em seguida, ocorre a multiplicação $0 \cdot (+\infty)$, resultando em 0, a expressão $0/0 = 0$ simplesmente evita um passo no processamento. △

Outro detalhe dessa definição é que $x/0$ pode resultar em $+\infty$ ou $-\infty$, dependendo do intervalo operado. Por exemplo, para $D_1 = [1, 2]$ e $D_2 = [-1, 0)$, x_1/x_2 é sempre negativo; logo, $1/0 = 2/0 = -\infty$. De modo geral, se o denominador da expressão X_1/X_2 é um intervalo da forma $X_2 = [\underline{X_2}, 0)$, então deve-se considerar $x/0 = -\infty$, para todo $x \neq 0$; do contrário, $x/0 = +\infty$.

Em suma, para a completude do contrator, define-se:

$$X/Y = \begin{cases} (-\infty, +\infty) & \text{se } 0 \in X \text{ e } 0 \in Y \\ (-\infty, +\infty) & \text{se } Y = [0, 0] \\ X/\langle[\underline{Y}, 0), (0, \overline{Y}]\rangle & \text{se } \underline{Y} < 0 < \overline{Y} \\ [\min\{\underline{X}/\underline{Y}, \underline{X}/\overline{Y}, \overline{X}/\underline{Y}, \overline{X}/\overline{Y}\}, \max\{\underline{X}/\underline{Y}, \underline{X}/\overline{Y}, \overline{X}/\underline{Y}, \overline{X}/\overline{Y}\}] & \text{caso contrário} \end{cases}$$

Potenciação, radiciação e logaritimização

As operações de potenciação e radiciação são implementadas conforme apresentado na Seção 2.3.2, adicionando-se a definição de expoente (ou índice) nulo:

$$X^0 = [1, 1] \tag{5.14}$$

$$\sqrt[0]{X} = (-\infty, +\infty) \tag{5.15}$$

Novamente, estabelece-se essa relação para manter a completude do contrator GAC.

Exemplo 5.4. A restrição $R_C \equiv x_1 = {x_2}^0$ é GAC em relação a x_1 se, e somente se, $D_1 = [1, 1]$. Por outro lado, se $1 \in D_1$, então R_C é GAC em relação a x_2, para qualquer domínio D_2. Logo,

$$D_1 \leftarrow D_1 \cap {D_2}^0 = D_1 \cap [1, 1] = [1, 1]$$
$$D_2 \leftarrow D_2 \cap \sqrt[0]{D_2} = D_2 \cap (-\infty, +\infty) = D_2$$

△

Do mesmo modo, como esta implementação considera apenas expoentes constantes (ou, equivalentemente, intervalos degenerados), não há redução de domínio possível ao projetar-se a restrição sobre o expoente. Logo, define-se:

$$\log_X Y = (-\infty, +\infty) \tag{5.16}$$

Outras operações

Como discutido no Capítulo 4, o método de otimização aqui proposto consiste em estender a valoração parcial $\langle x_1 = \min D_1\rangle$ a todas as variáveis da rede de forma consistente. Em teoria, se D_1 é um intervalo aberto à esquerda, então não possui mínimo. Devido à finitude de representação de números de máquina, no entanto, aproxima-se esse valor pelo menor número de máquina a maior que $\underline{D_1}$. O código abaixo implementa uma função `i_min` que retorna o mínimo de um intervalo X (uma função `i_max` é implementada de maneira análoga). A função `is_infn`(a) verifica se $a = -\infty$ e a função `r_cmp`$(a, \text{“} \bowtie \text{”}, b)$ verifica se $a \overset{\epsilon}{\bowtie} b$. A função

`nextafter`(a, b), fornecida pela biblioteca `math`, devolve o primeiro número de máquina depois de a, em direção a b. As constantes `EPSILON` e `EPSILON_REL` são, respectivamente, as tolerâncias ϵ e ϵ_R utilizadas na comparação de ponto flutuantes.

```
1  #include <math.h>
2
3  real i_min(intervalo X) {
4    if (X->T_e || is_infn(X->ext_e))
5      return X->ext_e;
6
7    // como X->ext_e + EPSILON "==" X->ext_e
8    // encontra o valor representado próximo à este
9    real valor = nextafter(X->ext_e + EPSILON, INFINITY);
10
11   // se X->ext_e tem alta magnitude, a operação
12   // acima ainda pode resultar em valor "==" X->ext_e
13   // Nesse caso, calcula-se utilizando epsilon relativo
14   if (r_cmp(valor, "==", X->ext_e))
15     valor = nextafter(X->ext_e + X->ext_e * EPSILON_REL), INFINITY);
16
17   // Se o valor obtido não está no intervalo,
18   // então o mínimo desse intervalo será o seu máximo
19   if (r_cmp(valor, ">=", X->ext_d))
20     valor = X->ext_d;
21
22   return value;
23 }
```

Outra função utilizada pelo otimizador é a bissecção de um intervalo na fase de ramificação do algoritmo (linha 12 do Algoritmo 19). Esta função deve particionar o intervalo X no seu ponto médio $(\underline{X} + \overline{X})/2$, o que não é bem definido caso um dos extremos seja $-\infty$ ou $+\infty$. Nesse caso, define-se uma constante `MAX_REAL` e particiona-se o intervalo, por exemplo, $[a, +\infty)$, nos subintervalos $[a,$ `MAX_REAL`$)$ e [`MAX_REAL`$, +\infty)$.

As demais funções da biblioteca intervalar, como a união e a interseção de intervalos, são implementadas de maneira usual, apenas verificando os extremos de cada intervalo. No entanto, deve-se atentar para o fato de que a comparação de números em ponto flutuante utilizando tolerância pode tornar certas operações inconsistentes. Por exemplo, computa-se

$$X \cap Y = [\max\{\underline{X}, \underline{Y}\}, \min\{\overline{X}, \overline{Y}\}] \tag{5.17}$$

Se os operandos forem abertos ou semi-abertos, esta propriedade deve ser mantida no intervalo resultante. No entanto, sendo ϵ a tolerância de igualdade na comparação de números

em ponto flutuante, a forma como os extremos são comparados pode resultar em um intervalo inconsistente:

$$[x, x] \cap (x - \epsilon, x + \epsilon) = \{\max\{\underbrace{x, x - \epsilon}_{x \overset{\epsilon}{=} x - \epsilon}\}, \min\{\underbrace{x, x + \epsilon}_{x \overset{\epsilon}{=} x + \epsilon}\}\} = (x - \epsilon, x] = \emptyset$$

Por fim, é importante notar que operações intervalares podem resultar em dois intervalos disjuntos (união, divisão e radiciação). Nesse caso, deve-se manter estes intervalos ordenados, facilitando a implementação de operações multi-intervalares.

5.2.3 Multi-intervalos

Um multi-intervalo é uma lista ordenada e finita de intervalos. Operações multi-intervalares são computadas operando-se todo par de intervalos que compõem seus operandos. O código abaixo mostra a implementação da função `M_op` que opera dois multi-intervalos X e Y de acordo com a operação `i_op` passada por parâmetro.

```
1  Mintervalo M_op(Mintervalo X, lista i_op(intervalo A, intervalo B),
2                  Mintervalo Y){
3
4   Mintervalo Z = cria_lista();
5   for (no p = primeiro_no(X); p; p = proximo_no(p))
6     for (no q = primeiro_no(Y); q; q = proximo_no(q)) {
7       lista temp = i_op(objeto(p), objeto(q)); // objeto do nó é um intervalo
8       // Z <- Z U temp, de forma ordenada e removendo redundâncias
9       merge(Z, temp);
10    }
11  if (primeiro_no(Z) == NULL)
12    Z = cria_Mintervalo_vazio();
13  return Z;
14 }
```

É importante manter a representação multi-intervalar ordenada e sem redundância, para evitar o crescimento exponencial de custo de memória, embora, como mostrado na Seção 3.3.1, há instâncias nas quais esse custo é inevitável.

Nesta implementação, define-se também as operações de interseção, união e bissecção de multi-intervalos na forma usual. Em particular, bissectar um multi-intervalo é trivial se o número de intervalos que o compõe é par; do contrário, particiona-se o intervalo central como descrito na Seção 5.2.2.

Um procedimento não trivial e de fundamental importância no `OGRe` é calcular a distância entre dois multi-intervalos. Como discutido brevemente na Seção 4.4.1, uma heurística

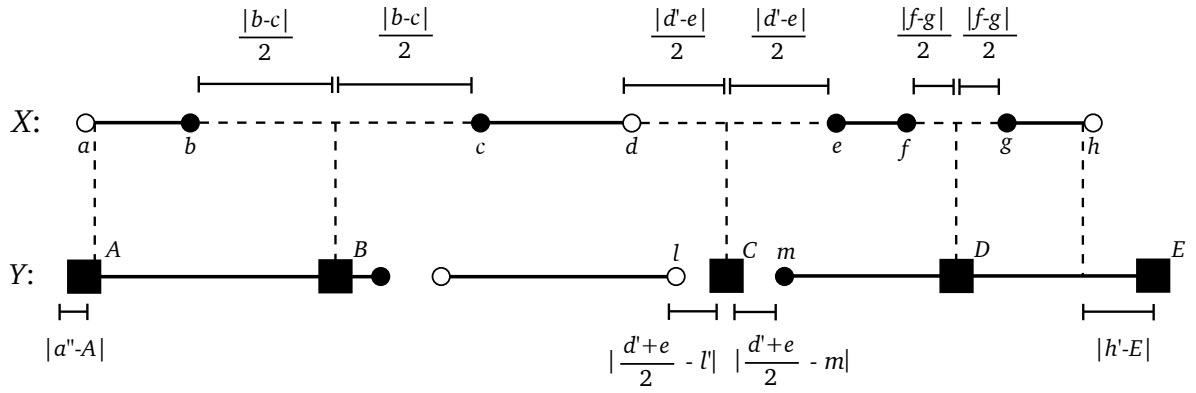


Figura 5.1: Esquema de distância entre dois multi-intervalos X e Y, tais que $X \subset Y$. Os quadrados $A, \ldots, E$ são os valores mais distantes de X, calculados como o ponto médio da distância entre os intervalos que o compõe. Em particular, o valor C não pertence a Y, logo, deve-se subtrair a distância desse valor para Y. A distância total é calculada pela distância máxima entre esses pontos.

para ramificar intervalos no Algoritmo 19 é escolher a partição que maximiza as chances da rede tornar-se RAC. Como uma restrição $R_C \equiv x_1 = x_2 \circ x_3$ satisfaz essa consistência se, e somente se, $D_1 \supseteq D_2 \circ D_3$, deseja-se medir qual a distância máxima entre valores de D_1 e $D_2 \circ D_3$, quando $D_1 \subseteq D_2 \circ D_3$. Se D_1 e $D_2 \circ D_3$ são ambos intervalos, essa medição é trivial:

$$dist = \max\{|\underline{D_1} - \underline{D_2 \circ D_3}|, |\overline{D_1} - \overline{D_2 \circ D_3}|\} \tag{5.18}$$

Se D_1 e $D_2 \circ D_3$ são multi-intervalos, a medição deve considerar a distância máxima entre todos os intervalos que compõem X, conforme ilustrado na Figura 5.1. Nesse exemplo, $X = \langle (a, b], [c, d), [e, f], [g, h) \rangle$, $Y = \langle [i, j], (k, l), [m, n] \rangle$ e $dist = \max\{d_A, d_B, d_C, d_D, d_E\}$, onde

$$d_A = |a'' - A| \qquad d_D = \frac{|f - g|}{2}$$

$$d_B = \frac{|b - c|}{2} \qquad d_E = |h' - E|$$

$$d_C = \max\{\frac{|d' - e|}{2} - |\frac{d' + e}{2} - l'|, \frac{|d' - e|}{2} - |\frac{d' + e}{2} - m'|\}$$

onde a'' é o menor número de máquina maior que a e d', l', m' e h' são, respectivamente, os maiores números de máquina menores que a, d, l, m e h.

