UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE DO CAMPUS ARARANGUÁ
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Lucas dos Santos de Souza Tramontin

Um *framework* baseado em restrições tabulares de domínio finito e restrições primitivas de domínio contínuo

Araranguá
2024

Lucas dos Santos de Souza Tramontin

Um *framework* baseado em restrições tabulares de domínio finito e restrições primitivas de domínio contínuo

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia de Computação submetido ao Centro de Ciências, Tecnologias e Saúde do Campus Araranguá da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Alexandre Leopoldo Gonçalves, Dr.

Araranguá
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Tramontin, Lucas dos Santos de Souza
Um framework baseado em restrições tabulares de domínio finito e restrições primitivas de domínio contínuo / Lucas dos Santos de Souza Tramontin ; orientador, Alexandre Leopoldo Gonçalves, 2024.
38 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá, Graduação em Engenharia de Computação, Araranguá, 2024.

Inclui referências.

1. Engenharia de Computação. 2. Configuração baseada em conhecimento. 3. Programação por restrições. 4. Restrição tabular. 5. Restrição por intervalos. I. Gonçalves, Alexandre Leopoldo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Computação. III. Título.

Lucas dos Santos de Souza Tramontin

Um *framework* baseado em restrições tabulares de domínio finito e restrições primitivas de domínio contínuo

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Engenharia de Computação e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Computação.

Araranguá, 26 de junho de 2024.

Documento assinado digitalmente
Jim Lau
Data: 05/07/2024 18:23:15-0300
CPF: ***.464.702-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Jim Lau, Dr.
Coordenador do Curso

Banca Examinadora:

Documento assinado digitalmente
Alexandre Leopoldo Goncalves
Data: 06/07/2024 13:08:08-0300
CPF: ***.740.349-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Alexandre Leopoldo Gonçalves, Dr.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Analucia Schiaffino Morales
Data: 05/07/2024 16:15:46-0300
CPF: ***.256.420-**
Verifique as assinaturas em https://v.ufsc.br

Profª. Analúcia Schiaffino Morales, Drª.
Avaliadora
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Olga Yevseyeva
Data: 06/07/2024 12:15:01-0300
CPF: ***.003.889-**
Verifique as assinaturas em https://v.ufsc.br

Profª. Olga Yevseyeva, Drª.
Avaliadora
Universidade Federal de Santa Catarina

Um *framework* baseado em restrições tabulares de domínio finito e restrições primitivas de domínio contínuo

A framework based on table constraints for finite domain and primitive interval constraints

Lucas dos Santos de Souza Tramontin[1] Alexandre Leopoldo Gonçalves[2]

2024, JUNHO

Resumo

A crescente demanda pela configuração de produtos por parte da indústria tem posicionado os *softwares* configuradores de produtos baseados em restrições entre as abordagens mais adequadas para atender à escalabilidade e diversidade dos cenários demandados. Pesquisas recentes sobre Problemas de Satisfação de Restrições têm trazido avanços significativos que contemplam a tarefa de configuração, como trabalhos sobre representação compacta de dados tabulares, muito comuns em bases de conhecimento de configuração de produtos. Outro tipo de dado que aparece com frequência nesse contexto, são variáveis de domínio contínuo. No entanto, a literatura sobre restrições se mostra escassa no que diz respeito a trabalhos que ligam tabelas e domínios contínuos, sendo as primeiras mais comuns em trabalhos voltados a domínios finitos. Este trabalho apresenta um *framework* para a construção de redes de restrições — incluindo modelos de produtos configuráveis — com restrições tabulares compactas sobre variáveis de domínio contínuo, por meio da junção de tabelas de domínio finito e restrições auxiliares. Para a avaliação do *framework*, são propostas três implementações baseadas em diferentes métodos de geração de restrições auxiliares, sendo uma utilizando restrições primitivas, outra utilizando uma restrição global popular na literatura, e a última utilizando uma restrição global que combina valores de domínio finito com intervalos arbitrários de domínio contínuo. Como resultados, é demonstrada a capacidade do *framework* em propiciar a integração harmoniosa de domínios contínuos e restrições tabulares, bem como a sua flexibilidade em permitir diferentes implementações de acordo com a aplicação. Também são avaliados cenários de satisfação envolvendo os conceitos integrados, para os quais o desempenho se mostra inferior aos cenários equivalentes em domínios finitos, quando factíveis. Conclui-se que o *framework* atende à demanda por domínios contínuos em tabelas, com a ressalva de que, se for possível transformar domínios contínuos em finitos sem perdas, esta se apresenta como uma opção desejável.

Palavras-chave: Configuração baseada em conhecimento. Programação por restrições. Restrição tabular. Restrição por intervalos.

[1] l.santos@grad.ufsc.br

[2] a.l.goncalves@ufsc.br

Um *framework* baseado em restrições tabulares de domínio finito e restrições primitivas de domínio contínuo

A framework based on table constraints for finite domain and primitive interval constraints

Lucas dos Santos de Souza Tramontin[3] Alexandre Leopoldo Gonçalves[4]

2024, JUNE

Abstract

The growing demand for the product configuration task by the industry has pushed constraint-based product configurator softwares to figure among the most appropriate approaches to fulfill the need for scalability and diversity seen in demanded scenarios. Recent research on constraint satisfaction problems have reached achievements whose benefits the configuration task takes advantage of, like the Smart Table Constraints, which promotes a more compact representation for tabular data, a usual format in product configuration knowledge bases. Another kind of data that is usually present in those bases are continuous domain variables. Even though, the literature on constraints is poor on research works putting both table constraints and continuous domain together, being the first ones predominantly present in works about finite domain constraint problems. This works proposes a framework to build constraint networks, including the ones for configurable product modeling, with Smart Table Constraints over continuous domain variables, doing so through the usual table constraints over finite domain and auxiliary constraints over the continuous domain variables. To evaluate the framework, three implementations are proposed, each of them having its own way to generate the auxiliary constraints. One uses primitive constraints, while another one employs a well-known global constraint, and the last one applies a global constraint able to link finite-domain values with arbitrary continuous-domain intervals. The results achieved primarily demonstrated the capacity of the proposed framework on providing a flexible integration between continuous domains and table constraints, allowing different implementations to be chosen according to the desired application. Satisfaction scenarios using the integrated concepts were also evaluated, for which the measured performance was lower than the equivalent scenarios over finite domain, when feasible. In conclusion, the framework fulfills the usage of continuous domains with tables, but a preferred alternative is to losslessly transform continuous domains into finite domains, when it's possible.

***Keywords*:** Knowledge-based configuration. Constraint programming. Table constraint. Interval constraint.

[3] l.santos@grad.ufsc.br

[4] a.l.goncalves@ufsc.br

1. INTRODUÇÃO

Os esforços das empresas por oferecer uma melhor experiência a seus clientes sobre produtos personalizados têm se demonstrado crescentes na literatura, manifestando-se através da busca por capturar ou manter novos clientes através da adaptação dinâmica de seus produtos às necessidades e ao poder de compra de clientes (MOON; CHADEE; TIKOO, 2008). Atualmente é possível encontrar uma ampla diversidade de produtos disponibilizados para customização, até mesmo pelo próprio cliente via plataformas web, como relógios[5], bicicletas[6], computadores pessoais[7], carros[8], motores e equipamentos elétricos[9] etc. Este paradigma, chamado de customização em massa, tem entre seus principais pilares as tecnologias de configuração de produto, que permitem a um usuário escolher valores para parâmetros personalizáveis no produto desejado, de acordo com restrições definidas pelo fabricante (HULGAARD, 2020; FELFERNIG *et al.*, 2014).

Nesse sentido, pesquisas por melhorias nas técnicas de configuração baseada em conhecimento têm produzido avanços contínuos, em favor da customização em massa. Tais avanços vêm desde a substituição dos clássicos configuradores baseados em regras *if/then* por representações de conhecimento baseadas em modelos, onde hoje predominam os sistemas baseados em restrições (FALKNER *et al*., 2016; JUNKER, 2006), a trabalhos mais recentes sobre requisitos de configuradores (FALKNER *et al*., 2020). Como exemplos citam-se a aceleração de diagnósticos (LE *et al*., 2023), a pré-compilação e portabilidade de bases de conhecimento (HULGAARD, 2020), a recomendação de diagnósticos (CARVALHO, 2021), a configuração inteligente de produtos e serviços (DONG *et al*., 2023), entre outros.

Entre os motivos elencados pela literatura para justificar o uso de restrições na configuração de produtos está a maturidade do paradigma, que é usado em diversas outras aplicações. Yap, Xia e Wang (2020) citam como exemplos a geração de código, otimização, sintetização de programas, robótica, web semântica, simulações etc. Também atribui-se à separação entre a representação do conhecimento e a lógica de raciocínio proporcionada pelas restrições, as seguintes vantagens da configuração com restrições: a) a redução do custo de desenvolvimento e manutenção desses sistemas; b) a conveniência para com o desenvolvimento de sistemas interativos; e c) a capacidade de prover explanação a respeito de conflitos entre as parametrizações escolhidas (FALKNER *et al*., 2016).

Falkner *et al*. (2020) pontuam alguns requisitos para configuradores que um usuário espera ter atendidos, entre eles, a apresentação clara das decisões que ainda devem ser tomadas para a conclusão da configuração. Além disso, como o apoio à tomada dessas decisões, o configurador deve impedir que o usuário escolha valores que não levam a uma solução válida, ou ao menos indicar que a escolha de tais valores leva a uma solução inválida, que não corresponde a um produto viável representado base de conhecimento.

Em sistemas baseados em restrições, satisfazer esse requisito pode ser um desafio, quando restrições mais elaboradas aparecem. Identificar se cada valor nos domínios da base de conhecimento leva a uma solução válida é uma tarefa de complexidade *NP-hard*, ou mesmo, impossível, se os domínios forem contínuos. Dessa forma, os *solvers* recorrem a técnicas de aproximação chamadas de consistências locais, que buscam maximizar a identificação de

---

[5] Configurador de relógios Rolex: https://www.rolex.com/watches/configure

[6] Configurador de bicicletas Ridley: https://www.ridley-bikes.com/en_INT/configurator

[7] Configurador de computadores pessoais Pichau: https://www.pichau.com.br/build

[8] Configurador de carros Porsche: https://configurator.porsche.com

[9] Configurador de motores Siemens: https://mall.industry.siemens.com/spice/cloudcm/configurator

valores que não levam a ao menos uma solução, preservando complexidade polinomial ao aceitar falsos-positivos (HULGAARD, 2020; JUNKER, 2006; LHOMME, 1993).

Para casos específicos, o uso de restrições globais (BELDICEANU; CONTEJEAN, 1994) e consistências locais mais fortes (MAIRY; DEVILLE; LECOUTRE, 2014) podem atenuar este inconveniente, oferecendo uma filtragem mais rigorosa de valores que não compõe soluções. No entanto, os *solvers* de Problemas de Satisfação de Restrições (do inglês *Constraint Satisfaction Problem* - CSP) tendem a direcionar seus esforços na exploração do espaço de busca, e em CSPs de domínio finito, formados por tipos de dados categóricos ou enumerados (CRUZ; BARAHONA, 2003; FAGES; CHABERT; PRUD'HOMME, 2013; FALKNER *et al*., 2020), deixando em aberto problemas na inferência de valores inválidos e na compatibilidade com dados decimais, os quais são comuns em configuradores. E mesmo para *solvers* de CSP numérico/contínuo, que trabalham com dados decimais, limitações decorrentes da necessidade de aplicar consistências dedicadas a variáveis e restrições contínuas (LHOMME, 1993) dificultam a elaboração de restrições complexas híbridas.

Restrição Tabular (do inglês *Table Constraint* - TC) é um tipo de restrição onde este problema se manifesta, sendo a literatura rica em trabalhos voltados à sua aplicação sobre CSPs de domínio finito, mas escassa na integração com domínios contínuos, como é observado na seção de trabalhos correlatos. Este trabalho propõe tratar essa deficiência através de um *framework* para transformação de restrições tabulares mistas em combinações de restrições tabulares exclusivas para domínios finitos com restrições primitivas sobre domínios contínuos. Tem em vista, assim, uma solução flexível quanto aos recursos presentes nos diferentes *solvers* CSP disponíveis atualmente. São propostas também três implementações compatíveis com o *framework*, sendo: a) uma baseada apenas em restrições simples sobre ambos os domínios, contínuos e finitos; b) uma segunda baseada em uma restrição global sobre domínios contínuos e finitos, visando otimização na filtragem; e c) uma proposta de restrição global e seu algoritmo de filtragem, para unir as melhores características das duas outras implementações.