5.3 O otimizador `OGRe`

O `OGRe` implementa o método `RAC_direcionada_aprox` (Algoritmo 19), juntamente com a biblioteca multi-intervalar e uma estrutura de dados para representar decom-

posições Epífitas. Para simplificar a implementação, efetua-se um pré-processamento nas instâncias substituindo-se toda restrição $R_C \in \Omega$ pela restrição equivalente $R_C' \equiv x_1 = x_2 \circ x_3$ tal que $t(R_C') = x_1$. Por exemplo, em uma decomposição Epífita $(\mathcal{A}, \Omega, t)$ tal que $\Omega = \{R_{C_1} \equiv x_1 = x_2 + x_3, R_{C_2} \equiv x_3 = x_4 \cdot x_5\}$ e $t(R_{C_1}) = x_2$ e $t(R_{C_2}) = x_5$, considera-se as restrições equivalentes $R_{C_1}' \equiv x_2 = x_1 - x_3$ e $R_{C_2}' \equiv x_5 = x_3/x_4$.

```
#ifndef __EPIFITA__
#define __EPIFITA__

typedef struct t_variavel *variavel;
struct t_variavel {
  Mintervalo multi_intervalo;
  lista restricoes;       // restrições das árvores que contém essa variável
  lista restricoes_omega; // restrições de Omega que contém essa variável
};

typedef struct t_restricao *restricao;
struct t_restricao {
  variavel X1, X2, X3;
  lista (*f_op)(intervalo A, intervalo B);    // operação
  lista (*f_iop)(intervalo A, intervalo B);   // (primeiro inverso)
  lista (*f_cop)(intervalo A, intervalo B);   // (segundo inverso)
  restricao pai;          // ordenação topológica da árvore
  lista filhos;
};

typedef struct t_arvore *arvore;
struct t_arvore {
  restricao raiz;
  lista folhas;
  lista variaveis;
  restricao t; // restricao R em Omega tal que t(R) pertence a esta árvore
};

typedef struct t_epifita *epifita;
struct t_epifita {
  lista arvores;
  lista omega;
};

#endif
```

O Algoritmo 19 possui três módulos principais: `GAC_direcionada`, `tenta_valoracao` e `seleciona_e_bissecta`. Como mencionado na Seção 4.4.1,

o contrator GAC direcionado pode ser substituído por um contrator GAC total a fim de reduzir ainda mais o espaço de busca. No otimizador aqui proposto, implementa-se ambas as estratégias. Mais especificamente, o contrator GAC direcionado é implementado como descrito pelo Algoritmo 17, enquanto o GAC total segue a estratégia do método `AC-3` (Algoritmo 4), juntamente com o contrator GAC ternário (Algoritmo 11).

Como discutido na Seção 3.3.1, GAC não pode ser alcançada computacionalmente devido à finitude de representação numérica. Embora a consistência de envoltória seja usualmente utilizada para aproximar GAC, os resultados apresentados no Capítulo 4 têm como pré-condição uma rede GAC. Da mesma forma que aproxima-se RAC por meio de uma tolerância ε, optou-se por manter o contrator GAC usual, mesmo que os extremos dos intervalos sejam, na prática, arredondados para os números de máquina mais próximos.

As subseções a seguir discutem brevemente a implementação dos módulos `tenta_valoracao` e `seleciona_e_bissecta`.

5.3.1 Valoração

Na linha 8 do Algoritmo 19, tenta-se valorar a rede a partir da atribuição inicial $\langle x_1 = \min D_1 \rangle$. Esse procedimento percorre uma ordenação d de restrições com grau de interseção no máximo 2, definido pela ordenação topológica da decomposição Epífita. Variáveis de restrições com grau de interseção no máximo 1 são valoradas de forma exata utilizando o contrator GAC, conforme descrito pelo Algoritmo 21. Esta função assume que, se o grau de interseção é 1, então a variável da restrição que pertence a restrições antecessoras na ordenação já foi valorada; logo, seu domínio é unitário. Embora GAC garanta a existência de valoração consistente, tal valoração pode não ser representada computacionalmente. Nesse caso, o algoritmo devolve ERRO (linha 14).

Por outro lado, restrições com grau de interseção 2 devem ser ε_{AR}-factíveis. O Algoritmo 22 recebe como entrada uma restrição R_C com duas variáveis já valoradas e infere o valor da variável restante. A valoração ocorre relaxando-se o domínio da variável, mas verifica-se a diferença entre este valor e seu domínio original (linhas 3 e 8). Assim como números em ponto flutuante são comparados considerando-se tanto o erro absoluto quanto o erro relativo, esta diferença considera as tolerâncias ε_A e ε_R.

Por fim, o Algoritmo 23 percorre a decomposição Epífita estendendo a valoração parcial $\langle x_1 = \min D_1 \rangle$. Este algoritmo devolve uma valoração da rede que satisfaz as restrições em Ω com tolerância ε_{AR}, ou uma lista de restrições I_Ω não satisfeitas. Se erros numéricos ocorreram durante o processo de inferência, o algoritmo devolve ERRO e nova ramificação ocorrerá no método principal. Como a busca pode continuar em subdomínios deste ramo, caso não seja possível obter uma valoração total da rede, os domínios em $\mathcal{D}$ são copiados para $\mathcal{D}'$ antes de serem processados.

Algoritmo 21 Valora variáveis GAC (`valora_GAC`(R_C, D_1, D_2, D_3))

Entrada: Restrição $R_C \equiv x_1 = x_2 \circ x_3$ sobre as variáveis x_1, x_2 e x_3.

Saída: D_1, D_2 e D_3 atualizados, tais que $|D_1| = |D_2| = |D_3| = 1$ e R_C é 0-factível.

1: `contratorGAC_ternario`(R_C, D_1, D_2, D_3)
2: **se** $|D_1| > 1$ **então**
3: $D_1 \leftarrow$ [`ponto_medio`(D_1), `ponto_medio`(D_1)]
4: `contratorGAC_ternario`(R_C, D_1, D_2, D_3)
5: **fim se**
6: **se** $|D_2| > 1$ **então**
7: $D_2 \leftarrow$ [`ponto_medio`(D_2), `ponto_medio`(D_2)]
8: `contratorGAC_ternario`(R_C, D_1, D_2, D_3)
9: **fim se**
10: **se** $|D_3| > 1$ **então**
11: $D_3 \leftarrow$ [`ponto_medio`(D_3), `ponto_medio`(D_3)]
12: **fim se**
13: **se** $D_1 = \emptyset$ ou $D_2 = \emptyset$ ou $D_3 = \emptyset$ **então**
14: **devolve** ERRO
15: **fim se**

Algoritmo 22 Infere variável restante (`infere_restante`$(R_C, D_1, D_2, D_3, \varepsilon_{AR}, I_\Omega)$)

Entrada: Restrição $R_C \equiv x_1 = x_2 \circ x_3$ sobre as variáveis x_1, x_2 e x_3 com domínios D_1, D_2 e D_3, tais que $|D_2| = |D_3| = 1$, conjunto de tolerâncias $\varepsilon_{AR} = \{\varepsilon_A, \varepsilon_R\}$ e pilha de restrições I_Ω.

Saída: D_1 atualizado, tal que $|D_1| = 1$ e (i) R_C é ε-factível ou (ii) R_C é inserido na pilha.

1: $T \leftarrow D_2 \circ D_3$
2: **se** $|T| = 1$ **então**
3: $(a, d) \leftarrow$ `ponto_mais_proximo`(D_1, T) // devolve o valor $a \in D_1$ mais próximo do
4: $D_1 \leftarrow [a, a]$ // intervalo T e a sua distância d
5: **senão**
6: // divisão por zero, logaritmo, etc, resultam em $T = (-\infty, +\infty)$
7: // nesse caso verifica-se a distância equivalente entre D_2 e $D_1 \bullet D_3$
8: $(a, d) \leftarrow$ `ponto_mais_proximo`$(D_2, D_1 \bullet D_3)$
9: // em qualquer das exceções, R_C é GAC em relação a x_1, para todo D_1
10: // logo, x_1 pode assumir qualquer valor de seu domínio
11: // (preferencialmente o valor mais distante dos seus extremos)
12: $D_1 \leftarrow$ [`ponto_medio`(D_1), `ponto_medio`(D_1)]
13: **fim se**
14: **se** $d > \varepsilon_A$ e $d > \varepsilon_R \cdot a$ **então**
15: `empilha`(R, I_Ω)
16: **fim se**

Algoritmo 23 Tenta valoração (`tenta_valoracao`$(\mathcal{R}, d, \varepsilon_{AR})$)

Entrada: Rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$ que possui uma decomposição Epífita $(\mathcal{A}, \Omega, t)$, ordenação de restrições d com grau de interseção 2 e conjunto de tolerâncias ε_{AR}.

Saída: Valoração I (não necessariamente consistente) de $\mathcal{R}$, valor de custo ótimo x_1 e pilha I_Ω de restrições em Ω que não são satisfeitas com tolerância ε_{AR}.

1: $I_\Omega \leftarrow \emptyset$
2: $\mathcal{D}' \leftarrow \mathcal{D}$
3: $D_1{}' \leftarrow [\min D_1{}', \min D_1{}']$ // $\min D_1{}'$ é implementada pela função `i_min`$(D_1{}')$
4: **para** todo $(R_{C_j} \equiv x_{j_1} = x_{j_2} \circ x_{j_3}) \in d$ **faça**
5: **se** $R_{C_j} \in \Omega$ **então**
6: $(D_{j_1}{}', I_\Omega) \leftarrow$ `infere_restante`$(R_{C_j}, D_{j_1}{}', D_{j_2}{}', D_{j_3}{}', \varepsilon_{AR}, I_\Omega)$
7: **senão**
8: $(D_{j_1}{}', D_{j_2}{}', D_{j_3}{}') \leftarrow$ `valora_GAC`$(R_{C_j}, D_{j_1}{}', D_{j_2}{}', D_{j_3}{}')$
9: **se** $D_{j_1}{}' = \emptyset$ ou $D_{j_2}{}' = \emptyset$ ou $D_{j_3}{}' = \emptyset$ **então**
10: **devolve** ERRO
11: **fim se**
12: **fim se**
13: **fim para**
14: $I \leftarrow \emptyset$
15: **para** todo $x_i \in \mathcal{X}$ **faça**
16: $I \leftarrow I \cup \{\langle x_i = \min D_i{}' \rangle\}$ // todo domínio $D_i{}'$ é degenerado
17: **fim para**
18: **devolve** (x_1, I, I_Ω)

5.3.2 Heurísticas

O procedimento `seleciona_e_bissecta` deve escolher alguma variável da rede para ser ramificada, isto é, para que seu domínio seja particionado. No `OGRe`, quatro heurísticas principais são aplicadas:

1. escolher variáveis de restrições $R_C \in \Omega$ na decomposição Epífita que não são satisfeitas com tolerância ε_{AR}, excluindo-se a variável $t(R_C)$, pois a redução de domínio das variáveis em $C \setminus \{t(R_C)\}$ diminuem o ε_{AR} necessário para que R_C torne-se RAC;

2. dar preferência pela restrição R_C cuja variável $t(R_C)$ pertence à árvore de maior índice na decomposição Epífita. De forma semelhante ao algoritmo para alcançar GAC direcionada, que processa restrições no sentido contrário à ordenação d, ramificar as variáveis no sentido inverso evita que restrições satisfeitas com tolerância ε_{AR} sejam reprocessadas;

3. ramificar primeiro restrições R_C tais que o domínio de $t(R_C)$ não é unitário. Em geral, a operação $D_2 \circ D_3$ resulta em um intervalo unitário apenas quando D_2 e D_3 são unitários,

acarretando uma sequência de ramificações que termina apenas quando estes intervalos atingirem a precisão mínima Δ;

4. uma vez escolhida a restrição cujas variáveis são candidatas à ramificação, verifica-se qual partição maximiza as chances da restrição tornar-se RAC.