Esta seção apresenta uma visão geral sobre configuração baseada em conhecimento, além da limitação abordada no trabalho, sendo apresentada a proposta de solução. As seções 2 e 3 discorrem sobre a fundamentação teórica, levantando os aspectos teóricos relevantes ao trabalho, e os trabalhos correlatos mais recentes. Na seção 4 o *framework* e as três propostas de implementação são apresentados, e seus resultados são discutidos na seção 5. Por fim, a última seção sumariza a pesquisa e elenca temas em aberto para trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. PROBLEMA DE SATISFAÇÃO DE RESTRIÇÕES

Satisfação de restrições é um paradigma voltado à modelagem e resolução de problemas de análise combinatória, nesse contexto chamados de Problemas de Satisfação de Restrições (CSP). Evoluiu da Programação por Restrições (do inglês *Constraint Programming* - CP), um conjunto de técnicas de Inteligência Artificial, Pesquisa Operacional e Ciência da Computação que busca resolver problemas definindo restrições como relações entre variáveis, adicionando capacidade de raciocínio sobre quais restrições devem ser satisfeitas (ROSSI; VAN BEEK; WALSH, 2006).

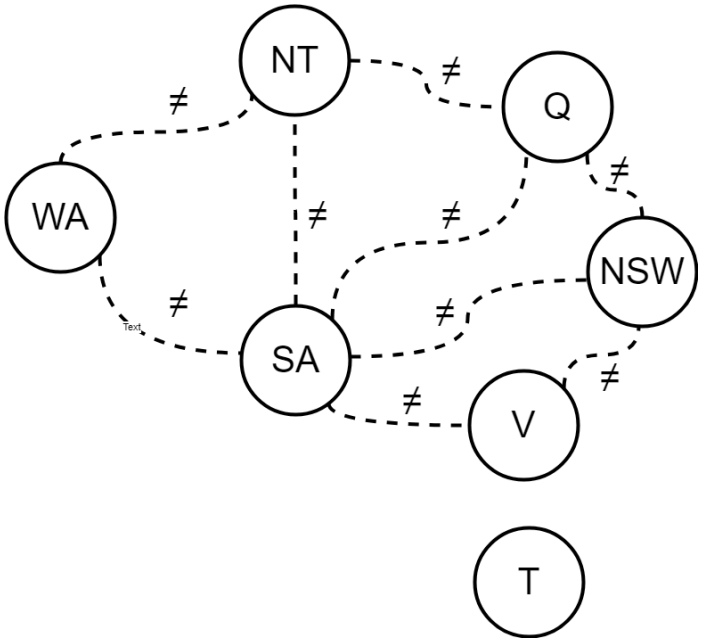
É possível definir um CSP em termos de sua rede de restrições (do inglês *constraint network* - CN) como uma tripla $P = (X, D, C)$, onde X é um conjunto finito de variáveis $X = \{x_1, x_2, \dots, x_n\}$, cada qual com um domínio definido no conjunto D, tal que $D = \{dom(x_1), dom(x_2), \dots, dom(x_n)\}$, e C é um conjunto de relacionamentos tal que $C = \{c_1, c_2, \dots, c_n\}$ entre variáveis de X, cada qual impondo restrições (do inglês *constraints*) sobre os domínios das variáveis que relaciona. Cada restrição é um par $c = (R_S, E)$, onde o escopo é

o conjunto de variáveis $E \subset X$ cujos domínios a restrição relaciona, e a relação é R_S. Esta é formada por tuplas onde cada elemento é um par $L = (x, v)$ que representa a atribuição do valor v à variável x, e recebe o nome de literal (MAIRY; DEVILLE; LECOUTRE, 2014). Dessa forma, cada tupla de R_S representa uma das combinações de literais permitidas pela restrição, de modo que a relação é um subconjunto do produto cartesiano de E. Quando a combinação é tal que seus valores estão todos presentes nos domínios das respectivas variáveis, esta é dita ser um suporte para tais valores (BESSIÈRE; RÉGIN, 1997).

Uma solução de um CSP P é um conjunto S de literais. A solução é dita completa se S possui um literal para cada variável de X, e S satisfaz C. Ou seja, cada restrição c em C possui em sua relação ao menos uma tupla cujos literais estão contidos em S (HOTZ *et al*., 2014; FREUDER; MACKWORTH, 2006). Múltiplas soluções completas podem existir para P, formando o conjunto $sol(P) = \{S_1, S_2, \ldots, S_n\}$, contendo as n soluções de P. Vale observar que $sol(P)$ é um subconjunto do produto cartesiano de D, o que implica que um CSP é equivalente a uma restrição com escopo $E = X$.

Para representar um CSP, a teoria de grafos é bastante conveniente, sendo capaz de expressar as variáveis do problema e seus relacionamentos através das restrições em uma notação simples e robusta, bem como uma convenção de visualização. Se todas as restrições têm escopo associando uma ou duas variáveis (CN binária), então o CSP pode ser representado por um grafo não direcionado simples (FREUDER; MACKWORTH, 2006). Este é definido por Netto (2022) como um par $G = (V, A)$ onde V é um conjunto discreto cujos elementos recebem o nome de vértices, e A é um conjunto de elementos definidos, cada um, em termos de um ou dois vértices, que recebem o nome de arestas. Para visualização, o esquema do grafo (também chamado simplesmente de grafo) pode ser construído desenhando-se um ponto ou pequena área delimitada para cada vértice, e uma linha para cada aresta, ligando os desenhos associados aos vértices correspondentes. Quando se deseja criar relacionamentos entre mais de dois vértices (*e.g.* em CNs não binárias), estes se chamam hiper arestas, e podem ser representados no grafo como curvas fechadas envolvendo tais vértices, formando um hipergrafo. À aresta direcionada também atribui-se o nome de arco.

Figura 1 – Grafo para a CN do problema *Australia Colors*



Fonte: HOTZ *et al*. (2014, p. 45)

Um exemplo simples que demonstra esses conceitos é o problema *Australia Colors* (HOTZ *et al*., 2014). Ele propõe colorir o mapa da Austrália com as cores do esquema RGB (do inglês, *red*, *green* e *blue*), de modo que cada estado seja colorido com uma dessas cores, e que estados adjacentes não tenham a mesma cor. A representação em grafo desse problema adota como vértices os estados, que são as variáveis deste CSP, ligados por restrições de diferença como arestas. A Figura 1 mostra o grafo resultante.

A definição dos conjuntos da tripla $P = (X, D, C)$ é então extraída do problema como o conjunto de variáveis $X = \{WA, NT, SA, Q, NSW, V, T\}$, os domínios respectivos para X como o conjunto $D = \{dom(WA) = \{r, g, b\}, dom(NT) = \{r, g, b\}, dom(SA) = \{r, g, b\}, dom(Q) = \{r, g, b\}, dom(NSW) = \{r, g, b\}, dom(V) = \{r, g, b\}, dom(T) = \{r, g, b\}\}$, e o conjunto de restrições $C = \{WA \neq NT, WA \neq SA, NT \neq SA, NT \neq Q, SA \neq Q, SA \neq NSW, SA \neq V, Q \neq NSW, NSW \neq V\}$.

Esta representação em expressões de C é chamada de representação *intensional*[10], e aceita predicados sobre os domínios das variáveis do escopo. Tem como alternativa a representação através de uma relação contendo as combinações de literais permitidos ou proibidos pela restrição, chamada de representação *extensional*[11]. Na forma *intensional*, a relação subjacente fica implícita, e as combinações permitidas podem ser inferidas como os valores que fazem o predicado ser verdadeiro. O Quadro 1 apresenta explicitamente a relação subjacente à restrição $WA \neq NT$ no problema *Australia Colors*. As demais restrições têm relações seguindo a mesma definição (BESSIÈRE, 2006; BESSIÈRE; RÉGIN, 1997). Neste trabalho, chama-se coluna uma tupla formada por todos os literais sobre a mesma variável de uma relação, na ordem em que aparecem. As colunas do Quadro 1 respeitam essa definição.

Quadro 1 – Combinações de valores permitidas para a restrição $WA \neq NT$

WA	NT
R	G
R	B
G	R
G	B
B	R
B	G

Fonte: elaborado pelo autor (2024)

Satisfazer o CSP *Australia Colors* é alcançável por um raciocínio automatizado tão simples quanto uma busca por força bruta no espaço formado pelos domínios das variáveis. Uma solução viável é qualquer das combinações encontradas nesse espaço de busca que satisfaz todas as restrições da CN. Um exemplo de solução para o problema *Australia Colors* seria o conjunto de literais $S = \{(WA, R), (NT, G), (SA, B), (Q, R), (NSW, G), (V, R), (T, R)\}$.

É fácil concluir que a resolução por força bruta se torna rapidamente inviável conforme o problema cresce. Para o problema acima, o número de combinações válidas de valores (definidas por Bessière (2006) como combinações que atribuem apenas valores presentes nos domínios das respectivas variáveis) é de $3^7 = 2187$, caracterizando complexidade exponencial. A Tabela 1 mostra o número de combinações válidas em função da quantidade variáveis e da quantidade de valores no domínio para algumas possibilidades.

[10] *Intension* é um termo em inglês sem tradução oficial. Neste texto e nos trabalhos referenciados, *intensional* descreve um conceito, o qual representa uma classe arbitrária de combinações de valores (tuplas).

[11] *Extension* é um termo em inglês sem tradução oficial. Neste texto e nos trabalhos referenciados, *extensional* enumera os membros de uma classe arbitrária de combinações de valores (tuplas).

Tabela 1 – Combinações possíveis de acordo com o tamanho do CSP

Valores por variável	5 variáveis	10 variáveis	15 variáveis	20 variáveis
5 valores	3125	9765625	30517578125	~9,537 x 10^{13}
10 valores	100000	10000000000	10^{15}	10^{20}
15 valores	759375	576650390625	~4,379 x 10^{17}	~3,325 x 10^{23}

Fonte: elaborado pelo autor (2024)

CSPs como — mas não limitado a — os utilizados em configuração de produto facilmente ultrapassam esses valores (JUNKER, 2006), de modo que técnicas para acelerar a busca de modo a torná-la viável nessas escalas têm recebido bastante atenção da literatura. Entre elas estão heurísticas para seleção conveniente de variáveis (WATTEZ *et al*., 2020; AUDEMARD; LECOUTRE; PRUD'HOMME, 2023), seleção de valores (FAGES; PRUD'HOMME, 2018) e aprendizado na busca para exploração inteligente do espaço de soluções (DECHTER, 1986), usualmente aplicadas sobre outras duas técnicas bem difundidas e usadas em conjunto: a busca por *backtrack*, e a propagação de restrições (KUMAR, 1992).

O *backtrack* consiste em estruturar a busca para que seja feita de maneira ordenada (em contraste com a força bruta), como em uma busca em profundidade no grafo de atribuições válidas de valores do CSP, e identificar quais caminhos não levam a soluções (GOLOMB; BAUMERT, 1965; CHMEISS; SAIS, 2004). Já a propagação consiste em avaliar a CN de modo a identificar e remover valores que não satisfazem uma ou mais restrições dos respectivos domínios.

Segundo Schulte e Stuckey (2008), a estratégia de busca padrão em algoritmos de *backtrack* consiste em dividir a busca em dois ramos a cada variável: no primeiro, a variável é instanciada (tem seu domínio restrito a um único valor), e no outro é removido da variável o valor de instanciação do primeiro. Para isso, o algoritmo adiciona uma nova restrição unária (cujo escopo possui apenas uma variável) que é adicionada ao CSP, gerando um outro CSP mais restrito (BESSIÈRE, 2006). A propagação após cada decisão tende a propiciar a diminuição do espaço de busca, já que faz com que o *backtrack* reconheça os valores removidos como caminhos que não levam a soluções, o que conduz a uma poda da árvore de busca (BESSIÈRE *et al*., 2009). Esta propriedade da propagação é bastante conveniente na configuração de produtos, e é melhor explorada nos próximos tópicos.