O Algoritmo 24 descreve o procedimento `seleciona_e_bissecta`, que recebe como parâmetro uma pilha I_Ω de restrições em Ω que não são satisfeitas com tolerância ε_{AR}. Apenas restrições desse conjunto são consideradas na ramificação, aplicando a heurística 1. Assume-se que essa pilha está ordenada da última para a primeira restrição e portanto, ao seguir esta ordem, aplica-se também a heurística 2. Uma restrição $R_C \in I_\Omega$, tal que o domínio de $t(R_C)$ é unitário, será escolhida apenas se não existir restrição satisfazendo o contrário (linhas 1 a 5), conforme a heurística 3. Escolhida uma restrição $R_C \equiv x_1 = x_2 \circ x_3$, bissecta-se os domínios das variáveis x_2 e x_3 e, para cada partição, verifica-se qual a distância entre D_1 e $D_2 \circ D_3$ (este valor é calculado como exemplificado na Figura 5.1). Essa distância é usada como heurística de quão longe está a restrição de ser RAC; simplificadamente, este parâmetro mede a maior diferença entre valores em $D_2 \circ D_3$ com valores em D_1 (se $D_1 \subseteq D_2 \circ D_3$, então a distância é 0). Esse parâmetro é então utilizado para decidir qual das variáveis é ramificada, x_2 ou x_3, bem como a ordem em que as partições serão processadas (linhas 13 e 17). Lembra-se que, no Algoritmo 19, o par $(D^\bullet, D^*)$ é empilhado nesta ordem e, portanto, D^* será processado antes que $D^\bullet$. Como nos métodos de ramificação e poda intervalares, utiliza-se uma precisão mínima Δ para o particionamento de intervalos. Dessa forma, quando ambos os domínios D_2 e D_3 considerados para ramificação atingirem essa precisão, o método encerra o processamento deste ramo na árvore de busca, retornando domínios vazios (linha 15).

5.4 Resultados experimentais

Com o objetivo de avaliar experimentalmente os resultados teóricos apresentados nesta tese e a decomposição Epífita de instâncias NCOP, aplicou-se o otimizador `OGRe` em um conjunto experimental de 130 instâncias da biblioteca GLOBAL *Library* (GAMS, 2002), que consiste de problemas reais de otimização com relevância industrial e acadêmica. Essa biblioteca está incorporada ao *benchmark* COCONUT (Shcherbina et al., 2003a,b), ambiente utilizado para a comparação de otimizadores globais em Neumaier et al. (2005).

As instâncias escolhidas para os experimentos limitaram-se àquelas com as seguintes características:

- problemas de otimização sem a ocorrência de variáveis discretas;
- instâncias que utilizam-se apenas dos operadores $+$, $-$, $\cdot$, $/$, $\hat{}$ e $\sqrt{}$, sendo estes últimos limitados, respectivamente, a expoentes e índices inteiros.

Algoritmo 24 Seleciona variável e bissecta domínio (`seleciona_e_bissecta`($\mathcal{R}$, I_Ω, Δ))

Entrada: Rede $\mathcal{R} = (\mathcal{X}, \mathcal{D}, C)$, pilha de restrições I_Ω e precisão $\Delta > 0$.
Saída: Subdomínios complementares $\mathcal{D}^\bullet$ e $\mathcal{D}^*$.

se $\forall(R_C \equiv x_1 = x_2 \circ x_3) \in I_\Omega : |D_1| = 1$ **então**
  $R_C \leftarrow$ `desempilha`(I_Ω) // heurísticas 1, 2 e 3
senão
  seja $R_C \equiv x_1 = x_2 \circ x_3$ a restrição de menor índice em I_Ω tal que $|D_1| > 1$
fim se
$d_2' \leftarrow d_2'' \leftarrow d_3' \leftarrow d_3'' \leftarrow +\infty$
se $|D_2| > \Delta$ **então**
  $(D_2', D_2'', d_2', d_2'') \leftarrow$ `particiona`(D_2, R_C) // devolve a dist. entre D_1 e $D_2 \circ D_3$ nas partições
fim se
se $|D_3| > \Delta$ **então**
  $(D_3', D_3'', d_3', d_3'') \leftarrow$ `particiona`(D_3, R_C)
fim se
seja $d_i^* = \min\{d_2', d_2'', d_3', d_3''\}$ e D_i^* a sua respectiva partição // heurística 4
se $d_i^* = +\infty$ **então**
  devolve $(\emptyset, \emptyset)$
senão
  seja $D_i^\bullet$ o complemento de D_i^* // por exemplo, D_2' é o complemento de D_2''
  $\mathcal{D}^* \leftarrow \mathcal{D} \setminus \{D_i\} \cup \{D_i^*\}$
  $\mathcal{D}^\bullet \leftarrow \mathcal{D} \setminus \{D_i\} \cup \{D_i^\bullet\}$
  devolve $(\mathcal{D}^\bullet, \mathcal{D}^*)$
fim se

Primeiramente, para cada instância desse conjunto obteve-se uma decomposição Epífita, através da codificação ternária do problema e do algoritmo para encontrar ordenação com grau de interseção no máximo 2 (Algoritmo 18). Verificou-se que todas as instâncias possuem pelo menos uma decomposição 1-Epífita, evidenciando que esta classe de problemas é relevante na prática. A Figura 5.2 apresenta o número de variáveis, restrições ternárias e árvores que compõem a decomposição Epífita de cada instância convertida, originalmente com m restrições e n variáveis. Nota-se que estes três parâmetros são proporcionais e que crescem linearmente no tamanho da fórmula, embora a soma $m + n$ não seja um parâmetro que reflete o tamanho real de uma instância, pois não mede o número de operações em cada restrição. Como heurística para obter a decomposição Epífita efetuou-se uma busca gulosa maximizando o tamanho da primeira árvore da decomposição. A Figura 5.3 ilustra uma decomposição Epífita da instância *ex_9_2_* do *benchmark*. Nota-se que a primeira árvore é composta de várias restrições, enquanto as demais possuem apenas uma restrição ou uma variável[3].

Para cada instância gerada, executou-se o otimizador `OGRe` com um tempo limite de 7200 segundos. Como não é trivial estabelecer os parâmetros ε_{AR} e Δ no caso geral, considerou-se um conjunto pré-definido de parâmetros, conforme mostra a Tabela 5.1.

O `OGRe` encontrou soluções ε_{AR}-factíveis para 79% das instâncias utilizando o contrator GAC direcionado e 83% utilizando o contrator GAC total. Destas instâncias, 74 foram resolvidas

[3]Curiosamente, no mundo real, árvores Epífitas também tendem a possuir essa estrutura, pois apenas a primeira planta tem contato direto com o solo.

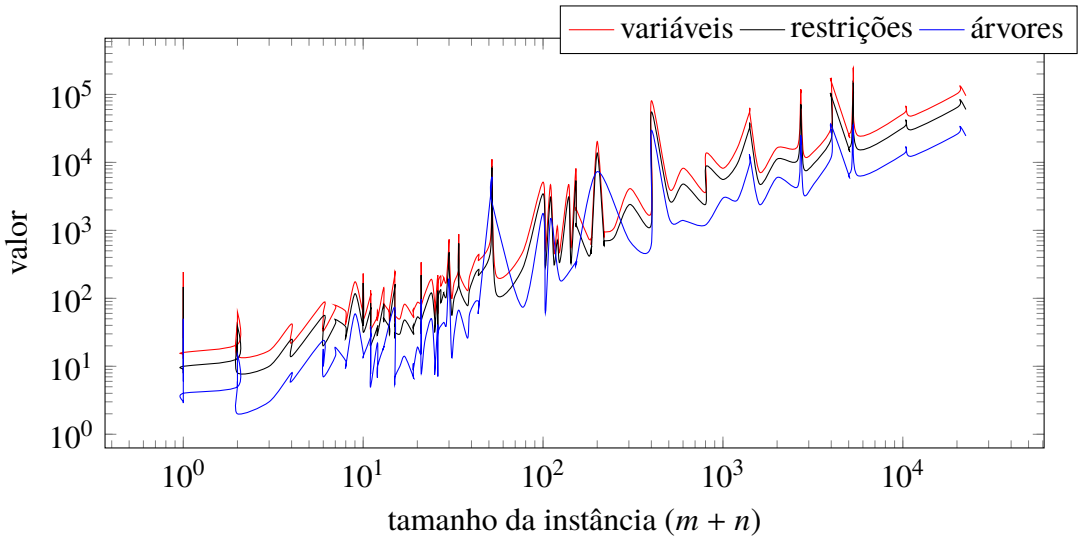


Figura 5.2: Número de variáveis, restrições e árvores da decomposição Epífita de instâncias com m restrições e n variáveis originais.

conjunto	ε_{AR}	Δ
A	10^{-4}	10^{-5}
B	10^{-4}	10^{-4}
C	10^{-4}	10^{-3}
D	10^{-3}	10^{-4}
E	10^{-3}	10^{-3}
F	10^{-3}	10^{-2}
G	10^{-3}	10^{-1}
H	10^{-2}	10^{-4}
I	10^{-2}	10^{-3}

conjunto	ε_{AR}	Δ
J	10^{-2}	10^{-2}
K	10^{-2}	10^{-1}
L	10^{-2}	1
M	10^{-1}	10^{-4}
N	10^{-1}	10^{-3}
O	10^{-1}	10^{-2}
P	10^{-1}	10^{-1}
Q	10^{1}	10^{-3}
R	1	10^{-3}

conjunto	ε_{AR}	Δ
S	1	10^{-2}
T	1	10^{-1}
U	1	0,5
V	5	10^{-2}
W	5	10^{-1}
X	5	1
Y	10	10^{-1}
Z	10	1

Tabela 5.1: Conjuntos de parâmetros de execução do otimizador `OGRe`.

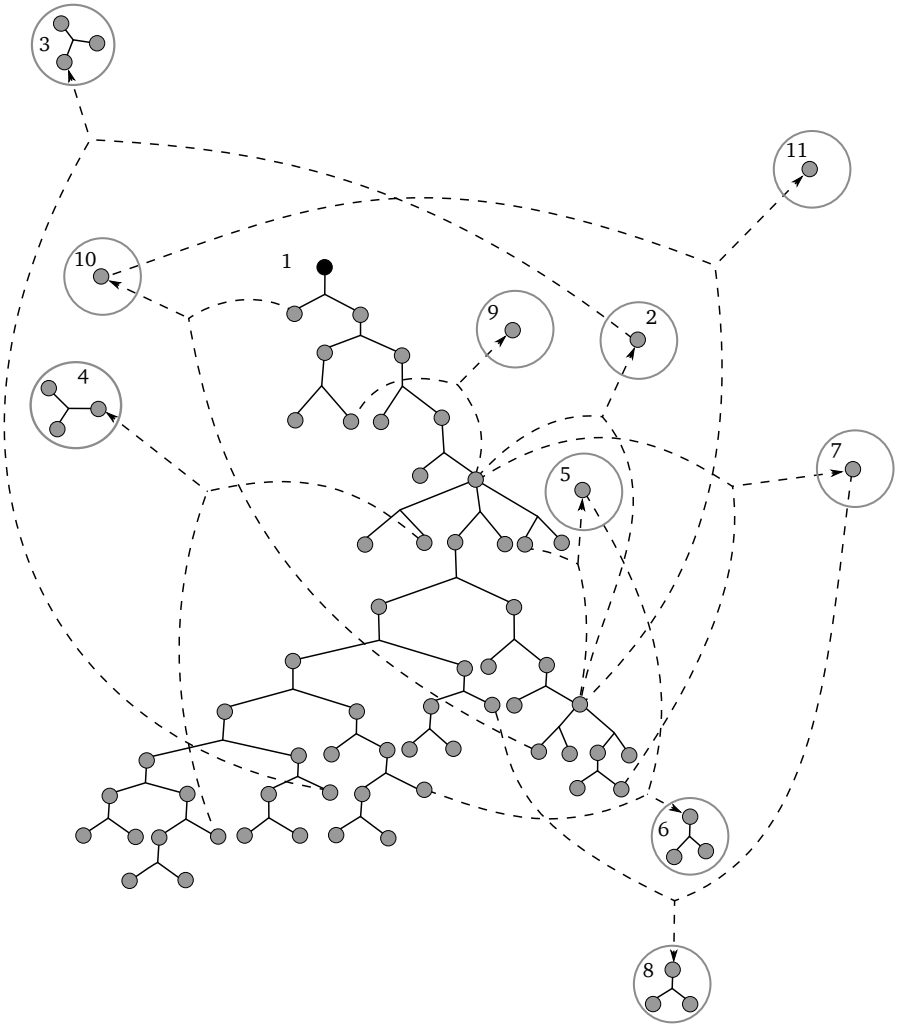


Figura 5.3: Uma decomposição Epífita da instância *ex_9_2_7* do *benchmark* COCONUT.

com um valor mínimo à função objetivo próximo ao ótimo global da instância com erro absoluto no máximo 10 (respectivamente, 82 instâncias considerando-se o contrator GAC total). Por exemplo, as instâncias *sample* e *rbrock*, já analisadas nesta tese, foram resolvidas com erro absoluto, respectivamente, 0,0223 e 0. Já a instância *ex_9_2_7*, cuja codificação está representada na Figura 5.3, foi resolvida com erro absoluto 0,0005.