2.2. PROPAGAÇÃO DE RESTRIÇÕES

A um CSP $P = (X, D, C)$, associa-se $L = \{P' = (X, D', C) \mid D' \leq D \; e \; sol(P') = sol(P)\}$, um conjunto de CSPs com os mesmos conjuntos de variáveis e restrições, e o mesmo conjunto de soluções que P, mas com domínios tais que são menores ou iguais que os domínios de P correspondentes às mesmas variáveis.

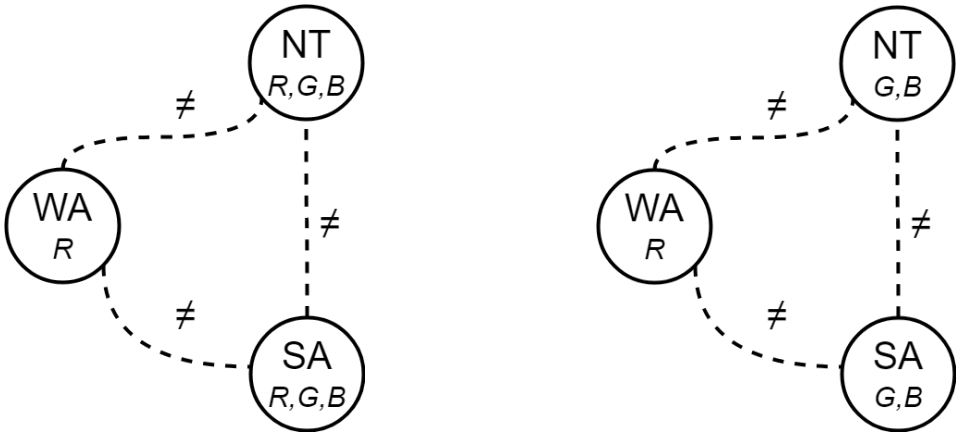
De acordo com Bessière (2006), a propagação é um processo iterativo que transforma P em P' ao filtrar valores dos domínios. Para isso, depende de um mecanismo de controle e de algoritmos de filtro. Estes são associados às restrições do CSP, e chamados de propagadores (PESANT, 2014). Quanto ao mecanismo de controle, é um algoritmo responsável por orquestrar a execução dos mesmos (SCHULTE; STUCKEY, 2014).

Propagadores implementam regras de filtragem que identificam a presença de condições suficientes para determinar que um valor a associado ao escopo de C não é parte de ao menos uma solução completa. Ou seja, o propagador remove a do domínio correspondente somente se for capaz de determinar que a não está presente na relação de soluções do CSP.

A Figura 2a mostra uma simplificação do problema da Figura 1, onde $X = \{WA, NT, SA\}$, $D = \{dom(WA) = \{R\}, dom(NT) = \{R, G, B\}, dom(SA) = \{R, G, B\}\}$ e $C = \{WA \neq NT, NT \neq SA, SA \neq WA\}$. Neste exemplo, o propagador da restrição $WA \neq NT$ identifica

que o valor R é o único valor possível para a variável WA, e conclui que R não pode estar presente em NT, pois a combinação $\{(WA, R), (NT, R)\}$ não está contida na relação adjacente à restrição. Raciocínio similar ocorre para a restrição $SA \neq WA$. Já para a restrição $NT \neq SA$, o propagador associado identifica que todos os valores nos domínios das variáveis do escopo da restrição estão contidos em ao menos um suporte da relação subjacente à restrição, e por isso não executa nenhum filtro (PESANT, 2014; KUMAR, 1992).

Figura 2 – a) o CSP original *P* e b) um CSP menor *P'*, com as mesmas soluções de *P*



Fonte: adaptado de HOTZ *et al*. (2014, p. 45)

A Figura 2b mostra o CSP resultante. É possível verificar que, para esta nova CN, o domínio de cada variável contém apenas valores permitidos pelas restrições. Dessa forma, os propagadores não mais conseguem remover valores dos domínios, a que se nomeia de ponto fixo (BESSIÈRE, 2006; CRUZ; BARAHONA, 2003).

O mecanismo de propagação é o algoritmo responsável por disparar a execução dos propagadores, identificando em quais momentos o filtro deve ser aplicado. De acordo com Kumar (1992), mecanismos de propagação têm diferentes formas de gerenciar, trocando desempenho por eficácia da filtragem. O mecanismo FC (*forward check*) monitora quando uma variável é instanciada, e executa os propagadores associados. Ao fazer isso, outras variáveis podem ser instanciadas, fazendo com que novos propagadores sejam marcados para execução pelo mecanismo de propagação. Este mecanismo é relativamente rápido, mas tende a produzir uma filtragem pouco eficaz em alguns cenários.

O RFL (do inglês *really full lookahead*) é um mecanismo de propagação que se destaca por oferecer o maior nível de filtragem oferecido pelos propagadores. Aliado com melhorias como as apresentadas por Schulte e Stuckey (2008), leva a uma busca com menor custo de processamento, e tem servido de base para implementações usadas por *solvers* como Gecode[12] e Choco-Solver[13] (PRUD'HOMME; FAGES, 2022). Este algoritmo inicia marcando todos os propagadores para execução. Quando um propagador altera o domínio de uma variável, o mecanismo busca por outros propagadores associados a esta variável que já foram executados. Propagadores encontrados são então marcados para nova execução, e o processo se repete, atingindo o ponto fixo quando mais nenhuma execução está pendente.

A visualização desse processo é descrita por Pesant (2014) sobre a CN: quando um vértice (domínio de variável) perde valores, as arestas incidentes (propagadores das restrições) são ativadas, possivelmente filtrando outros vértices em suas extremidades, o que leva a ativação de outras arestas incidentes e possíveis outros vértices sofrendo filtragem, fazendo do

[12] Código-fonte disponível em https://github.com/Gecode/gecode

[13] Código-fonte disponível em https://github.com/chocoteam/choco-solver

processo iterativo e caracterizando o nome propagação. Os domínios finitos garantem que um ponto fixo seja atingido.

Detalhes de implementação dos propagadores também podem levar a diferentes níveis de filtragem de valores que não levam a soluções. Uma das propriedades que caracterizam estes níveis em termos de sua eficácia é a consistência, descrita a seguir.

2.2.1. Consistência

A consistência é uma propriedade usada para classificar formalmente a capacidade de filtragem de diferentes algoritmos de propagação (MAIRY; DEVILLE; LECOUTRE, 2014). Idealmente, o nível de consistência desejado de alcance em uma propagação seria a consistência global, que define um CSP cujos domínios contêm apenas literais partes de uma solução completa. Uma busca por força bruta aliada a um mecanismo de propagação capaz de atingir este nível de consistência sempre encontraria uma solução na primeira tentativa. No entanto, calcular CSPs globalmente consistentes é um problema com complexidades de espaço e tempo exponenciais (KUMAR, 1992). Isto tem direcionado os esforços da literatura para definições de consistência que buscam se aproximar tanto quanto possível da consistência global, mantendo-se em complexidades polinomiais (BESSIÈRE, 2006).

De acordo com Mairy, Deville e Lecoutre (2014), a consistência de arco generalizada (do inglês *generalized arc consistency* - GAC) ou consistência de domínio (PESANT, 2014), é a forma de consistência mais difundida em CP, apresentada por Mackworth (1977) juntamente com algoritmos para impor a propriedade a um CSP. Trabalhos posteriores têm produzido novos algoritmos para GAC, aumentando sua eficiência em custo computacional, como mostrado por Yap, Xia e Wang (2020).

A propriedade GAC pode estar associada a um literal, a uma restrição e/ou ao próprio CSP. Um literal é GAC se, e somente se, possui ao menos um suporte em todas as restrições cujos escopos contêm a variável associada. Uma restrição é GAC se, e somente se, todas as variáveis têm em seus domínios apenas valores GAC. E por fim, um CSP é GAC se, e somente se, todas as suas restrições o são.

Consistências podem ser comparadas em termos de sua capacidade de filtragem. Uma consistência é dita mais forte que outra se sua capacidade de filtrar é maior ou igual, em todos os cenários aplicáveis. Mairy, Deville e Lecoutre (2014) apresentam e comparam algumas das consistências mais fortes que GAC, que têm recebido crescente atenção da literatura. Os autores pontuam que tais consistências têm custo computacional e de desenvolvimento elevados, de modo que nem sempre são justificadas pelo seu maior poder de poda. A PWC (do inglês *pairwise consistency*) e sua generalização kWC (do inglês *k-wise consistency*) buscam garantir que cada restrição permita apenas tuplas que também são permitidas por qualquer outra (ou *k*-1 outras, simultaneamente) restrição. Já a maxRPWC (do inglês *max Restricted Pairwise Consistency*) é similar a GAC, mas adicionalmente proíbe valores que são permitidos por uma restrição, mas a tupla que os permite não é compatível com as tuplas válidas de qualquer outra restrição da CN. E, por fim, a fPWC (do inglês *full pairwise consistency*) une GAC e PWC em uma única propriedade.

2.2.2. Consistências baseadas em aproximação por intervalos

Em configuradores avançados, pode ocorrer a necessidade de parametrizar atributos numéricos do configurável, como grandezas físicas, por exemplo. Isso leva à necessidade de se aplicar os chamados CSPs numéricos, onde estão envolvidas além das variáveis de domínio finito encontradas em CSPs convencionais, variáveis de domínio contínuo e restrições sobre as mesmas.

Consistências aplicáveis a domínios finitos, mesmo definições simples como GAC, se tornam impraticáveis em domínios contínuos. Uma técnica mais adequada envolve a propagação de intervalos, onde em vez de filtrar valores não suportados do domínio e manter conjuntos de valores suportados, são mantidos conjuntos de intervalos representáveis que contêm os valores suportados, e a filtragem de valores ocorre através de ajustes nos seus limites. Como o uso de algoritmos como GAC nesse esquema é inviável, são definidas consistências específicas para propagação de intervalos (LHOMME, 1993).

A consistência de envoltória (do inglês *hull consistency* - HC) aborda estes problemas ao definir uma envoltória como um domínio aproximado, expresso por um único intervalo, que é o intervalo mínimo representável contendo todos os valores do domínio real suportados pelas restrições. Essa definição elimina o problema da representação computacional usual de valores em domínios contínuos (HOUGH, 2019) ao usar a aproximação por intervalos. Além disso, elimina o problema da explosão combinatória por preservar apenas o único intervalo mínimo representável que contém todos os valores do domínio a aproximar, em vez de manter múltiplos pequenos intervalos, como definido para a consistência de intervalo (BENHAMOU; OLDER, 1997).

Algoritmos como o HC4 (GRANVILLIERS; BENHAMOU, 2001) implementam a HC eficientemente em uma restrição contínua simples através das chamadas operações de estreitamento (do inglês *narrowing operations*), as quais são funções que calculam novos limites para os intervalos correspondentes a cada variável do escopo da restrição, utilizando análise de intervalos (JAULIN *et al.*, 2001). Por exemplo, para a restrição $c: x + y = z$ com domínios para as variáveis de seu escopo $Ix = [a..b]$, $Iy = [c..d]$ e $Iz = [e..f]$, são calculadas as envoltórias $Hx = Ix \cap [e - b..f - a]$, $Hy = Iy \cap [e - d..f - c]$ e $Hz = Iz \cap [a + c..b + d]$. Como se faz necessário atingir o ponto fixo, cada operação é reaplicada iterativamente, com H no lugar de I, até que nenhuma envoltória calculada seja mais estreita que a correspondente calculada na iteração anterior (BENHAMOU, 1995).

Como apontado por Benhamou e Older (1997) e Lhomme (1993), a remoção de todos os valores inconsistentes de uma caixa é inviável, devido à natureza contínua dos domínios representados. Além disso, ao usar a envoltória para representar domínios aproximados, a HC deixa de armazenar um grande volume de informação sobre valores não suportados no domínio. Isso faz com que as consistências baseadas em intervalo sejam de ordem inferior às consistências obtidas em domínios discretos, agravando o problema da consistência local. Trabalhos como o de Cruz e Barahona (2003), que propõe uma generalização de ordem superior para a HC, bem como algoritmos para atingi-la, baseando-se na ideia de representar múltiplas restrições contínuas como uma única restrição global, buscam mitigar esse impacto. Neste trabalho, no entanto, será usada para domínios contínuos apenas a HC, como definida nesta seção.