Dado o tempo limite de 7200 segundos e o conjunto de parâmetros ε_{AR} e Δ da Tabela 5.1, algumas instâncias não puderam ser resolvidas com um erro absoluto admissível. Por exemplo, a instância *ex5_4_2* possui mínimo global 7512,23, enquanto a melhor execução do `OGRe` encontrou o valor 7158,75, com um erro absoluto de 353,48. No entanto, neste caso o erro relativo é de apenas 4,7%. A Figura 5.4 ilustra a distribuição das instâncias em relação aos erros absoluto e relativo gerados pelo otimizador. Instâncias com erros similares foram aglomeradas a fim de uma melhor visualização; a circunferência de cada ponto aumenta conforme

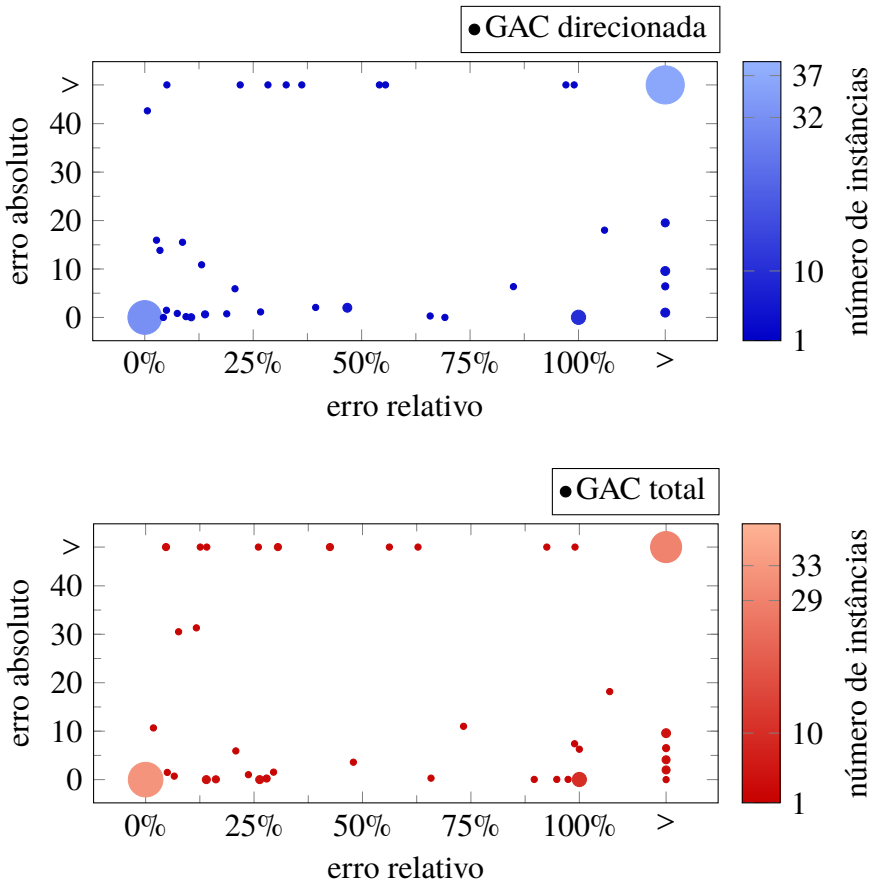


Figura 5.4: Relação entre erro absoluto e erro relativo de cada instância obtidos pelo otimizador `OGRe` utilizando um contrator GAC direcionado (gráfico superior, em azul) e um contrator GAC total (gráfico inferior, em vermelho). A circunferência de cada ponto é proporcional ao número de instâncias agrupadas com erros semelhantes. Os números de instâncias representadas pelos três círculos maiores estão indicados na barra lateral de cada gráfico.

o número de instâncias naquela coordenada. Dessa forma, instâncias no canto inferior esquerdo são aquelas resolvidas com alta precisão, enquanto as instâncias localizadas no canto superior direito apresentam um grande erro absoluto e relativo, ou não foram resolvidas pelo método. Em particular, 32 instâncias foram resolvidas com erros absoluto e relativo próximos a 0 utilizando o contrator GAC direcionado, enquanto 33 foram resolvidas com a mesma aproximação utilizando o contrator GAC total. Ambas as abordagens apresentam uma quantidade de 10 instâncias resolvidas com erro absoluto aproximadamente 0 e erro relativo 100%; para essas instâncias, o `OGRe` encontrou como solução o valor 0, enquanto o mínimo global é um valor diferente de 0, embora bem próximo a este. A Figura 5.5 evidencia a diferença utilizando-se os contratores GAC direcionado e total. Nota-se que o comportamento geral do otimizador é similar em ambas as abordagens, mas o contrator total é capaz de encontrar soluções melhores no mesmo limite de tempo, pois poda o espaço de busca de maneira mais eficiente.

A Figura 5.6 mostra a distribuição de erros gerados pelo otimizador em relação ao tamanho das instâncias, utilizando ambos os contratores. Embora o *benchmark* utilizado não seja uniforme, isto é, possui mais instâncias pequenas do que grandes, nota-se que não há uma

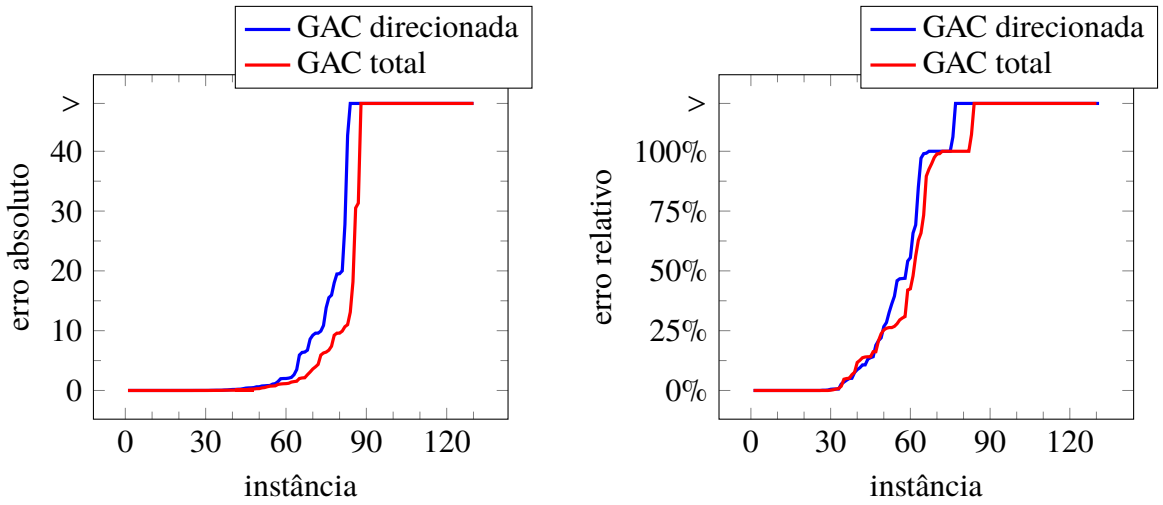


Figura 5.5: Comparação entre os resultados obtidos utilizando-se o contrator GAC direcionado (em azul) e o total (em vermelho). O gráfico esquerdo compara os erros absolutos obtidos por cada técnica, enquanto o direito compara os erros relativos. As instâncias estão ordenadas de acordo com os respectivos erros.

relação evidente entre tamanho e erro gerado. É provável que o erro esteja mais relacionado com a estrutura do problema, principalmente com as operações que definem cada restrição da rede e a sua não linearidade, pois, nestes casos, pequenas variações nos intervalos de cada variável e no parâmetro ε_{AR} podem resultar em uma grande variação no valor da função objetivo.

Embora o objetivo destes experimentos não seja comparar o tempo de execução do otimizador em relação a outros otimizadores do estado da arte, a Figura 5.7 apresenta uma a relação entre tempo de processamento e tamanho da instância na execução do `OGRe`, considerando ambos os contratores. Nesse caso, a abordagem direcionada apresenta um tempo de processamento ligeiramente melhor que a versão total, embora, assim como na relação entre erro e tamanho da instância, não fica evidente uma relação entre o tempo de processamento e o tamanho de cada instância (número de restrições ternárias), pois o espaço de busca é definido, principalmente, pelos comprimentos dos intervalos iniciais de cada variável.

Por fim, as Tabelas 5.2, 5.3 e 5.4 apresentam os resultados de cada execução do otimizador utilizando o contrator GAC total. As duas primeiras colunas apresentam o nome da instância no *benchmark* COCONUT e o respectivo número de restrições ternárias após a sua decomposição Epífita. Em seguida, apresentam-se os parâmetros ε_{AR} e Δ da melhor execução do `OGRe` com essa instância e a solução encontrada pelo otimizador, bem como os erros absoluto (EA) e relativo (ER) desta solução em relação ao ótimo global de cada instância (se este for 0, então o erro relativo não é definido). O tempo de execução, em segundos, também é apresentado, onde TLE (Tempo Limite Excedido), indica que a execução foi interrompida porque o tempo de execução excedeu o limite de 7200 segundos. As instâncias em negrito são aquelas resolvidas com erro absoluto menor do que 10 ou erro relativo menor do que 15%. Para uma melhor visualização, os dados foram arredondados.

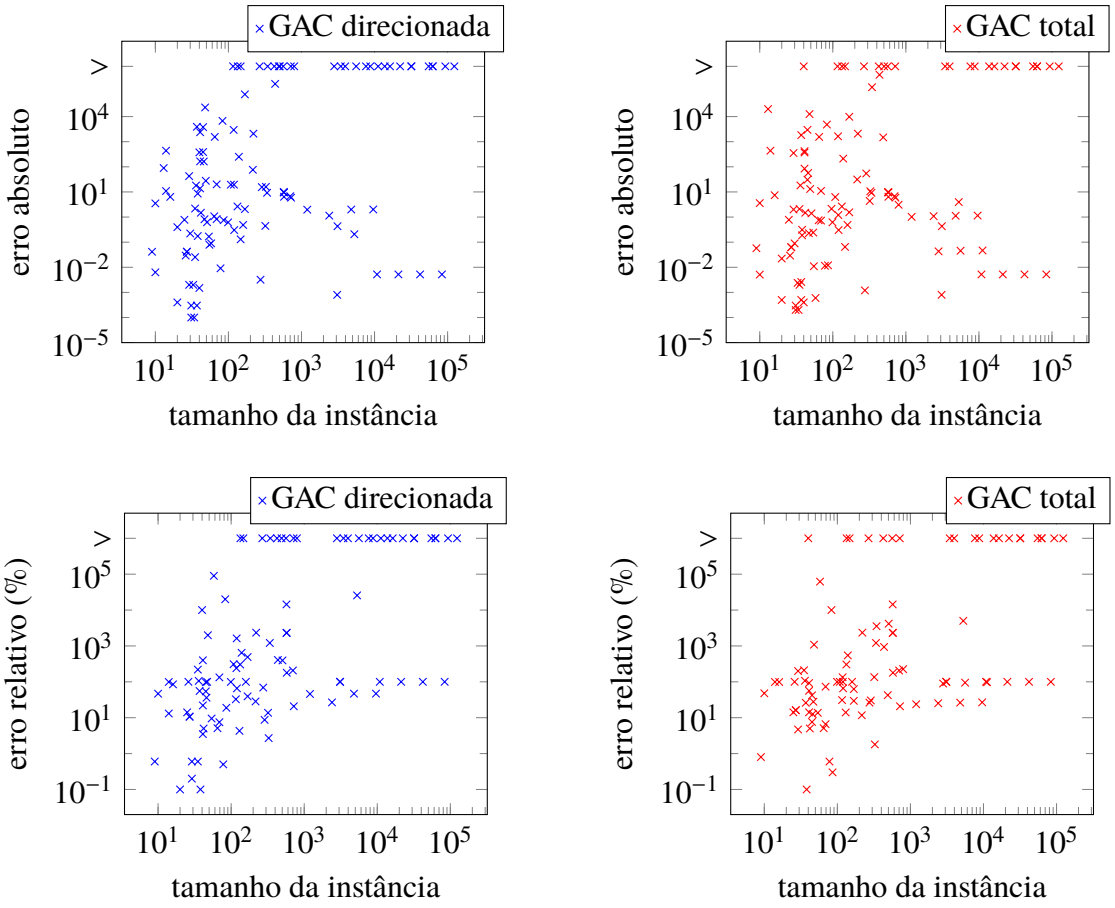


Figura 5.6: Relação entre o tamanho da instância (número de restrições ternárias) e os erros absoluto e relativo obtidos utilizando-se GAC direcionada (em azul) e GAC total (em vermelho). Os gráficos superiores mostram os erros absolutos em cada instância, enquanto os inferiores mostram os erros relativos.