2.2.3. Restrições globais e restrições tabulares

Os *solvers* de CSP costumam fornecer recursos para compor restrições complexas e específicas, quando demandado pelo problema modelado, a partir de outras mais primitivas[14]. No entanto, essa decomposição da restrição original tende a intensificar os problemas de consistência local, pois a consistência é aplicada individualmente sobre cada restrição decomposta, renunciando a uma visão global do escopo original. Restrições globais mitigam esse efeito, ao identificar cenários comuns a CSPs em que algoritmos especializados

[14] Uma das formas de decompor restrições complexas é através de *reified constraints*, que são restrições estendidas em uma variável booleana responsável por indicar se a restrição está ou não satisfeita. Essa técnica pode ser usada para construir fórmulas proposicionais sobre restrições, como a disjunção apresentada no segundo parágrafo desta seção (BELDICEANU *et al.*, 2013).

conseguem inferir com eficiência consistência global sobre o escopo (BELDICEANU; CONTEJEAN, 1994), ou escolhendo decomposições convenientes que garantem um nível de consistência local mais próximo de GAC (BESSIÈRE *et al*., 2009).

Uma restrição que apresenta estes problemas é a restrição de pertinência (do inglês *membership-constraint*), que é satisfeita se a (única) variável de seu escopo tiver em seu domínio ao menos um valor de uma lista L, a qual expressa os valores permitidos para a variável, e que equivale à disjunção $x = v_1 \vee x = v_2 \vee ... \vee x = v_n$ quando $L = [v_1, v_2, \dots, v_n]$ (STOLZENBURG, 1996). É intuitiva a conclusão que, quando avaliada nessa forma por um mecanismo de propagação, não ocorrerá filtro sobre $dom(x)$, pois desde que x tenha em seu domínio mais de um valor de L, as igualdades correspondentes pode ser ou não verdadeiras, o que confere GAC às disjunções sem que nenhuma filtragem ocorra (LHOMME, 2004).

Também frequente na literatura, e com definição que manifesta o problema da disjunção, é a restrição tabular, que é um tipo de restrição expressa na forma *extensional*. É um formato bastante utilizado em configuração de produtos, e outras aplicações onde o conhecimento provém de bancos de dados (LECOUTRE; SZYMANEK, 2006). Pela definição da forma *extensional*, uma restrição tabular é uma disjunção de suas tuplas, que por sua vez são conjunções de seus literais. Portanto, se decomposta em restrições primitivas apresenta problema similar ao da restrição de pertinência sob mesma condição. As restrições tabulares podem ainda ser representadas pelas tuplas não suportadas, quando então são chamadas de restrições tabulares negativas, sendo um complemento das restrições tabulares positivas (BESSIÈRE e RÉGIN, 1997). Neste trabalho, são consideradas apenas tabelas positivas cujas tuplas são todas suportes, de modo que os termos serão usados de forma intercambiável.

As restrições tabulares apresentam crescimento exponencial de sua complexidade espacial em função de sua aridade, o que tende a desacelerar o raciocínio sobre as mesmas (MAIRY; DEVILLE; LECOUTRE, 2015; BENNAI *et al*., 2023). Dada a importância desse tipo de restrição, a literatura tem se enriquecido de técnicas para filtragem eficiente. Os primeiros trabalhos dedicados a tabelas incluem os algoritmos GAC-Schema de Bessière e Régin (1997), voltados para restrições *intensional* e *extensional* de aridade *n*. Algoritmos anteriores costumavam ser escritos exclusivamente para restrições binárias. Lecoutre e Szymanek (2006) trazem como uma evolução dos GAC-Schema, algoritmos dedicados a explorar a estrutura interna da tabela (em especial o fato das tuplas estarem disponíveis explicitamente) para atingir GAC sobre uma restrição tabular com maior eficiência. Sobre essa linha de pesquisa, outros algoritmos de destaque foram desenvolvidos, como o STR (ULLMANN, 2007) e o STR2+ (LECOUTRE, 2011), implementando particionamento da lista de tuplas, e os estados da arte STRbit (WANG *et al*., 2016) e CT (DEMEULENAERE, 2015), usando representações esparsas compactas (*bitsets*) para manipular as estruturas internas.

Outra abordagem que tem sido usada para aumentar a eficiência da manipulação de tabelas são as representações compactas, como a *Smart Table Constraint* (STC), proposta por Mairy, Deville e Lecoutre (2015), entre outras identificadas em seu trabalho. Esta técnica promove uma redução nas complexidades espacial e temporal, por mesclar representações *intensional* e *extensional*, além de facilitar a administração das tabelas, por agrupar múltiplos suportes em um. A implementação dos autores o faz ao estender uma restrição tabular com operações aritméticas simples, chamadas de restrições de tupla (do inglês *tuple constraint* - TC). Neste trabalho, são usadas as operações relacionais $\{<, \leq, =, \neq, \geq, >\}$, e operações especiais de pertinência $\{\in, \notin\}$ e de universalidade $\{*\}$ em suas células. Estas duas últimas, em particular, são propostas anteriores de representação compacta de restrições tabulares. O $*$ vem da definição de suportes curtos (do inglês *short-supports*), e além de propiciar a representação compacta, permite que a avaliação dos suportes representados se dê simplesmente ignorando a variável que o recebe (NIGHTINGALE *et al.*, 2013). Já o $\in$ é baseado no conceito de tuplas

comprimidas (do inglês *compressed tuples*) que, tal qual o $*$, permite representar múltiplos valores de uma variável em uma mesma tupla, mas aceitando qualquer subconjunto do domínio (KATSIRELOS; WALSH, 2007).

Também usada neste trabalho, a restrição de índice de elemento (do inglês *element constraint* - *EC*) é outra restrição com algoritmo especializado proposto por Van Hentenryck e Carillon (1988). Seu objetivo é garantir que, sejam i uma variável inteira, e uma variável de valor, e L uma coleção ordenada finita de valores, e é igual ao i-ésimo elemento de L. Os autores descrevem a filtragem imposta por essa restrição aos domínios respectivos das variáveis envolvidas, como a remoção de valores inconsistentes de i quando e é atualizado, e vice-versa, o que é compatível com a definição de GAC vista anteriormente.

2.3. CONFIGURAÇÃO BASEADA EM CONHECIMENTO

A customização em massa propõe a oferta de produtos personalizados de acordo com a necessidade do cliente, mantendo a experiência do mesmo em termos de custos, prazos e processo de aquisição próximos aos de produtos padronizados (MOON; CHADEE; TIKOO, 2008). Felfernig *et al*. (2014) argumentam que o uso de tecnologias de configuração de produto, cujo conceito é sumarizado por Junker (2006) como "a tarefa de compor um sistema customizado a partir de componentes genéricos", traz diversos benefícios. Entre eles, pontuam: a) a redução nos erros, a melhoria nos resultados e o aumento no volume de vendas ao automatizar o processamento dos requisitos do usuário (do inglês *user requirement* - UR); b) a redução na confusão das muitas alternativas (tradução livre do inglês *mass confusion*) e na dificuldade de elencar requisitos complexos, aspectos que tendem a dificultar a escolha de soluções otimizadas, através de recursos que auxiliam e/ou guiam o usuário na tomada de decisão; e c) a integração e gestão eficiente do conhecimento sobre o produto ao processo de customização, ou configuração.

Diversas técnicas têm sido usadas para atender a tarefa de configuração, destacando-se as abordagens baseadas em regras, e as baseadas em restrições. A primeira é mais simples, e se baseia em encadeamento de regras condicionais *if/then*, proveniente de sistemas especialistas. No entanto, a direcionalidade e interdependências faziam com que, com o crescimento dos modelos, a manutenção se tornasse extremamente dispendiosa (JUNKER, 2006; FALKNER *et al*., 2016). A literatura tem mostrado que as abordagens baseadas em restrições se destacam cada vez mais, sanando estes problemas e trazendo novos desafios.

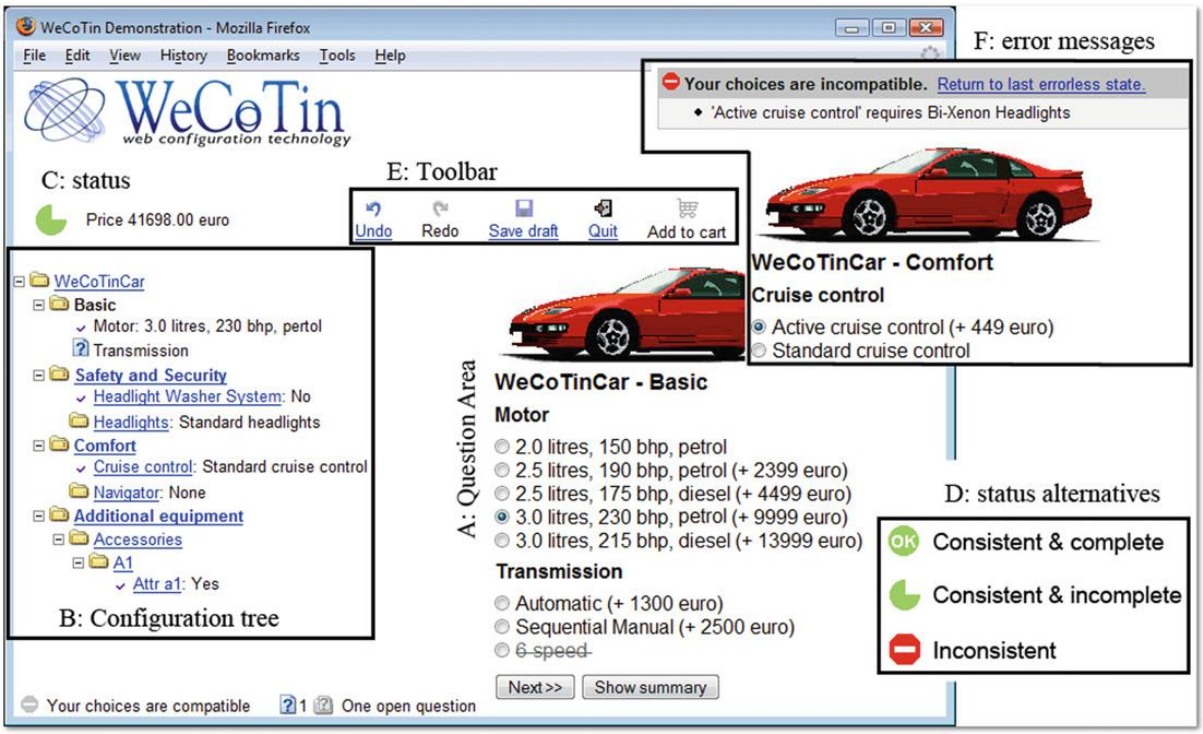
Falkner *et al*. (2016) reportam que os usuários normalmente não são capazes de fornecer todos os seus requisitos de antemão, antes preferindo serem guiados na tomada de decisão, vendo os efeitos de cada decisão sobre a configuração, e o que resta a ser decidido. Isso faz com que a tarefa de configuração seja, normalmente, um processo iterativo e interativo, onde o usuário interage com um software configurador de produto, esperando uma resposta rápida e concisa (FALKNER *et al*., 2020). Este tem acesso ao conhecimento na forma de catálogos de produtos em tabelas de banco de dados, que devem ser convertidas em restrições, junto às restrições funcionais sobre os componentes catalogados (JUNKER, 2006).

A filtragem de valores é uma forma de prover *feedback* ao usuário ao longo da tarefa de configuração. Idealmente, a consistência global seria desejada, mas como seu cálculo pode ter alta complexidade, recorre-se a consistências locais, como a GAC (JUNKER, 2006). O objetivo é que se remova o máximo possível de valores inválidos, e que os valores removidos possam ser marcados como inviáveis na interface do usuário a cada decisão tomada, de modo a reduzir a chance de que o mesmo faça escolhas que o levem a uma configuração inviável, ou ao menos que esteja ciente desta consequência (FALKNER *et al*., 2020). Estipulados estes propósitos, a tarefa de configuração interativa é formalizada como uma CN definida em termos da base de conhecimento sobre o produto a ser configurado, estendida com os requisitos de

usuário (cliente) como restrições, a serem submetidas para um configurador (JUNKER, 2006; HOTZ *et al*., 2014).

A Figura 3 apresenta um configurador de carros onde é possível observar alguns dos requisitos comuns a configuradores. Os campos de múltipla escolha anotados com "*A: Question Area*" indicam os valores de domínio para as variáveis *Motor* e *Transmission*. Há um UR instanciando *Motor* para o valor *3.0 litres, 230...*, enquanto o valor *6 speed* está tachado para *Transmission*. Isso possivelmente indica que este último valor é inconsistente com o requisito de usuário e que, portanto, a variável teve seu domínio reduzido para apenas os dois primeiros valores. Na mesma figura ainda é possível observar um destaque anotado com "*F: error messages*" indicando uma situação de contradição, com a mensagem "Suas escolhas são incompatíveis" (tradução nossa), e a lista de requisitos que se contradizem (LEITNER *et al*., 2014).

Figura 3 – WeCoTin: Ambiente de configuração (configurador de carro)



Fonte: Leitner *et al*. (2014, p. 101)

3. TRABALHOS CORRELATOS

A literatura é escassa em trabalhos voltados para manipulação de números reais em restrições tabulares, mas apresenta diversas publicações voltadas a propor melhorias para esse tipo de tabela, abordando aspectos como desempenho computacional, redução do consumo de espaço de armazenamento e facilidade de administração e representação. Os trabalhos aqui relacionados foram localizados a partir de uma busca realizada nas bases de publicações científicas ACM Digital Library®, IEEEXplore®, ScienceDirect®, Scopus®, Spring Link® e Web of Science®, limitada a trabalhos publicados entre 2014 e 2023. A *string* de busca utilizada foi (("*table constraint*" OR "*tabular constraint*" OR "*table constraints*" OR "*tabular constraints*") AND ("*constraint programming*" OR "*constraint satisfaction problem*")), cuja busca retornou um total de 244 resultados, distribuídos nas bases como consta na Tabela 2.

Uma análise de relevância sobre os títulos das obras para com este trabalho reduziu os resultados para 73, que após remoção de duplicados resultou em 43 trabalhos. Sobre estes foi

realizada uma classificação em notas de 0 a 10 pela leitura de seus resumos, seguida por leitura aprofundada dos artigos considerando a ordem decrescente da nota. Dessa leitura, 10 artigos foram selecionados por sua relevância para este texto, e um breve resumo dos mesmos é apresentado nos próximos parágrafos.

Tabela 2 – Distribuição das publicações encontradas na revisão da literatura

Base consultada	**Publicações encontradas**
ACM Digital Library®	18
IEEEXplore®	4
ScienceDirect®	3
Scopus®	53
Spring Link®	140
Web of Science®	26
Total	244

Fonte: elaborado pelo autor (2024)

Buscando aprimorar a representação de restrições tabulares, Mairy, Deville e Lecoutre (2015) propõem a *Smart Table Constraint*, cuja definição estende o conceito de restrição tabular ao formalizar uma generalização dos conceitos de suportes curtos e tuplas comprimidas. Além das operações aritméticas simples já descritas na fundamentação teórica, a definição proposta determina o formato das expressões usadas nas tuplas, de modo a formar uma CN acíclica interna à tabela, que aumenta a eficiência da busca por tuplas válidas e viabiliza seu uso na propagação. Os autores também sugerem o algoritmo *smartSTR*: um propagador baseado em STR que usa decomposição em árvore sobre a CN para identificar tuplas válidas. Os resultados para os testes de desempenho mostraram competitividade com propagadores dedicados a cenários específicos, com pior cenário dobrando o tempo de busca, enquanto que contra o propagador de suportes curtos (genérico), os cenários mostraram piora de 20 a 30 vezes para a referência, contra apenas duas vezes no *smartSTR*.

Younsi *et al*. (2023) apresentam uma melhoria para métodos de busca sobre decomposições do hipergrafo de restrições em hiper árvores. Tais abordagens têm melhor complexidade teórica em relação a métodos clássicos. No entanto, apresentam custo fixo significativo, pois dependem de representar as restrições em forma de relações (forma tabular) e fazer operações de junção entre relações contidas em um mesmo hiper nodo da árvore. Como as relações têm complexidade exponencial, o custo de memória é o que mais se destaca em cenários reais. Além disso, as operações de junção provocam explosão de memória. Esses problemas se destacaram especialmente no cenário de teste baseado em configuração de produto usado no trabalho, onde várias das variações do cenário levaram a estouro de memória. Nas variações que concluíram, o melhor tempo de busca para a primeira solução foi de aproximadamente 16 vezes maior que o algoritmo CT, por conta da grande quantidade de restrições com escopo disjunto. Para os demais cenários avaliados, o método se mostrou competitivo, em especial nos modelos menores. Em alguns destes, foi possível resolver a tarefa de satisfação em até 3% do tempo usado pelo CT.

Também propondo um aprimoramento de representação, Audemard, Lecoutre e Maamar (2020) definem as tabelas segmentadas, também estendendo o conceito de tuplas comprimidas e suportes curtos, bem como outras representações compactas de tabela. Os autores definem tuplas segmentadas como compostas por segmentos, cada qual envolvendo uma ou mais variáveis do escopo da tabela. Cada segmento pode ser um valor, o operador asterisco (em ambos os casos para apenas uma variável), ou uma tabela aninhada. Dessa forma, demonstram ser possível representar problemas como o da otimização de palavras cruzadas (CD), de forma mais compacta que em outras representações. A avaliação experimental mostrou ganho de desempenho significativo comparada a abordagens clássicas em versões

maiores do problema CD, continuando a obter resultados mesmo em tamanhos para os quais as outras excederam limites de memória ou tempo impostos ao teste. No entanto, argumenta-se que não é trivial a compressão automatizada eficiente de tabelas para segmentos.

Bennai *et al*. (2023) também trazem uma proposta de compressão, mas com ênfase em uma heurística para compressão automatizada e eficiente da tabela. Sua abordagem recorre a métodos de análise de bancos de dados transacionais para identificar padrões (*frequent itemsets*) em uma restrição tabular, de modo a comprimir múltiplas tuplas a um par composto pelo *itemset* que compartilham, e uma tabela aninhada com escopo formado pelas variáveis não envolvidas no mesmo. Além disso, propõe o uso da área do *itemset*, definida como o produto entre a frequência com que aparece na tabela pelo seu tamanho, como o parâmetro a maximizar na mineração dos padrões. Os resultados experimentais foram obtidos aplicando GAC à tabela comprimida com algoritmos baseados em STR já conhecidos. O uso da heurística indicou redução de até cinco vezes no tempo de execução sobre o algoritmo para tabelas comprimidas aplicado diretamente, e a combinação chegou a ser duas vezes mais rápida que o STR2 (sem compressão) na busca. Quanto aos demais algoritmos comparados, apenas os estados da arte CT e STRbit mostraram resultados melhores.

O trabalho de Paparrizou e Stergiou (2015) apresenta algoritmos baseados em consistência de ordem superior, especificamente maxRPWC, visando aprimorar implementações anteriores baseadas nesse tipo de consistência. São algoritmos especializados em restrições tabulares, contrastando com seus antecessores agnósticos. Os autores demonstram que a alteração no custo computacional na busca, quando comparados a outros algoritmos de filtro da literatura, varia de acordo com características das restrições presentes nos cenários. Em dois dos cenários, o melhor dos algoritmos propostos para cada um ficou duas e três ordens de grandeza mais lento, enquanto no cenário de maior vantagem, o melhor dos algoritmos propostos ficou 2,5 vezes mais rápido que o melhor usado de referência. O desempenho depende fortemente da presença de interseções não-triviais entre as tabelas, e tende a se degradar com o crescimento dos domínios. O destaque, porém, fica por conta da maior filtragem promovida pela consistência de ordem superior quando comparado a algoritmos baseados em GAC, bem como pelo menor uso de memória quando comparado com adaptações anteriores do algoritmo STR para consistência de ordem superior.

Já Schneider e Choueiry (2018) desenvolvem seus algoritmos baseados no algoritmo CT, sobre o qual conseguem explorar propriedades provenientes de uma abordagem híbrida entre GAC e PWC. Discorrem sobre: a) identificar quando GAC equivale a PWC, evitando aplicar o custo fixo do segundo; b) avaliar a representação em grafo da CN de modo a reduzir o número de escopos considerados pelo PWC; c) identificar as variáveis relevantes para o PWC a fim de reduzir o volume de dados armazenados; e d) filtrar blocos de suportes com base em propriedades que os relacionam ao analisar a PWC. Os testes concluíram que o algoritmo performa melhor que outras abordagens de PWC na busca, chegando a ⅓ do tempo dos demais algoritmos do tipo. Apesar disso, ainda apresenta desempenho inferior ao do algoritmo de referência com tempos entre 10% e 100% maiores. No entanto, a análise de nodos percorridos na busca, prova a filtragem superior da fPWC. A contagem de nodos ficou próxima da metade dos algoritmos de GAC, e entre 1% e 2% inferior aos demais algoritmos de PWC. Outro ponto de destaque foi o uso de memória, que diferente de seus pares, ficou tão baixo quanto nos GAC.

Li *et al*. (2019) abordam o problema da complexidade das restrições tabulares a partir de uma perspectiva mais ampla sobre a propagação, adaptando não apenas o algoritmo de filtro, como também o mecanismo de propagação, para que em conjunto, permitam processamento paralelo de tabelas. Definem GAC temporária (do inglês *Temporary GAC* - TGAC) como uma GAC sobre cópias dos estados das variáveis, cópia essa que é usada pelos algoritmos de filtro para a filtragem do domínio, em vez de filtrar diretamente as variáveis originais. Também provam que seu ajuste do mecanismo de propagação para execução paralela dos algoritmos de

filtro garante a convergência de TGAC para GAC sob o custo de chamar os filtros mais vezes que seu equivalente serial. Os autores provam que os algoritmos baseados em STR e CT disponíveis na literatura seguem um *framework* em comum, e propõem para seus algoritmos de filtro um novo *framework* que o estende, facilitando a aplicação em algoritmos existentes. A avaliação experimental foi efetuada nos algoritmos estado da arte CT e STRbit, sendo possível observar que a abordagem se destaca em modelos com mais restrições, tabelas volumosas e/ou domínios amplos, chegando a propagar até 8 vezes mais rápido que suas versões seriais. Nos casos que não se enquadram nessas características, a propagação serial é até 3 vezes mais rápida.

Tendo em vista a forma como GAC se destaca no desempenho da busca comparado a consistências de ordem superior, Likitvivatanavong, Xia e Yap (2014) propõem um trabalho sobre como converter uma CN em outra equivalente, rearranjada em relação à original de tal forma que, quando imposta GAC, equivale a impor fPWC sobre a original. Os autores demonstram que é possível obter essa CN equivalente ao criar novas variáveis para representar as combinações de variáveis que são comuns ao relacionamento entre duas (PWC) ou mais (*k*-WC) restrições tabulares. Novas colunas, correspondentes às variáveis combinadas, são incluídas nas respectivas tabelas. O domínio de cada uma dessas novas variáveis é inferido como a união das combinações permitidas pelas tabelas originais para o respectivo conjunto de variáveis originais, cuja combinação é representada pela nova variável. Experimentalmente, a abordagem mostrou maior eficiência quando comparada a algoritmos de fPWC que a precederam, bem como se mostrou capaz de aprimorar seu desempenho quando usados em conjunto. Ao aplicar a abordagem sobre algoritmos de GAC, como os da família STR, alguns cenários apresentaram redução de até 50% do tempo da busca, mas em outros marcou resultado inferior, justificado pela filtragem mais efetiva.