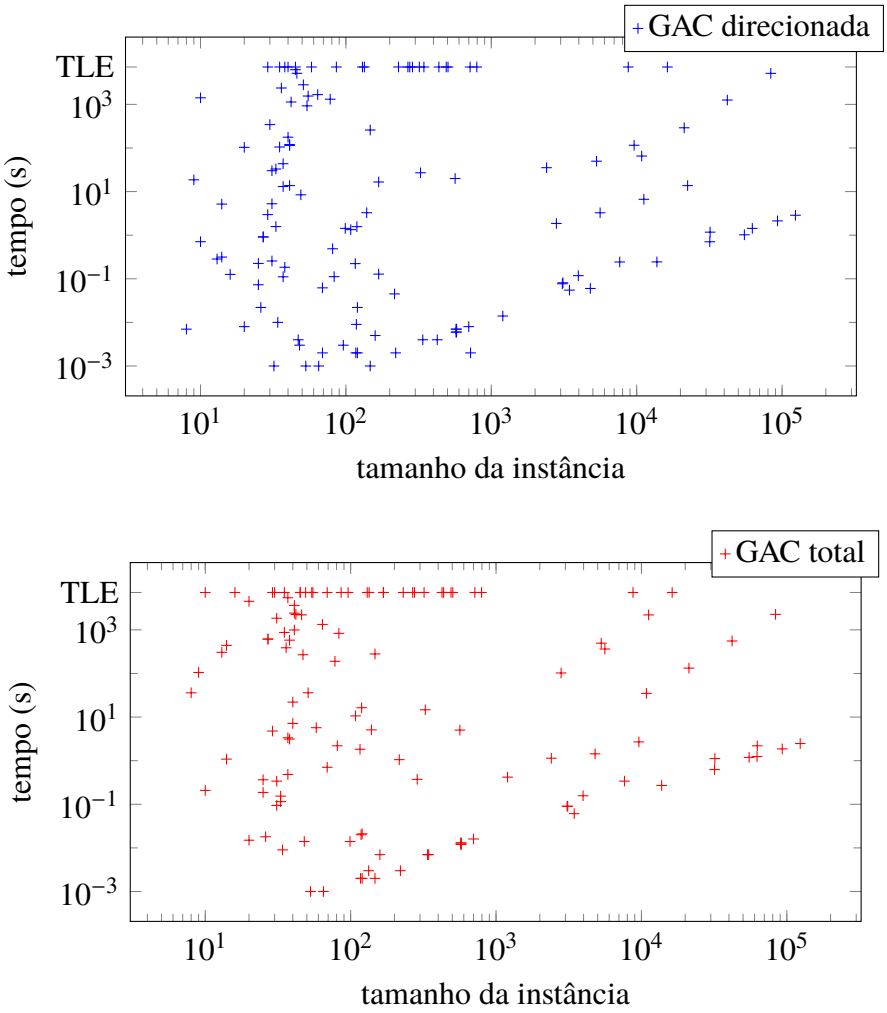


Figura 5.7: Relação entre o tamanho da instância e o tempo de execução do otimizador `OGRe` utilizando o contrator GAC direcionado (gráfico superior, em azul) e o contrator GAC total (gráfico inferior, em vermelho), com limite de tempo de 7200s.

instância	$\|C\|$	ε_{AR}	Δ	solução	EA	ER	tempo (s)	ótimo
ex4_1_4	8	10^{-2}	1	0	**0**	-	36,05	0
rbrock	8	10	10^{-1}	0	**0**	-	0	0
ex4_1_6	9	10^{-4}	10^{-3}	6,9425	**0,0575**	**0,8%**	105,4	7
ex4_1_7	10	10^{-1}	10^{-1}	-3,9062	**3,5938**	47,9%	TLE	-7,5
ex4_1_8	10	10^{-3}	10^{-2}	-16,7338	**0,0051**	**0%**	0,208	-16,739
ex4_1_5	13	10	1	-19938,9	19938,9	-	305,3	0
ex4_1_3	14	10^{-1}	10^{-1}	-4,2948	439,377	99%	1,085	-443,672
ex7_2_6	14	10^{-4}	10^{-3}	-83,2503	**0**	**0%**	439,6	-83,2503
ex4_1_1	16	10^{-1}	10^{-2}	-0,0840	**7,4033**	98,9%	TLE	-7,4873
ex9_2_8	20	1	10^{-3}	1,5005	**0,0005**	**0%**	0,015	1,5
sample	20	10^{-4}	10^{-3}	726,66	**0,0223**	**0%**	4517	726,679
ex4_1_9	25	10^{-2}	10^{-1}	-6,2861	**0,7780**	**14,1%**	0,365	-5,5080
ex8_1_6	25	10^{-4}	10^{-5}	-10,086	**0**	**0%**	0,187	-10,086
ex9_2_4	26	10^{-4}	10^{-5}	0,5	**0**	**0%**	0,018	0,5
mhw4d	26	5	10^{-1}	0	**0,0293**	100%	0	0,0293
ex7_2_2	27	10^{-4}	10^{-3}	-0,4518	**0,0629**	16,2%	612,8	-0,3888
ex8_1_8	27	10^{-4}	10^{-3}	-0,4518	**0,0629**	16,2%	615,6	-0,3888
ex5_4_2	29	10^{-1}	10^{-3}	7158,75	353,48	**4,7%**	TLE	7512,23
wall	29	10^{-3}	10^{-3}	-1	**2**	200%	4,779	1
house	30	10^{-4}	10^{-3}	-4500,1	**0,0893**	**0%**	TLE	-4500
ex7_3_2	31	10^{-4}	10^{-5}	1,0897	**0,0002**	**0%**	1849	1,0899
ex9_2_2	31	10^{-2}	10^{-1}	100	**0**	**0%**	0,094	100
ex9_2_5	31	10^{-4}	10^{-4}	4,9997	**0,0003**	**0%**	0,339	5
nemhaus	32	1	0,5	31	**0**	**0%**	0	31
ex9_1_4	33	10^{-4}	10^{-4}	-37,0024	**0,0024**	**0%**	0,115	-37
ex9_1_5	33	10^{-1}	10^{-4}	-1	**0**	**0%**	0,154	-1
ex9_1_8	34	10^{-4}	10^{-5}	-3,2502	**0,0002**	**0%**	0,009	-3,25
ex3_1_4	35	10^{-2}	10^{-4}	-3,9981	**0,002**	**0%**	TLE	-4
ex9_2_6	35	10^{-1}	10^{-2}	1,0962	**2,0962**	210%	872,0	-1
ex2_1_1	36	10^{-2}	10^{-3}	1,1703	18,1703	107%	388,6	-17
ex3_1_1	37	10^{-1}	10^{-3}	5218,12	1831,12	26%	5445	7049,25
ex3_1_3	37	10^{-4}	10^{-4}	-310,003	**0,0026**	**0%**	3,343	-310
ex9_2_7	37	10^{-1}	10^{-4}	17,0005	**0,0005**	**0%**	0,483	17
ex14_1_5	38	10^{-1}	10^{-4}	-0,1875	**0,1875**	-	580,7	0
ex2_1_2	38	5	10^{-2}	-212,688	**0,3122**	**0,1%**	3,125	-213
ex8_1_7	40	10	10^{-1}	$+\infty$	$+\infty$	-	22,07	0,0293
ex9_2_1	40	10^{-4}	10^{-4}	16,9996	**0,0004**	**0%**	7,134	17
ex5_2_2_case1	41	1	0,5	-769,892	369,89	92,5%	994,8	-400
ex5_2_2_case2	41	1	0,5	-684,812	84,8116	**14,1%**	3635	-600
ex5_2_2_case3	41	1	0,5	-328,748	421,25	56,2%	2414	-750
chance	42	10^{-4}	10^{-3}	28,4095	**1,4849**	**5%**	2256	29,8944
ex7_2_3	45	10^{-1}	10^{-1}	4091,012	2958,24	42%	TLE	7049,25
haverly	45	1	10^{-1}	-369,480	30,520	**7,6%**	TLE	-400
ex5_2_4	46	10^{-1}	10^{-1}	-506,884	56,8845	**12,6%**	2204	-450

Tabela 5.2: Resultados utilizando o contrator GAC total - parte 1.

instância	$\|C\|$	ε_{AR}	Δ	solução	EA	ER	tempo (s)	ótimo
ex7_3_3	47	10^{-1}	10^{-1}	0,5893	**0,2282**	27,9%	269,1	0,8175
process	48	1	10^{-3}	-13765,7	12604,3	1085%	0,014	-1161,34
ex14_1_1	49	10^{-1}	10^{-1}	13,096	13,096	-	TLE	0
himmel11	51	10^{-1}	10^{-3}	-30666,9	**1,4019**	**0%**	36,07	-30665,5
ex7_2_10	53	10	10^{-1}	0,1	**0**	**0%**	0,001	0,1
alkyl	54	10^{-2}	10^{-3}	-1,5232	**0,2419**	**13,7%**	TLE	-1,7650
dispatch	55	10^{-4}	10^{-4}	3155,3	**0,011**	**0%**	TLE	3155,29
ex9_2_3	58	10^{-1}	10^{-4}	0,0006	**0,0006**	-	5,704	0
ex7_2_5	64	10^{-4}	10^{-3}	10121,7	**0,7457**	**0%**	1332	10122,5
ex3_1_2	65	5	1	-32217,4	1551,89	**5,1%**	0,001	-30665,5
ex2_1_3	69	10^{-2}	10^{-1}	-3,9986	11,0014	73,3%	TLE	-15
ex2_1_4	69	5	1	-11,721	**0,7208**	**6,6%**	0,705	-11
ex5_3_2	78	10^{-2}	1	1,8527	**0,0115**	**0,6%**	190	1,8642
circle	81	10^{-4}	10^{-5}	4,5742	**0**	**0%**	2,196	4,5742
ex7_3_1	83	10	1	4818,73	4818,39	10000%	830,3	0,3417
sambal	86	10^{-1}	10^{-3}	3,9562	**0,012**	**0,3%**	TLE	3,9682
ship	96	5	10^{-1}	2,1282	**2,1282**	-	TLE	0
ex8_4_1	99	5	10^{-2}	0	**0,6186**	100%	0,014	0,6186
ex7_3_4	108	5	1	0	**6,2746**	100%	10,65	6,2746
hydro	116	1	10^{-2}	3018718	1348226	30,9%	1,829	4366944
ex14_2_5	117	5	10^{-2}	0	**0**	-	0,002	0
ex7_2_1	118	1	10^{-1}	-411,251	1638,46	134%	0,02	1227,21
ex7_3_5	119	10	1	0	**1,2029**	100%	16,38	1,2029
ex14_1_6	120	1	0,5	0,1577	**0,3033**	65,8%	0,021	0,4610
immun	120	1	10^{-3}	0	**0**	-	0,002	0
ex8_4_5	130	10^{-2}	10^{-3}	0,0004	**0**	**14%**	TLE	0,0003
himmel16	133	5	1	-3,5	**2,6340**	304%	0,003	-0,8660
harker	134	10	10^{-1}	$2{,}5 \cdot 10^{17}$	$2{,}5 \cdot 10^{17}$	$3 \cdot 10^{14}$%	TLE	-986,51
ex2_1_6	139	5	1	-251,357	212,357	545%	5,091	-39
ex14_1_2	147	1	10^{-2}	-0,0647	**0,0647**	-	280,2	0
ex4_1_2	147	10	10^{-1}	$-3{,}3 \cdot 10^{10}$	$3{,}3 \cdot 10^{10}$	$5 \cdot 10^{7}$%	0,002	-663,5
ex8_4_2	159	5	10^{-2}	0	**0,4852**	100%	0,007	0,4852
ex2_1_8	168	5	1	25453,8	9814,8	62,8%	TLE	15639
meanvar	168	1	10^{-3}	6,7883	**1,5449**	29,5%	TLE	5,2434
ex2_1_5	216	5	1	-299,34	31,3248	**11,7%**	1,056	-268,015
linear	220	10	1	-2000	2089	2347%	0,003	89
prolog	230	5	10^{-2}	0	**0**	-	TLE	0
abel	267	5	1	$1{,}4 \cdot 10^{21}$	$1{,}4 \cdot 10^{21}$	$6 \cdot 10^{18}$%	TLE	225,195
ex8_4_3	275	10^{-3}	10^{-3}	0,0059	**0,0012**	26,3%	TLE	0,0046
chakra	287	5	1	-233,408	54,274	30,3%	0,373	-179,134
ex5_3_3	320	10	10^{-1}	7,5962	**4,3622**	135%	TLE	3,2340
minlphi	326	5	10^{-2}	571,567	10,669	**1,8%**	14,69	582,236
ex8_3_9	338	10	1	-10	**9,237**	1211%	0,007	-0,7630