Os trabalhos avaliados, apesar de em sua maioria terem como objetivo e como métrica principal a otimização do custo computacional na busca de soluções, colaboram com a configuração baseada em conhecimento através das melhorias na filtragem. Isto ocorre tanto em custo computacional, relevante para a experiência do usuário do configurador, quanto na efetividade da filtragem através de consistências de ordem superior, que agilizam o processo de completar a configuração ao eliminar mais valores inválidos a cada novo requisito do usuário em domínios finitos. Este trabalho busca estender essas colaborações para domínios reais, que embora divirjam da natureza discreta das restrições tabulares, podem vir a ser convenientemente tratados nesse tipo de restrição em cenários específicos de configuração.

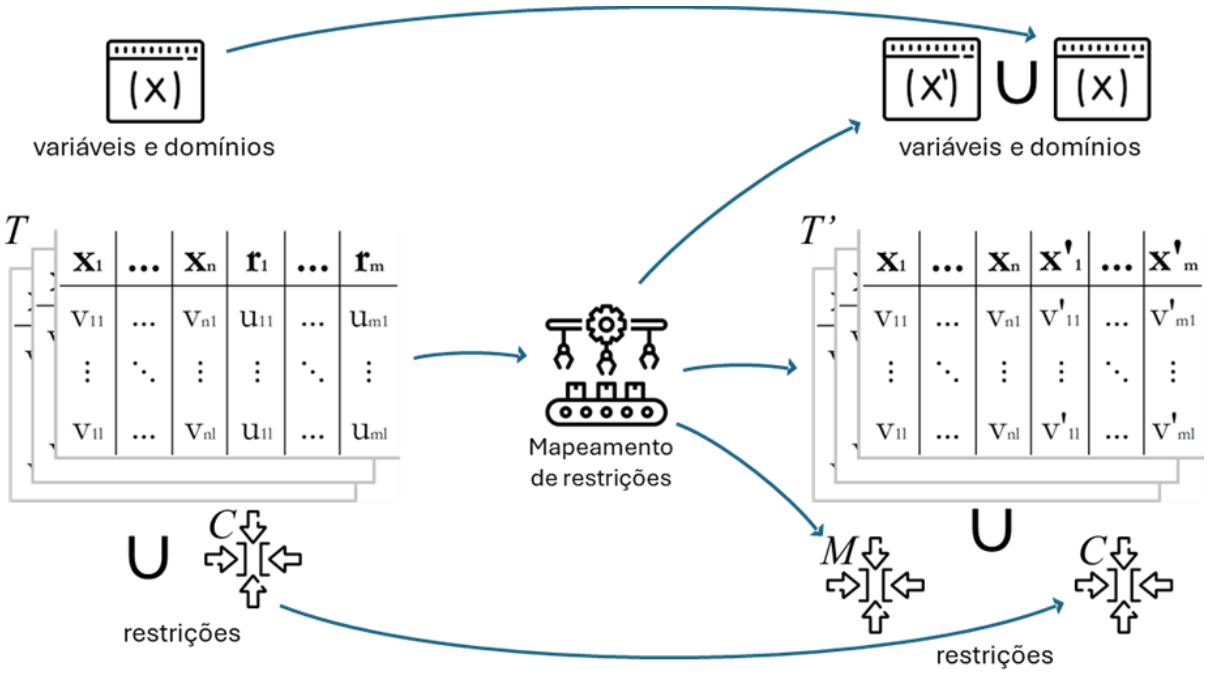
4. *FRAMEWORK* PROPOSTO

O *framework* proposto por este trabalho objetiva permitir a modelagem de CSPs com tabelas capazes de receber colunas reais por meio da adaptação de restrições tabulares. Além disso, três variações de implementação compatíveis com o *framework* são apresentadas. A primeira implementação é mais simples, construída sobre uma tabela adaptada e uma restrição de mapeamento composta por restrições primitivas de lógica e aritmética, abrindo espaço para uma consistência local mais fraca. Já a segunda propõe a combinação de uma restrição tabular com uma restrição de índice de elemento. Esta implementação se aplica apenas a colunas reais que não usam operações TC especiais, ou a tabelas de representação convencional, não STC. Por fim, a terceira implementação visa unir a generalização da primeira com a consistência de ordem superior da segunda, ao propor um algoritmo de filtro que estende a definição da EC para suportar os operadores definidos pela representação compacta das STCs. Além disso, por ser orientado a colunas, o *framework* permite que na mesma tabela coexistam variáveis de domínios contínuo ou finito, e cada coluna pode ser tratada com a implementação mais adequada.

Em todas as três implementações, um mesmo esquema é seguido. Seja um CSP $P = (X, D, C \cup T)$ que tem em sua composição o conjunto de tabelas T como restrições cujos escopos contêm variáveis reais. De P deriva um CSP $P' = (X \cup X', D \cup D', C \cup T' \cup M)$ equivalente, onde a equivalência ocorre substituindo: a) o conjunto X de P pela sua união com um conjunto X' de variáveis de rótulo; b) D pela sua união com um conjunto D' dos rótulos que compõe os domínios de X'; e c) T por um conjunto T' de restrições tabulares adaptadas que substitui T de forma bijetora, em união com um conjunto M de restrições, que junto com T' representam sem perdas as relações de T.

Os conjuntos X' e D' são gerados de acordo com T, sendo que para cada coluna real em T, uma nova variável de rótulo é adicionada a X'. Simultaneamente, adicionam-se a D' os domínios correspondentes a cada variável de rótulo adicionada, onde o domínio de cada uma é definido com base nos literais que formam a coluna correspondente em T. As tabelas de T' são construídas copiando-se as tabelas de T e substituindo-se as colunas reais por colunas de rótulo, enquanto que as restrições de M são restrições desenvolvidas de modo a gerar um mapeamento entre cada variável real da tabela original com a variável de rótulo correspondente na tabela adaptada. O *framework* descrito até aqui é ilustrado na Figura 4 para uma tabela, e cada implementação descrita nas subseções a seguir propõe uma forma de gerar as restrições de M.

Figura 4 – *Framework* de conversão de CN com tabelas contendo colunas reais. As tabelas de P (à esquerda) são substituídas por novas tabelas em P' (à direita), com novas variáveis de rótulo (x'_i) no lugar das variáveis reais (r_i). As novas variáveis são adicionadas às variáveis já presentes em P, bem como as restrições adicionais necessárias



Fonte: elaborado pelo autor (2024)

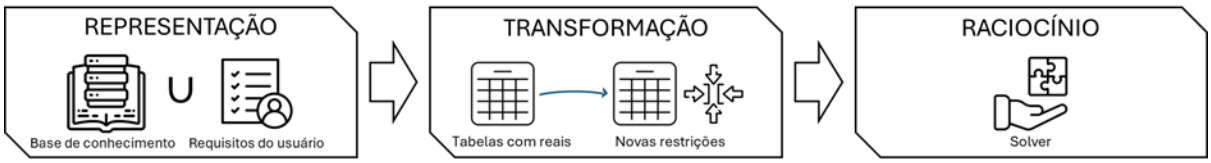
O *framework* inclui uma etapa adicional à tarefa de configuração, fazendo com que o fluxo de execução ocorra como na Figura 5, onde aparecem as seguintes etapas:

- REPRESENTAÇÃO: a tarefa de configuração é construída, fornecendo-se as variáveis, domínios correspondentes e restrições da base de conhecimento, bem como os requisitos do usuário;
- TRANSFORMAÇÃO: o passo adicionado à tarefa de configuração, onde as tabelas da base de conhecimento que atuam sobre variáveis reais são convertidas para

tabelas e restrições suportadas pelo *solver*. O esquema da Figura 4 é executado nesta etapa;

- RACIOCÍNIO: a etapa onde o *solver* irá executar o raciocínio automatizado sobre o conhecimento transformado, fazendo inferências sobre os domínios.

Figura 5 – Tarefa de configuração modificada



Fonte: elaborado pelo autor (2024)

A seguir são descritas as três implementações propostas para a transformação das tabelas e criação dos novos domínios e restrições.

4.1. MAPEAMENTO VIA COMPOSIÇÃO DE RESTRIÇÕES SIMPLES

Considerando uma tabela não compactada, esta implementação propõe substituir os literais reais que a compõem por literais de rótulo que identifiquem o suporte, e por predicados sobre estes literais. Para cada literal substituído, um predicado é adicionado sob forma de uma restrição de implicação. Esta tem uma restrição de igualdade da variável com o rótulo do suporte como antecedente, e uma restrição de igualdade da variável real com o valor do literal original.

Como um suporte é uma combinação permitida, tem-se que quando o filtro do propagador da tabela filtra uma variável de rótulo para um único valor, o antecedente da restrição de implicação gerada para aquele suporte se torna verdadeiro, de modo que para a implicação ser satisfeita, o consequente precisa ser verdadeiro também. Isso faz com que a restrição de igualdade que constitui esse consequente tente filtrar a variável real correspondente para que seja instanciada com o valor da tabela original correspondente ao rótulo.

Em contrapartida, quando a variável real é filtrada para que não tenha em seu domínio um dado valor, originalmente presente na tabela, o consequente da implicação se torna falso. Por definição, para que a restrição de implicação seja satisfeita, a sua contrapositiva deve ser também. Logo, o antecedente precisará ser falso, e o filtro da restrição de igualdade que o constitui irá tentar remover o rótulo correspondente (o que substituiu o valor real filtrado na tabela original) do domínio da variável de rótulo. Com essa informação, o propagador da tabela será capaz de identificar o suporte correspondente como inválido.

Observando a propagação no sentido da tabela para as restrições de implicação, observa-se que, enquanto a variável de rótulo não é instanciada, os antecedentes das implicações adjacentes podem ou não ser verdadeiros e, portanto, nenhuma filtragem ocorre na variável real substituída até que apenas um suporte seja válido. Essa falta de conhecimento global sobre o estado da tabela como um todo faz com que a consistência alcançada por esta implementação seja inferior à GAC.

Apesar da ordem de consistência inferior, diferente das implementações a seguir, esta pode ser feita usando apenas restrição tabular e restrições primitivas, sem recorrer a restrições mais elaboradas, como nas implementações apresentadas a seguir. Além disso, quando a tabela original é compactada com TCs, basta substituir o consequente de cada implicação por uma restrição primitiva que implemente o operador correspondente.

4.2. MAPEAMENTO VIA RESTRIÇÃO DE ÍNDICE DO ELEMENTO - EC

A EC permite mapear um conjunto de identificadores em um índice para um conjunto de valores. Dada a simplicidade de sua definição, é possível usar para consumir tabelas com colunas reais, com a vantagem da visão global que uma restrição global oferece.

Nessa implementação, cada coluna da tabela é convertida para uma EC. O domínio da variável de rótulo criada para substituir a variável real é criado com números inteiros em sequência, de 1 a *n* (inclusive), onde *n* é a quantidade de valores reais distintos. A EC então é criada de modo a mapear cada um desses rótulos a um valor real distinto do conjunto de literais.

Dessa forma, os propagadores da EC garantem HC ao filtrar qualquer rótulo cujo valor real correspondente tenha sido filtrado e ao preservar apenas o intervalo real cujos limites correspondem ao menor e ao maior dos rótulos suportados. Este comportamento atenua o problema de filtragem abaixo do ideal, proveniente da composição de restrições primitivas usada pela implementação 4.1. De fato, usando-se variáveis de domínio finito no lugar da variável real, a restrição resultante da combinação de uma restrição tabular com múltiplas EC pelo método proposto é GAC (prova descrita abaixo), que é a consistência mais forte atingível ao analisar restrições individualmente. No entanto, este método fica limitado a tabelas não compactadas, ou tabelas cujas colunas reais tenham apenas operações de igualdade, visto que não permite mapear rótulos para predicados arbitrários.

O raciocínio da prova de que este método é GAC quando as variáveis substituídas têm domínio finito é o que segue. Se o valor de uma variável substituída é consistente, então o rótulo que indica a posição desse valor no conjunto ordenado responsável pelo mapeamento é consistente. Por sua vez, se um rótulo é consistente, então a restrição tabular tem suporte para o mesmo. Pela lei da transitividade, se conclui que, se um valor da variável substituída é consistente, então a tabela possui suporte para o mesmo.

4.3. MAPEAMENTO VIA RESTRIÇÃO ESPECIFICADA COM BASE NA EC

Esta implementação visa unir a flexibilidade de representação propiciada pela primeira implementação com a consistência GAC oferecida pela segunda, a fim de oferecer um *feedback* mais assertivo ao usuário do configurador que a implementa.