Tabela 5.3: Resultados utilizando o contrator GAC total - parte 2.

instância	$\|C\|$	ε_{AR}	Δ	solução	EA	ER	tempo (s)	ótimo
ex2_1_7	344	10	1	-151620	147470	3553%	0,007	-4150,41
otpop	425	10^{-4}	10^{-4}	$+\infty$	$+\infty$	-	TLE	0
ex2_1_10	436	1	0,5	511393	462075	937%	TLE	49318
ex5_2_5	490	1	10^{-3}	-4986,116	1486,12	42,5%	TLE	-3500
demo7	504	10	10^{-1}	$6{,}4 \cdot 10^7$	$6{,}6 \cdot 10^7$	4170%	TLE	-1589042
ex8_3_6	564	5	10^{-1}	$+\infty$	$+\infty$	-	5,024	-0,5000
ex8_3_3	572	10	10^{-1}	-10	**9,5834**	2300%	0,012	-0,4166
ex8_3_5	572	10	1	-10	**9,9309**	14368%	0,013	-0,0691
ex8_3_2	577	10	1	-10	**9,5877**	2325%	0,013	-0,4123
ex8_3_4	577	5	1	-10	**6,42**	179%	0,012	-3,5800
ex7_3_6	643	10^{-4}	10^{-4}	infactível	**0**	**0%**	0	infactível
ex8_3_8	700	1	0,5	-10	**6,7439**	207%	0,016	-3,2561
ex8_3_7	715	10^{-4}	10^{-4}	$+\infty$	$+\infty$	-	TLE	-1,2326
ex8_6_1	720	10	1	-22,5	**5,9225**	20,8%	0	-28,423
ex8_3_10	795	5	1	-4,4570	**3,0877**	226%	TLE	-1,3693
camshape100	1200	10	1	-5,3006	**1,0164**	23,7%	0,419	-4,2841
camshape200	2400	5	1	-5,3627	**1,0842**	25,3%	1,139	-4,2785
catmix100	2803	10	1	-0,005	**0,0431**	89,6%	102,7	-0,0481
qp4	3082	5	1	0	**0,0008**	100%	0,09	0,0008
qp5	3110	5	10^{-1}	0	**0,4315**	100%	0,09	0,4315
elec25	3450	1	10^{-2}	$+\infty$	$+\infty$	-	0,061	243,812
qp2	3969	10	1	$+\infty$	$+\infty$	-	0,157	0,0008
camshape400	4800	5	10^{-2}	-5,4007	**1,125**	26,3%	1,436	-4,2757
qp3	5294	10	10^{-1}	-3,9487	**3,9496**	$5 \cdot 10^3$%	495	0,0008
catmix200	5603	5	1	-0,0025	**0,0456**	94,8%	363,7	-0,0481
qp1	7644	10	1	$+\infty$	$+\infty$	-	0,34	0,0008
turkey	8751	10^{-4}	10^{-4}	$+\infty$	$+\infty$	-	TLE	-29330
camshape800	9600	10	10^{-1}	-5,421	**1,1463**	26,8%	2,695	-4,2747
gasoil50	10825	1	10^{-1}	0	**0,0052**	100%	34,91	0,0052
catmix400	11203	10	1	-0,0013	**0,0468**	97,4%	2205	-0,0481
elec50	13775	5	10^{-1}	$+\infty$	$+\infty$	-	0,27	1055,18
pinene50	16270	10^{-4}	10^{-4}	$+\infty$	$+\infty$	-	TLE	19,8722
gasoil100	21225	1	10^{-2}	0	**0,0052**	100%	132,5	0,0052
catmix800	22403	5	1	$+\infty$	$+\infty$	-	0	-0,0481
minsurf25	31825	10^{-1}	10^{-4}	$+\infty$	$+\infty$	-	0,626	$1{,}7 \cdot 10^7$
torsion25	31988	1	0,5	$+\infty$	$+\infty$	-	1,119	-0,4175
gasoil200	42025	10	10^{-1}	0	**0,0052**	100%	555,5	0,0052
elec100	55050	5	10^{-1}	$+\infty$	$+\infty$	-	1,188	4448,35
minsurf50	62425	10^{-2}	1	$+\infty$	$+\infty$	-	1,242	$6{,}8 \cdot 10^7$
torsion50	62638	1	0,5	$+\infty$	$+\infty$	-	2,194	-0,4181
gasoil400	83625	5	10^{-2}	0	**0,0052**	100%	2266	0,0052
minsurf75	93025	10^{-2}	10^{-4}	$+\infty$	$+\infty$	-	1,862	$1{,}5 \cdot 10^8$
minsurf100	123625	10^{-1}	10^{-4}	$+\infty$	$+\infty$	-	2,473	$2{,}7 \cdot 10^8$

Tabela 5.4: Resultados utilizando o contrator GAC total - parte 3.

5.5 Conclusão

Este capítulo apresentou o otimizador `OGRe`, que implementa o método `RAC_direcionada_aprox`, apresentado na Seção 4.4.1, juntamente com uma biblioteca multi-intervalar. Resultados experimentais também foram apresentados, aplicando-se o `OGRe` em um conjunto de 130 instâncias de problemas de otimização global propostos no *benchmark* COCONUT.

O otimizador `OGRe` caracteriza uma contribuição prática desta pesquisa, disponibilizado com código-fonte aberto em Derenievicz (2018). Alguns detalhes de sua implementação, relatados neste capítulo, também constituem contribuições desta tese. Em particular, discutiu-se sobre a multiplicação de intervalos abertos contendo 0 e a interseção de intervalos degenerados, quando utilizada uma tolerância para a verificação de igualdade de pontos flutuantes. Também apresentou-se uma definição alternativa de divisão intervalar que mantém o contrator GAC completo e não utiliza a multiplicação intervalar como parte do processamento. Neste capítulo, também detalhou-se os procedimentos utilizados pelo método `RAC_direcionada_aprox`, principalmente em relação às heurísticas utilizadas na fase de ramificação (Seção 5.3.2).

Em relação aos experimentos, pode-se concluir que o método proposto é capaz de encontrar a solução de instâncias NCOP mesmo com uma grande tolerância ε_{AR}, embora sua aplicabilidade em instâncias grandes não esteja clara. A implementação preliminar do otimizador `OGRe` não considera técnicas avançadas de retrocesso, aprendizado ou busca local, de forma que não se pode compará-lo a otimizadores do estado da arte. Mesmo assim, as heurísticas propostas alcançaram bons resultados no caso geral.

Vale ressaltar que todas as instâncias do experimento apresentaram largura epífita 1, mas esta não foi uma condição de escolha de instâncias, o que pode sugerir que a classe de problemas que possuem uma decomposição Epífita representa um grande número de problemas de otimização reais. Outro aspecto interessante observado nos experimentos é que não houve crescimento exponencial na representação de multi-intervalos; em geral, cada multi-intervalo foi composto por um ou dois intervalos, resultantes da operação de radiciação intervalar.

6 CONCLUSÃO

Na Seção 1.3, definiu-se como objetivos desta tese identificar uma condição suficiente para otimização global sem retrocesso, introduzir um parâmetro de medida de complexidade que identifique o nível de pré-processamento que alcança essa condição e caracterizar subclasses de problemas para os quais é possível alcançar essa condição de forma eficiente. O Teorema 4.9 estabelece a k-consistência relacional direcionada forte como condição suficiente para otimização global sem retrocesso de decomposições k-Epífitas; por outro lado, toda instância NCOP possui uma decomposição k-Epífita, para algum $k \geq 1$. Assim, o parâmetro k, definido pela largura epífita do hipergrafo de restrições, mede o nível de consistência relacional necessária para que a rede torne-se livre de retrocesso. Em particular, decomposições 1-Epífitas constituídas de uma única árvore, isto é, redes cujo hipergrafo é Berge-acíclico, são resolvidas de forma eficiente alcançando-se apenas a consistência de arco generalizada (Corolário 4.1). Desta forma, conclui-se que os objetivos desta tese foram atingidos.

6.1 Contribuições

As contribuições desta tese são detalhadas a seguir.

- Nos Capítulos 2 e 3, apresenta-se um levantamento dos principais conceitos e técnicas utilizadas nos campos de satisfação de restrições e análise intervalar. Essa revisão é apresentada de maneira didática e linear, objetivando uma melhor assimilação do conteúdo por parte do leitor. Esse material caracteriza-se, dessa forma, como uma fonte de consulta introdutória ao tema de processamento de restrições;

- Nas Seções 4.1 e 4.2, a ideia de busca sem retrocesso em CSPs, seguindo uma ordenação de variáveis $(x_1, \ldots, x_n)$, é estendida a problemas de otimização através da codificação ternária da rede e da função objetivo, definindo uma variável raiz x_1 pela qual deve-se iniciar a busca;

- Na Seção 4.2, introduz-se um parâmetro de ciclicidade da rede de restrições chamado grau de interseção, classificando instâncias NCOP em três classes primárias: instâncias acíclicas, instâncias parcialmente acíclicas e instâncias cíclicas. O grau de interseção difere do parâmetro de interseções maximais proposto por Jégou (1993);

- Na Seção 4.2.1, prova-se que um hipergrafo $\mathcal{H}$ tem grau de interseção no máximo 1 se, e somente se, $\mathcal{H}$ é Berge-acíclico (Teorema 4.1) e estende-se a condição de solução livre de retrocesso de CSPs a problemas de otimização, mostrando que GAC direcionada é suficiente para resolver instâncias Berge-acíclicas (Corolário 4.1);

- Na Seção 4.2.2, prova-se que RAC direcionada é suficiente para resolver instâncias NCOP com grau de interseção no máximo 2 (Teorema 4.2);

- Na Seção 4.3.1, define-se o conceito de decomposição Epífita, que caracteriza a classe de problemas parcialmente acíclicos proposta na Seção 4.2. Prova-se que um hipergrafo possui uma decomposição Epífita se, e somente se, tem grau de interseção no máximo 2 (Teorema 4.3). Conclui-se que RAC direcionada é suficiente para resolver instâncias que possuem decomposições Epífitas (Corolário 4.2);