Para isso, é proposta uma restrição inspirada na restrição de índice de elemento, que diferente desta, mapeia valores inteiros para conjuntos de valores reais, e não para valores únicos. Além disso, estes intervalos podem ter interseções não vazias. Tais intervalos são representados usando a mesma notação apresentada pelas STCs, expressando os conjuntos de valores reais através de TCs. Este trabalho assume a restrição proposta como original, pois não foi encontrada definição similar na literatura.

Para associar essa restrição a uma restrição tabular convertida, assim como estabelecido no *framework* e respeitado nas demais implementações, as TCs reais são substituídas por literais de rótulo, e a nova restrição é criada de modo a mapear cada rótulo à TC substituída. Devido ao sombreamento dos intervalos, o mapeamento não pode ser uma função bijetora, como especificado para a EC. Antes, representa uma coleção de expressões condicionais, como ocorre na implementação 4.1.

Formalmente, $\forall (l, R) \in M, x_l = l \Rightarrow x_r \in R$, onde a função de mapeamento M é um conjunto de pares (l, R), l é um rótulo e R é um conjunto de valores reais, x_l é a variável de rótulos e x_r é a variável real que está sendo mapeada. É possível aproximar esta definição da definição de EC, fazendo com que os valores de l em M sejam subsequentes. Dessa forma, M equivale a uma tupla O de conjuntos de valores reais, tal que a restrição aqui proposta visa garantir que x_r está contida no l-ésimo conjunto de O quando $x_l = l$.

Apesar de haver equivalência com o conjunto de restrições de implicação usado na primeira implementação proposta neste texto, a restrição aqui apresentada abre caminho para

o desenvolvimento de um propagador próprio, capaz de uma visão global sobre o mapeamento, a partir da qual pode atingir HC. Este trabalho também propõe o algoritmo de filtro para este propagador, como representado no Algoritmo 1.

Algoritmo 1 – Algoritmo de filtro inspirado na EC, para associar inteiros a intervalos

```
1  M <- {(l1, R1), …, (ln, Rn)}
2  função filtrar()
3      M′ <- M
4      para cada (l, R) em M′
5          se l ∉ dom(xl)
6              M′ <- M′ - {(l, R)}
7          senão,
8          se R ∩ dom(xr) = {}
9              M′ <- M′ - {(l, R)}
10             dom(xl) <- dom(xl) - {l}
11     H <- {}
12     para cada (l, R) em M′
13         H <- H ∪ R
14     dom(xr) <- dom(xr) ∩ H
15     se dom(xl) = {} ∨ dom(xr) = {}
16         retorna contradição;
```

Fonte: elaborado pelo autor (2024)

O algoritmo assume um mapa `M`, onde cada entrada mapeia um rótulo `l` para um intervalo real `R`. Quando usado em restrições tabulares, estes intervalos podem ser inferidos a partir de uma TC e o domínio da variável de valor x_r. Para o escopo deste trabalho, apenas a abstração é relevante, ficando a critério da implementação decidir entre avaliação preguiçosa ou ansiosa do intervalo. Espera-se que `M` seja uma função definida em todo o domínio da variável de rótulo x_l. Ou seja, para cada valor de x_l existe uma única entrada em `M`.

A função `filtrar` implementa o filtro do propagador, que atua sobre x_l e x_r com base em uma análise de `M`. Primeiramente, é efetuada uma cópia `M′` de `M`, permitindo a remoção de entradas inválidas do mapa. Em seguida, cada entrada de `M′` é avaliada, a fim de verificar se a mesma constitui suporte a valores das variáveis do escopo, e removida de `M′` caso não o faça. Uma entrada `i` não será suporte se o rótulo `li` não estiver no domínio de x_l, ou se o conjunto `Ri` for disjunto ao domínio de x_r. Como os rótulos têm associação biunívoca para com as entradas, ou seja, cada rótulo tem suportes apenas na entrada a que está associado, a remoção de uma entrada é condição suficiente para remover o rótulo correspondente do domínio de x_l. Por fim, o filtro constrói a envoltória `H` contendo todos os valores suportados, através da união dos intervalos de cada entrada que não foi removida de `M′` na etapa anterior e filtra x_r ao definir seu domínio como a intersecção entre seu domínio antes da filtragem e `H`. Após filtradas, ambas as variáveis do escopo da restrição, se o algoritmo identificar que ao menos uma delas ficou com domínio vazio, retornará uma contradição indicando ao mecanismo de propagação do *solver* que ao menos uma das variáveis não possui valores suportados pela restrição em seu domínio. Caso contrário, a função encerra silenciosamente.

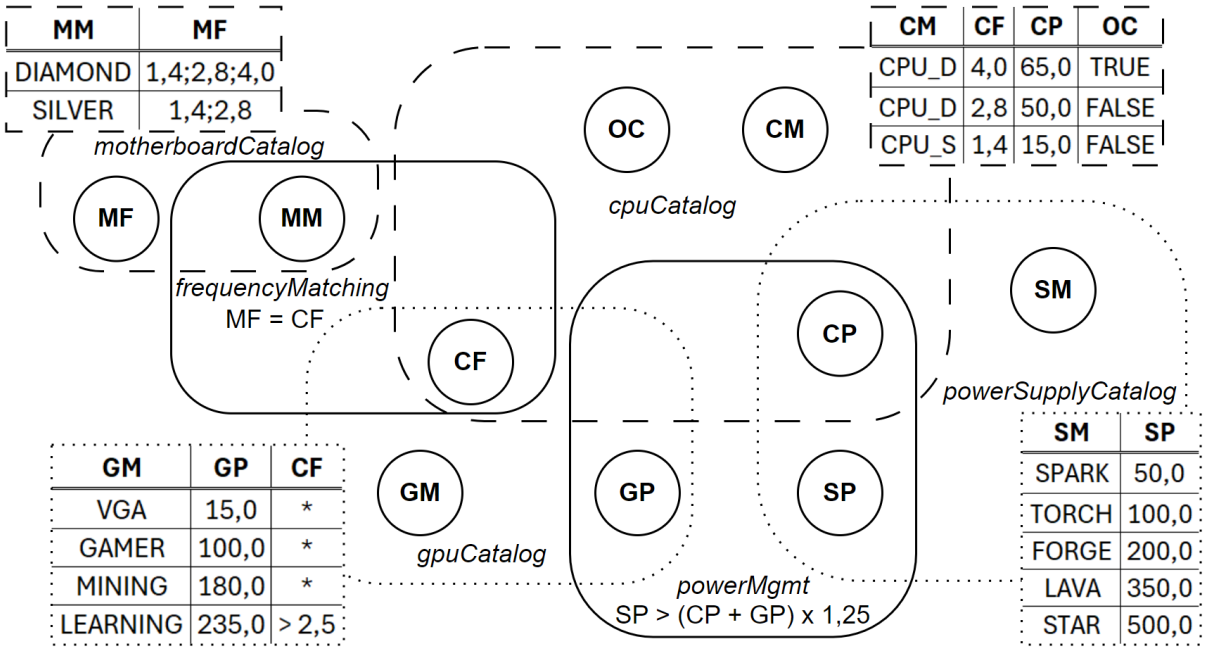
5. RESULTADOS EXPERIMENTAIS E DISCUSSÕES

5.1. DESCRIÇÃO DOS CENÁRIOS DE ESTUDO

A avaliação das implementações propostas para o *framework* foi realizada por dois caminhos distintos. Em um deles foi priorizada a análise da efetividade dos filtros, em termos da capacidade de filtragem de valores inviáveis, o que determina o nível de consistência alcançado e qual implementação é mais adequada para uso em configuradores de produto. No outro, o desempenho de busca foi aferido para cada implementação, a fim de identificar a que melhor atende tarefas como satisfação e otimização.

Para a análise de efetividade dos filtros, foi adaptado de Hotz *et al*. (2014) um modelo para configuração de um computador pessoal (PC). A CN adaptada é mostrada no hipergrafo da Figura 6. As variáveis CF e MF representam frequências de trabalho para a CPU e para a placa mãe, e têm domínio contínuo 1,0..4,0. As variáveis CP, GP e SP são as potências máximas da CPU e da GPU, e a potência nominal da fonte de alimentação, respectivamente, todas com domínio contínuo de 10,0..500,0. Para a análise dos resultados, foi atribuída precisão de 10^{-3} a essas cinco variáveis contínuas. As variáveis CM, MM, GM e SM representam os modelos de CPU, placa mãe, GPU e fonte de alimentação, enquanto que a variável OC indica se a CPU configurada terá *overclock*. Essas cinco variáveis são categóricas, com os respectivos domínios compostos pelos conjuntos de valores suportados nas mesmas pelas tabelas do modelo. Para análise dos resultados, estes domínios foram enumerados (por exemplo, {*CPU_D*=0; *CPU_S*=1}).

Figura 6 – Hipergrafo representando a CN para a configuração de um PC



Fonte: elaborado pelo autor (2024)

A figura mostra ainda que a CN é formada por duas restrições *intensional* e quatro restrições tabulares, que aparecem como hiper arestas rotuladas com as expressões e as relações subjacentes, respectivamente. Também aparece como rótulo um identificador (em itálico) para cada uma. Uma variante deste CSP foi também usada para avaliar a implementação 4.2 do *framework*, visto que esta não suporta as generalizações de STC para reais. Nesta variante, as tabelas *gpuCatalog* e *motherboardCatalog* foram substituídas por versões não compactadas delas mesmas. Para a tabela *gpuCatalog*, como o operador "$*$" indica qualquer dos valores do

domínio, e trata-se de um domínio contínuo, foram usados apenas os valores de CM suportados por outras restrições (*i.e.* 1,4; 2,8 e 4,0).

Por fim, para a aferição do desempenho na busca de soluções, foram coletados cenários de *benchmark* da competição de *solvers* CSP organizada pelos desenvolvedores do XCSP[15]. Esta competição também foi usada como referência de *benchmark* para trabalhos como Lecoutre e Szymanek (2006), Yap, Xia e Wang (2020) e Younsi *et al*. (2023).

O portal disponibilizou um pequeno *script* para descarregar todos os cenários disponíveis. Na preparação, foram removidos os *benchmarks* de competições anteriores, ficando para avaliação apenas os cenários apresentados com ênfase pelo portal. Estes totalizaram 84 problemas, cada qual com múltiplas variantes, diferindo em aspectos como tamanho e tipo de restrições utilizadas. Dessas variantes, foram identificadas 6260 com restrições tabulares positivas, em 34 dos 84 problemas.

Dessas 6260 variantes, foram escolhidas dez de maneira parcialmente arbitrária, com o cuidado de evitar problemas formados por unicamente tabelas binárias e variáveis booleanas a fim de obter cenários mais diversos. Problemas com restrições globais também foram evitados, pois como os problemas tiveram suas variáveis de domínio finito convertidas para domínio contínuo, o suporte a tais restrições precisaria ser revisto, o que foge ao escopo deste trabalho. Além disso, foram ignorados CSPs com tabelas compactadas, a fim de facilitar a elaboração dos cenários (como descrito na próxima seção). A seleção culminou em uma lista de 10 variantes de quatro problemas distintos, as quais aparecem listadas na seção 5.3, no Quadro 4. Dessas dez variantes, três não possuem solução válida, enquanto as demais possuem ao menos uma.

5.2. MATERIAIS E MÉTODOS UTILIZADOS

Visto que predominam variáveis de domínio finito nos escopos de tabelas, os problemas coletados da XCSP passaram por um pré-processamento para converter as variáveis de domínio inteiro em variáveis reais de intervalo único, que são suportadas pelo Choco-Solver. Os domínios das variáveis reais foram definidos como o intervalo entre, e contendo, os valores reais correspondentes aos inteiros mínimo e máximo presentes no domínio da variável original. Tratamento similar foi efetuado nas tabelas, onde os valores inteiros suportados foram convertidos para os valores reais correspondentes. Combinando estes problemas com cada uma das três implementações propostas para o *framework*, três abordagens são formadas.