- Na Seção 4.3.2, mostra-se que uma rede $\mathcal{R}$ possui uma decomposição Epífita se, e somente se, todo subconjunto S de restrições nessa rede possui pelo menos uma variável (além da variável raiz) que ocorre em uma única restrição em S (Teorema 4.4). Adapta-se o método de encontrar ordenação de largura mínima proposto por Freuder (1982) para obter uma ordenação com grau de interseção no máximo 2 (Algoritmo 18) em tempo linear (Teorema 4.5). Argumenta-se que uma rede pode ter distintas decomposições Epífitas e que um problema de otimização pode ser remodelado adicionando-se novas constantes a fim de tornar sua decomposição possível (Exemplo 4.5);

- Na Seção 4.4, propõe-se um método de ramificação e poda para aproximar RAC direcionada sem alterar a estrutura do hipergrafo (Algoritmo 19). Para isso, introduz-se os conceitos de satisfazibilidade com tolerância e $\varepsilon_{AR}-$factível. O método proposto é uma variante do algoritmo de ramificação e poda intervalar usual, provendo uma heurística de escolha de variáveis na fase de ramificação de modo a minimizar a tolerância necessária para que a rede torne-se RAC direcionada, visto que esta é condição suficiente para otimização global sem retrocesso;

- Na Seção 4.5, estende-se o conceito de decomposição Epífita a hipergrafos que não possuem ordenação de arestas com grau de interseção no máximo 2, através do conceito de decomposição k-Epífita. Define-se o parâmetro largura epífita como uma extensão natural da largura de grafos proposta por Freuder (1982) e mostra-se a relação direta entre largura epífita e decomposição k-Epífita (Teorema 4.7). Prova-se que decomposição Epífita e decomposição 1-Epífita são conceitos equivalentes, isto é, ordenações de arestas com grau de interseção no máximo 2 podem ser convertidas em ordenações de vértices com largura epífita 1, e vice-versa (Teorema 4.6). Mostra-se que uma rede $\mathcal{R}$ possui uma decomposição k-Epífita se, e somente se, todo subconjunto S de restrições nesta rede possui pelo menos uma variável (além da variável raiz) que ocorre em no máximo k

restrições em S (Teorema 4.8) e aplica-se o algoritmo de Freuder (1982) para encontrar a largura epífita de um hipergrafo (Algoritmo 20). Por fim, prova-se que instâncias que possuem decomposições k-Epífitas são resolvidas sem retrocesso se alcançada k-consistência relacional direcionada forte (Teorema 4.9). Como toda instância NCOP possui uma decomposição k-Epífita, para algum $k \geq 1$, então k-consistência relacional direcionada forte é condição suficiente para otimização global sem retrocesso;

- Na Seção 4.6, prova-se que a largura epífita de um hipergrafo não é definida pela largura do seu grafo primal ou dual (Teorema 4.10), ou pela largura de hipergrafo (Teorema 4.11), ou pelas larguras de árvore e de hiperárvore (Teorema 4.12);

- No Capítulo 5, apresenta-se o otimizador `OGRe`, fundamentado nos resultados teóricos obtidos nesta tese. Em particular, na Seção 5.2 são propostas algumas definições de operações que tornam o contrator GAC ternário completo, e, na Seção 5.3.2, propõe-se um conjunto de heurísticas para a escolha de variáveis no otimizador `OGRe`, mas que também podem ser consideradas em outros métodos intervalares.

6.2 Limitações e questões em aberto

A principal limitação prática dos resultados aqui apresentados surge do fato que alcançar k-consistência relacional direcionada, para qualquer $k \geq 1$, exige a adição de novas restrições na rede, alterando a largura epífita do hipergrafo de restrições. A mesma limitação ocorre com os resultados teóricos de Freuder (1982) e Jégou (1993).

Por outro lado, o algoritmo proposto para aproximar consistência de arco relacional direcionada exige a definição dos parâmetros ε_{AR} e Δ, isto é, a tolerância permitida na satisfação de restrições que compõem os ciclos da rede e o comprimento mínimo ao qual é permitido a bissecção de domínios no método de ramificação e poda. Estes parâmetros não são trivialmente definidos no caso geral e a qualidade da solução obtida pelo método, isto é, a diferença entre o valor encontrado e o ótimo global de cada instância, depende destes parâmetros.

O otimizador `OGRe` não implementa diversas técnicas comumente utilizadas em otimizadores do estado da arte, como técnicas de busca local, aprendizado por conflitos, retrocesso não cronológico, heurísticas, entre outros. Além disso, a biblioteca multi-intervalar proposta considera apenas os operadores algébricos básicos e não implementa uma estrutura de dados sofisticada para representar multi-intervalos. A consistência de arco generalizada não é alcançada em sistemas computacionais, visto que as bordas podem ser arredondadas para o número de máquina mais próximo.

Três questões principais podem ser levantadas acerca dos resultados e contribuições apresentadas nesta tese:

1. Qual a incidência de problemas de otimização reais com largura epífita maior que 1?

2. A largura e a altura epífita de uma rede também são parâmetros de complexidade relevantes em relação aos métodos de otimização da comunidade de programação matemática e outros métodos de otimização?

3. Qual decomposição Epífita de uma instância NCOP é a melhor para a execução de otimizadores e para alcançar-se RAC mais rapidamente?

Em relação à questão 3, lembra-se que o Algoritmo 18 (`ordem_de_grau_2`) não especifica qual variável de grau 1 deve ser escolhida em cada iteração da construção da decomposição, pois a completude do método independe desta escolha. No entanto, diferentes escolhas podem resultar em diferentes decomposições Epífitas.

6.3 Trabalhos futuros

As questões levantadas na Seção 6.2 são os principais pontos para pesquisa futura. Em particular, propõe-se abordar as questões 1 e 2 ampliando os experimentos apresentados no Capítulo 5. Para a questão 3, é possível obter diversas decomposições Epífitas com diferentes escolhas de variável de grau 1 no Algoritmo 18 e, assim, comparar a execução do `OGRe` com diferentes representações da mesma instância, analisando a execução de cada uma delas na tentativa de estabelecer heurísticas de escolhas de boas decomposições Epífitas no caso geral.

Propõe-se também incrementar o otimizador `OGRe` com técnicas que aceleram a busca ou podam o espaço de busca de maneira mais eficiente, como retrocesso não cronológico e técnicas de busca local. Além disso, outras heurísticas de escolhas de variáveis e partições de domínios podem ser implementadas. Alternativamente, as heurísticas de ramificação de variáveis propostas nesta tese, que minimizam a tolerância necessária para alcançar a consistência relacional da rede, podem ser implementadas em otimizadores do estado da arte e comparadas com outras heurísticas propostas na literatura.

Por fim, propõe-se utilizar o método de ramificação e poda intervalar aqui proposto para abordar problemas de otimização sobre variáveis discretas, como programação inteira mista ou otimização combinatória, e também para incrementar otimizadores do estado da arte em relação a busca local, pré-processamento ou limitante inferior de solução ótima.

REFERÊNCIAS

Araya, I., Neveu, B. e Trombettoni, G. (2012). An interval extension based on occurrence grouping. *Computing*, 94(2-4):173–188.

Araya, I. e Reyes, V. (2016). Interval branch-and-bound algorithms for optimization and constraint satisfaction: a survey and prospects. *Journal of Global Optimization*, 65(4):837–866.

Araya, I., Trombettoni, G., Neveu, B. e Chabert, G. (2014). Upper bounding in inner regions for global optimization under inequality constraints. *Journal of Global Optimization*, 60(2):145–164.

Arnborg, S. e Proskurowski, A. (1989). Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11 – 24.

Batnini, H., Michel, C. e Rueher, M. (2005). Mind the gaps: A new splitting strategy for consistency techniques. Em van Beek, P., editor, *Proceedings of the 11th Principles and Practice of Constraint Programming. CP 2005*, páginas 77–91. Springer.

Beeri, C., Fagin, R., Maier, D., Mendelzon, A., Ullman, J. e Yannakakis, M. (1981). Properties of acyclic database schemes. Em *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '81, páginas 355–362, New York, NY, USA. ACM.

Beeri, C., Fagin, R., Maier, D. e Yannakakis, M. (1983). On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513.

Benhamou, F., Goualard, F., Granvilliers, L. e Puget, J.-F. (1999). Revising hull and box consistency. Em *Proceedings of the 15th International Conference on Logic Programming*, páginas 230–244. MIT press.

Benhamou, F. e Granvilliers, L. (2006). Continuos and interval constraints. Em Rossi, P. F., van Beek e Walsh, T., editores, *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, páginas 569–601. Elsevier, New York, NY, USA.

Benhamou, F., McAllester, D. e van Hentenryck, P. (1994). CLP(intervals) revisited. Em *Proceedings of the 1994 International Symposium on Logic Programming*, ILPS '94, páginas 124–138, Cambridge, MA, USA. MIT Press.

Benhamou, F. e Older, W. J. (1992). Applying interval arithmetic to real, integer and boolean constraints. Relatório técnico, BNR, Bell Northern Research.

Benhamou, F. e Older, W. J. (1997). Applying interval arithmetic to real, integer, and boolean constraints. *The Journal of Logic Programming*, 32(1):1 – 24.

Berge, C. (1985). *Graphs and Hypergraphs*. Elsevier Science Ltd., Oxford, UK, UK.

Bertsekas, D. P. (1999). *Nonlinear programming*. Athena scientific Belmont.

Bertsimas, D. e Tsitsiklis, J. (1997). *Introduction to Linear Optimization*. Athena Scientific, 1st edition.

Bessiere, C. (2006). Constraint propagation. Em Rossi, F., van Beek, P. e Walsh, T., editores, *Handbook of Constraint Programming*, capítulo: 3. Elsevier.

Bessière, C. e Régin, J.-C. (2001). Refining the basic constraint propagation algorithm. Em *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, páginas 309–315, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bessiere, C., Stergiou, K. e Walsh, T. (2008). Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, 172(6):800 – 822.

Bessière, C. (1994). Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1):179 – 190.

Bhurjee, A. K. e Panda, G. (2012). Efficient solution of interval optimization problem. *Mathematical Methods of Operations Research*, 76(3):273–288.

Brouwer, A. E. e Kolen, A. W. J. (1980). A super-balanced hypergraph has a nest point. Relatório Técnico ZW146, Math. centr. report, Amsterdam.

Chabert, G., Trombettoni, G. e Neveu, B. (2004). New light on arc consistency over continuous domains. Relatório Técnico RR-5365, INRIA.

Cioffi, A. (2005). Bialbero di casorzo. Retirado de Wikimedia Commons website: `https://goo.gl/bZwLHR`, sob a licença CC BY-SA 3.0, disponível em `https://goo.gl/L3bYmf`. Acessado em 17/03/2018.

Cohen, D. A. e Jeavons, P. G. (2017). The power of propagation: when gac is enough. *Constraints*, 22(1):3–23.

Cruz, J. e Barahona, P. (2001). *Global Hull Consistency with Local Search for Continuous Constraint Solving*, páginas 349–362. Springer Berlin Heidelberg, Berlin, Heidelberg.

Csendes, T. e Ratz, D. (1997). Subdivision direction selection in interval methods for global optimization. *SIAM Journal on Numerical Analysis*, 34(3):922–938.

Dantzig, G. (1963). *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ.

Davis, E. (1987). Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331.

Davis, M., Logemann, G. e Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397.

Davis, M. e Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215.

Dawson, B. (2012). Comparing floating point numbers. `https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/`. Acessado em 25/03/2018.

Dechter, R. (1992). From local to global consistency. *Artificial Intelligence*, 55(1):87–107.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, USA.

Dechter, R. e Pearl, J. (1987a). The cycle-cutset method for improving search performance in ai applications. Em *Proceedings of the 3rd IEEE Conference on AI Applications*, páginas 224–230, Orlando FL.

Dechter, R. e Pearl, J. (1987b). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1 – 38.

Dechter, R. e Pearl, J. (1989). Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353 – 366.

Dechter, R. e Rish, I. (1994). Directional resolution: The davis-putnam procedure, revisited. Em *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, páginas 134–145. Morgan Kaufmann.

Dechter, R. e van Beek, P. (1997). Local and global relational consistency. *Theoretical Computer Science*, 173(1):283 – 308.

Derenievicz, G. A. (2014). Otimização de sistemas intervalares não lineares acíclicos. Dissertação de Mestrado, Programa de Pós-Graduação em Informática - UFPR, Curitiba - PR.

Derenievicz, G. A. (2018). OGRe - otimização global relaxada, em gitlab C3SL - centro de computação científica e software livre. `https://gitlab.c3sl.ufpr.br/gaderenievicz/OGRe`. Acessado em 03/04/2018.

Eaton, J. W., Bateman, D., Hauberg, S. e Wehbring, R. (2017). *GNU Octave version 4.2.2 manual: a high-level interactive language for numerical computations*.

Edmonson, W. W. e van Emden, M. H. (2008). Interval semantics for standard floating-point arithmetic. *Computing Research Repository*, abs/0810.4196.

Fagin, R. (1983). Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550.

Faltings, B. (1998). Arc consistency for continuous variables. *Artificial Intelligence*, 65: 363–376.

Faltings, B. e Gelle, E. M. (1997). Local consistency for ternary numeric constraints. Em *Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan*, páginas 392–397.

Fourier, J. (1824). *Analyse des travaux de l'Académie royale des sciences, pendant l'année 1824*. imprimerie de Firmin Didot.

Franzle, M., Herde, C., Teige, T., Ratschan, S. e Schubert, T. (2007). Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236.

Freuder, E. C. (1978). Synthesizing constraint expressions. *Communications of the ACM*, 21(11):958–966.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32.

GAMS (2002). Global library. `http://www.gamsworld.org/global/globallib.htm`.

Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition.

Gottlob, G., Greco, G. e Scarcello, F. (2014). *Treewidth and Hypertree Width*, página 3–38. Cambridge University Press.

Gottlob, G., Grohe, M., Musliu, N., Samer, M. e Scarcello, F. (2005). Hypertree decompositions: Structure, algorithms, and applications. Em Kratsch, D., editor, *Graph-Theoretic Concepts in Computer Science*, páginas 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gottlob, G., Leone, N. e Scarcello, F. (2000). A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243 – 282.

Gottlob, G., Leone, N. e Scarcello, F. (2003). Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4):775 – 808. Special Issue on PODS 2001.

Granvilliers, L., Goualard, F. e Benhamou, F. (2000). Box consistency through weak box consistency. Em *Proceedings of the International Conference on Tools with Artificial Intelligence*.

Han, C.-C. e Lee, C.-H. (1988). Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence*, 36(1):125 – 130.

Hansen, E. (1980). Global optimization using interval analysis — the multi-dimensional case. *Numerische Mathematik*, 34(3):247–270.

Hansen, E. e Walster, G. W. (2004). *Global optimization using interval analysis*. Monographs and textbooks in pure and applied mathematics. Marcel Dekker, Inc., New York.

Hascoët, L., Hossain, S. e Steihaug, T. (2013). Structured computation in optimization and algorithmic differentiation. 46:94–95.

Hestenes, M. R. (1969). Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320.

Hyvönen, E. (1989). Constraint reasoning based on interval arithmetic. Em *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Vol. 2*, IJCAI'89, páginas 1193–1198, San Francisco, USA. Morgan Kaufmann Publishers Inc.

Hyvönen, E. (1992). Constraint reasoning based on interval arithmetic: the tolerance propagation approach. *Artificial Intelligence*, 58(1):71 – 112.

Janssen, P., Jégou, P., Nouguier, B. e Vilarem, M.-C. (1989). A filtering process for general constraint-satisfaction problems: achieving pairwise-consistency using an associated binary representation. Em *IEEE Workshop on Tools for AI*.

Jégou, P. (1993). On the consistency of general constraint-satisfaction problems. Em *Proceedings of the 11th National Conference on Artificial Intelligence*, AAAI'93, páginas 114–119. AAAI Press.

Kampen, E. V. (2010). *Global Optimization using Interval Analysis: Interval Optimization for Aerospace Applications*. Tese de doutorado, Aerospace Engineering, Zutphen, The Netherlands.

Karakashian, S., Woodward, R. J., Reeson, C., Choueiry, B. Y. e Bessiere, C. (2010). A first practical algorithm for high levels of relational consistency. Em *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, páginas 101–107. AAAI Press.

Kearfott, R. B. (1992). An interval branch and bound algorithm for bound constrained optimization problems. *Journal of Global Optimization*, 2(3):259–280.

Kearfott, R. B. (2009). Globsol user guide. *Optimization Methods Software*, 24(4-5):687–708.

Kearfott, R. B. e Walster, G. W. (2002). Symbolic preconditioning with taylor models: Some examples. *Reliable Computing*, 8(6):453–468.

Khachiyan, L. (1980). Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53 – 72.

Kreinovich, V. e Bernat, A. (1994). Parallel algorithms for interval computations: An introduction. *Interval Computations*, (3):6–62.

Land, A. H. e Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):pp. 497–520.

Lebbah, Y. (2009). ICOS: a branch and bound based solver for rigorous global optimization. *Optimization Methods and Software*, 24(4-5):709–726.

Lebbah, Y., Michel, C. e Rueher, M. (2007). An efficient and safe framework for solving optimization problems. *Journal of Computational and Applied Mathematics*, 199(2):372 – 377.

Lehmer, D. H. (1957). Combinatorial problems with digital computers. Em *In Proceedings of the 4th Canadian Mathematical Congress*, páginas 160–173.

Lhomme, O. (1993). Consistency techniques for numeric CSPs. Em *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, páginas 232–238, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

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

Mackworth, A. K. (1977b). On reading sketch maps. Em *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'77, páginas 598–606, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Mackworth, A. K. e Freuder, E. C. (1985). The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65 – 74.

McShane, E. (1983). *Unified Integration*. Pure and Applied Mathematics. Elsevier Science.

Messine, F. (2005). *A Deterministic Global Optimization Algorithm for Design Problems*, páginas 267–294. Springer US, Boston, MA.

Misener, R. e Floudas, C. (2014). Antigone: Algorithms for continuous / integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59(2):503–526.

Mohr, R. e Henderson, T. C. (1986). Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225 – 233.

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95 – 132.

Moore, R. E. (1966). *Interval analysis*. Prentice-Hall Englewood Cliffs, N.J.

Moore, R. E., Kearfott, R. B. e Cloud, M. J. (2009). *Introduction to interval analysis*. Society for Industrial and Applied Mathematics, Philadelphia, USA.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L. e Malik, S. (2001). Chaff: Engineering an efficient sat solver. Em *Proceedings of the 38th Annual Design Automation Conference*, páginas 530–535, New York, USA. ACM.

Neumaier, A. (1991). *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Neumaier, A., Shcherbina, O., Huyer, W. e Vinkó, T. (2005). A comparison of complete global optimization solvers. *Mathematical Programming*, 103(2):335–356.

Ninin, J., Messine, F. e Hansen, P. (2015). A reliable affine relaxation method for global optimization. *A Quarterly Journal of Operations Research*, 13(3):247–277.

Norvig, P. e Russell, S. (2017). *Inteligência Artificial: Tradução da 3a Edição*. Elsevier Brasil.

Ortiz, G. A. (2012a). Rosenbrock's function in 3D. Retirado de Wikimedia Commons website: `https://goo.gl/z5Yojq`, sob a licença CC BY-SA 3.0, disponível em `https://goo.gl/L3bYmf`. Acessado em 17/03/2018.

Ortiz, G. A. (2012b). Sphere function in 3D. Retirado de Wikimedia Commons website: `https://goo.gl/kG35te`, sob a licença CC BY-SA 3.0, disponível em `https://goo.gl/L3bYmf`. Acessado em 17/03/2018.

Ortiz, G. A. (2012c). Styblinski-tang function. Retirado de Wikimedia Commons website: `https://goo.gl/T4zikK`, sob a licença CC BY-SA 3.0, disponível em `https://goo.gl/L3bYmf`. Acessado em 17/03/2018.

Papadimitriou, C. H. e Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Potra, F. A. e Wright, S. J. (1997). *Primal-dual interior-point methods*. SIAM.

Ratschek, H. e Rokne, J. (1988). *New Computer Methods for Global Optimization*. Halsted Press, New York, NY, USA.

Robertson, N. e Seymour, P. (1986). Graph minors. II. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322.

Rokicki, T., Kociemba, H., Davidson, M. e Dethridge, J. (2014). God's number is 20. `http://www.cube20.org`. Acessado em 17/07/2016.

Rosenbrock, H. H. (1960). An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184.

Rossi, F., van Beek, P. e Walsh, T., editores (2006). *Handbook of Constraint Programming*. Elsevier Science Inc., New York, USA.

Rump, S. M. (1999). *INTLAB — INTerval LABoratory*, páginas 77–104. Springer Netherlands, Dordrecht.

Sahinidis, N. V. (1996). Baron: A general purpose global optimization software package. *Journal of global optimization*, 8(2):201–205.

Sam, J. (1995). *Constraint consistency techniques for continuous domains*. Tese de doutorado, Lausanne.

Sam-Haroud, D. e Faltings, B. (1996). Consistency techniques for continuous constraints. *Constraints*, 1(1):85–118.

Shcherbina, O., Neumaier, A., Sam-Haroud, D., Vu, X.-H. e Nguyen, T.-V. (2003a). Benchmark COCONUT - library1. `http://www.mat.univie.ac.at/~neum/\glopt/coconut/Benchmark/Library1_new_v1.html`. Acessado em 17/03/2018.

Shcherbina, O., Neumaier, A., Sam-Haroud, D., Vu, X.-H. e Nguyen, T.-V. (2003b). *Benchmarking Global Optimization and Constraint Satisfaction Codes*, páginas 211–222. Springer Berlin Heidelberg, Berlin, Heidelberg.

Sidebottom, G. e Havens, W. S. (1992). Hierarchical arc consistency for disjoint real intervals in constraint logic programming. *Computational Intelligence*, 8.

Sotiropoulos, D. G. e Grapsa, T. N. (2001). An interval branch-and-bound algorithm for global optimization using pruning steps.

Suprajitno, H. e bin Mohd, I. (2010). Linear programming with interval arithmetic. *International Journal of Contemporary Mathematical Sciences*, 5:323–332.

Sutherland, I. E. (1963). Sketchpad: A man-machine graphical communication system. Em *Proceedings of the 1963 Spring Joint Computer Conference*, AFIPS '63 (Spring), páginas 329–346, New York, NY, USA. ACM.

Trombettoni, G., Araya, I., Neveu, B. e Chabert, G. (2011). Inner regions and interval linearizations for global optimization. Em *Proceedings of the 25th AAAI Conference on Artificial Intelligence*.

van Beek, P. e Dechter, R. (1995). On the minimality and global consistency of row-convex constraint networks. *Journal of the ACM*, 42(3):543–561.

van Beek, P. e Dechter, R. (1997). Constraint tightness and looseness versus local and global consistency. *Journal of the ACM*, 44(4):549–566.

Van Hentenryck, P. (1997). Numerica: A modeling language for global optimization. Em *Proceedings of the 15th International Joint Conference on Artificial Intelligence - Vol. 2*, IJCAI'97, páginas 1642–1647, USA. Morgan Kaufmann Publishers Inc.

van Hentenryck, P., McAllester, D. e Kapur, D. (1997). Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2).

Vu, X.-H. (2005). *Rigorous solution techniques for numerical constraint satisfaction problems*. Tese de doutorado, Lausanne.

Wallace, R. J. (1993). Why AC-3 is almost always better than AC-4 for establishing arc consistency in csps. Em *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, páginas 239–245, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Waltz, D. (1972). Generating semantic descriptions from drawings of scenes with shadows. Relatório Técnico MAC AI-271, MIT, Cambridge.

Waltz, D. (1975). Understanding line drawings of scenes with shadows. Em Winston, P. H., editor, *The Psychology of Computer Vision*, página 19–91. McGraw-Hill.

Zhang, Y. e Yap, R. H. C. (2001). Making AC-3 an optimal algorithm. Em *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, páginas 316–321, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Zhu, Y., Huang, G., Li, Y., He, L. e Zhang, X. (2011). An interval full-infinite mixed-integer programming method for planning municipal energy systems – a case study of beijing. *Applied Energy*, 88(8):2846 – 2862.