Adicionalmente, os cenários originais foram preservados para elaborar casos de controle, onde o mecanismo de STC nativo do *solver* (exclusivo para domínios finitos) foi utilizado. Visto que o pré-processamento descrito acima foi usado apenas para a representação de dados, sem afetar a semântica dos CSPs envolvidos, as soluções encontradas pelas quatro abordagens devem ser compatíveis. Ou seja, ao estender o CSP de domínio finito do controle com novas restrições unárias de igualdade, cada qual sendo um literal de uma solução para uma das três abordagens, o CSP gerado deve ser consistente.

Também foi usada uma transformação das CNs baseada no algoritmo proposto por Katsirelos e Walsh (2007) para comprimir restrições tabulares. A ele foram incluídas algumas alterações para escolher o resultado com o menor número de suportes entre os gerados por algumas das heurísticas propostas pelos autores, e para substituir o operador de pertinência, usado pelo algoritmo para a compressão, por operadores de TC. Foram usados os operadores "$*$" (qualquer valor do domínio), "$\geq$" (qualquer valor maior ou igual que) e "$\leq$" (simétrico a "$\geq$") quando aplicáveis, além do próprio "$\in$" (qualquer valor pertencente ao conjunto finito dado) gerado no algoritmo de compressão. Um exemplo é mostrado no Quadro 2. A combinação desta conversão com as implementações 4.1 e 4.3 compõem as outras duas

[15] Acesso aos cenários da XCSP em https://xcsp.org/instances

abordagens avaliadas. A implementação 4.2 não pode ser usada com esta conversão, pois a EC é definida sobre pares de literais, que não permitem representar as tabelas compactadas.

Quadro 2 – Exemplo de STC gerada para teste. Esta foi gerada da variante de problema `Renault-medium-pos` e possuía originalmente 5 colunas e 63 suportes. Foi transformada em 4 suportes compactos, com 6 "$*$", 3 "$\in$" e 1 "$\geq$".

x1[0]	x4[5]	x4[7]	x5[5]	x3[3]
$*$	1	0	$\in \{0;99\}$	$*$
$\in \{0;2;3\}$	0	0	99	$*$
≥ 2	0	0	0	$*$
$*$	0	1	$\in \{0;99\}$	$*$

Fonte: elaborado pelo autor (2024)

Na implementação 4.2 foi utilizada, para variável de rótulos, a implementação de variável inteira de intervalo, em vez de uma variável com domínio enumerado. Visto que no Choco-Solver as variáveis reais são de intervalo único, ambas as implementações de variável inteira permitem o mesmo nível de filtragem nas variáveis envolvidas com a restrição, quando aplicadas à variável de rótulo. No entanto, usar variáveis de intervalo reduz o custo computacional de armazenar e processar valores intermediários de seus domínios.

Por fim, para a avaliação dos cenários foi utilizado o Choco-Solver, que é uma biblioteca Java® *open-source* cujo *solver* oferece suporte para restrições tabulares, CSP contínuo e uma API para desenvolvimento de propagadores personalizados, além de maturidade nos recursos para CNs de domínio finito (PRUD'HOMME; FAGES; LORCA, 2016; PRUD'HOMME; FAGES, 2022). As implementações foram codificadas também em Java, assim como os cenários de teste. Estes foram executados em um PC com processador Core i7-11700 com 8 GB de memória RAM disponíveis, e máquina virtual Java OpenJDK Temurin-17.0.11+9 de 64 bits.

5.3. APRESENTAÇÃO DOS RESULTADOS

Os testes de efetividade dos filtros foram realizados adicionando requisitos de usuário às tarefas de configuração construídas sobre o modelo da Figura 6. Tais URs foram escolhidas de modo a cobrir cenários diversos, para os quais espera-se que o *solver* responda com diferentes comportamentos, a fim de permitir a comparação entre as implementações propostas na seção 4.

O Quadro 3 mostra cada um dos cenários distribuídos em suas linhas, e enumerados na coluna "#". Mostra também o resultado do raciocínio do *solver* ao confrontar cada um com cada uma das implementações do *framework* (colunas com títulos de 4.1 a 4.3, indicando a implementação correspondente), bem como as URs que os compõem (coluna URs). Para os resultados do *solver*, "OK" significa que nada foi mudado além do próprio domínio correspondente às URs. "Contradição" indica que o *solver* foi capaz de identificar uma contradição envolvendo as URs e as restrições do modelo. Os cenários que não se aplicam à implementação 4.2 aparecem com "NA" na coluna correspondente. Já os demais resultados representam como o *solver* filtrou o domínio de algumas variáveis de interesse para a análise.

Os primeiros três cenários dispostos na ordem em que aparecem poderiam representar a evolução de uma configuração, com o usuário incluindo requisitos no configurador um a um, e o mesmo fazendo a consistência no *solver*. As saídas mostram que a implementação 4.1 não foi capaz de filtrar domínios logo na primeira UR, omitindo do usuário a informação de que o literal *SM = SPARK* deixou de ser viável. As outras implementações o fizeram, além de informar a contradição tão logo o usuário escolheu tal literal como requisito. A implementação 4.1, porém, informou a contradição apenas mais tardiamente, quando o usuário informou um

requisito que fez com que restasse apenas um suporte válido na tabela *cpuCatalog*. Implementações 4.2 e 4.3 ambas trouxeram os mesmos resultados, indicando a equivalência das mesmas em tabelas com representação não compacta.

Quadro 3 – Análise de efetividade dos filtros

#	URs	4.1 (predicados)	4.2 (EC)	4.3 (propagador)
1	*CM = CPU_D*	OK	*SM ≠ SPARK*	*SM ≠ SPARK*
2	Todas de 1, mais *SM = SPARK*	OK	Contradição	Contradição
3	Todas de 2, mais *OC = FALSE*	Contradição	Contradição	Contradição
4	*GM = LEARNING* *CM = CPU_S*	Contradição	NA	Contradição
5	*GM = LEARNING* *CM = CPU_D* *OC = TRUE*	OK	NA	*SM = STAR*
6	Todas de 5, mais *SM = FORGE*	Contradição	NA	Contradição
7	*SM = STAR* *GM = GAMER* *CM = CPU_S*	OK	NA	OK
8	*MM = SILVER* *CM = CPU_D*	OK	NA	*OC = TRUE*
9	Todas de 8, mais *OC = TRUE*	Contradição	NA	Contradição
10	Todas de 8, mais *OC = FALSE*	OK	NA	OK
11	*MM = DIAMOND* *CM = CPU_D* *OC = TRUE*	OK	NA	OK

Fonte: elaborado pelo autor (2024)

Nos cenários envolvendo tabelas em representação compacta de STC, a implementação 4.1 também mostrou não ser capaz de filtrar valores não mais suportados por alguma tabela. Isso é visível nos cenários #5 e #8, onde a filtragem se propaga a partir da ação de tabelas com suporte a intervalos contínuos. Em ambos os casos, porém, é possível observar que a adição de URs já pode criar cenários (*e.g.* cenários #6 e #9, respectivamente) onde a implementação é capaz de identificar o estado de consistência corretamente.

Nos testes realizados neste trabalho, a filtragem abaixo do ideal da implementação 4.1 se mostrou decorrente não apenas da baixa consistência intrínseca à mesma, mas também por características da biblioteca de *solver*. No Choco-Solver, os predicados sobre domínios contínuos são avaliados apenas se as variáveis estão instanciadas e, portanto, esse método tende a não filtrar rótulos de suportes quando uma coluna real tem alguns de seus predicados invalidados.

Os demais cenários do Quadro 3 demonstram que ambas as implementações 4.1 e 4.3 respondem corretamente em situações em que resta um suporte ou menos válido nas restrições envolvidas. Os cenários com NA não se aplicam à implementação 4.2 porque envolvem tabelas compactadas com TCs. No entanto, os mesmos testes foram aplicados sobre a variante do modelo sem tabelas compactadas, usada nos cenários de 1 a 3, e os resultados foram os mesmos obtidos com 4.3 sobre o modelo original.

Estas observações indicam que a implementação 4.3 é a mais versátil para se obter uma boa filtragem de valores inviáveis durante o processo de configuração interativa. No entanto,

para estes cenários, não foram aferidas métricas sobre o uso de recursos computacionais. Tais parâmetros foram coletados e avaliados nos cenários dos testes de desempenho.

O Quadro 4 mostra um compilado dos dados coletados nos testes de desempenho. A primeira linha mostra como os dados estão organizados nas demais. Para a primeira coluna, "Instance" indica o problema avaliado, "UTuple" indica quantas tuplas não comprimidas o problema original tinha, e "STuple" indica quantas tuplas compactadas por TC resultaram do problema transformado, assim como a razão percentual entre "STuple" e "UTuple". Além disso, "SAT" indica que é possível satisfazer o problema, enquanto "UNSAT" indica que não há solução válida que o satisfaça. Para as demais colunas, "Mem" indica a quantidade de memória em *megabytes* consumida para carregar o problema (aferido logo antes de iniciar a busca), ou "EM" para sinalizar estouro de memória. Como a memória mínima alocada pelos testes foi fixada em 64 *megabytes*, os cenários que consumiram menos memória aparecem com essa quantidade. "Time" indica o tempo em segundos para encontrar a primeira solução (ou provar que não é possível satisfazer o modelo), ou "TE" para sinalizar tempo esgotado antes de qualquer conclusão ser alcançada. "N/s" mostra quantos nós da árvore de busca por segundo o *solver* conseguiu visitar por cenário, e "Fails" indica quantas falhas (estados de inconsistência) encontrou antes de encerrar. Por fim, "Props" indica quantos propagadores foram instanciados no modelo para cada cenário. Com exceção deste último que se mostrou constante para cada variante de cenário avaliada, todos os demais são médias aritméticas das dez iterações executadas.

Quadro 4 – Média dos dados coletados para os cenários de desempenho avaliados

Instance			Mem	Time	N/s	Mem	Time	N/s	Mem	Time	N/s	Mem	Time	N/s	Mem	Time	N/s	Mem	Time	N/s
UTuple	STuple	SAT	Props		Fails	Props		Fails	Props		Fails	Props		Fails	Props		Fails	Props		Fails
			NAT			**APB**			**AEM**			**ASE**			**SPB**			**SSE**		
reg-s20-p03-c20-d10-n10-l5-08			64	3,73	3992	272	TE	973	64	148	1,2e4	64	251	4704	316	TE	2070	128	496	1,4e4
10857	7244 (67%)	UNSAT	100		1,5e4	1,3e5		5,8e5	400		1,8e6	400		1,2e6	1e5		1,2e6	400		6,9e6
reg-s20-p03-c20-d10-n10-l5-44			64	3,06	4100	205	TE	1014	64	35	1,3e4	64	104	4749	312	TE	2034	91	44	1,6e4
10896	7239 (66%)	SAT	100		1,3e4	1,3e5		6,1e5	400		4,5e5	400		4,9e5	1e5		1,2e6	400		7,1e5
pigeonsPlus-10-04			71	75	8128	1761	TE	295	64	46	1,4e4	64	310	2601	64	TE	1,7e4	64	391	1,8e5
69040	525 (1%)	UNSAT	65		6,1e5	1,1e6		1,8e5	215		6,4e5	215		8,1e5	1,1e4		9,9e6	215		7,1e7
pigeonsPlus-06-03			64	0,03	9567	64	TE	1,5e4	64	0,04	5575	64	0,05	4420	64	TE	6,1e4	64	0,14	3,2e4
1116	123 (11%)	UNSAT	27		288	1,2e4		9e6	87		224	87		222	1491		3,7e7	87		4414
Kakuro-hard-006-ext			64	0,01	100	169	213	1077	64	0,01	100	64	0,02	50	135	TE	854	86	0,03	33
3772	3772 (100%)	SAT	54		0	6,6e4		2,3e5	214		0	214		0	6,6e4		5,1e5	214		0
Kakuro-medium-044-ext			64	0,01	100	64	242	1537	64	0,01	100	64	0,02	50	132	155	1490	64	0,02	50
1866	1866 (100%)	SAT	58		0	3,3e4		3,7e5	222		0	222		0	3,3e4		2,3e5	222		0
Kakuro-easy-011-ext			64	0,01	100	64	138	2037	64	0,01	100	64	0,02	50	130	165	2254	64	0,02	50
2070	2070 (100%)	SAT	52		0	3,3e4		2,8e5	212		0	212		0	3,3e4		3,7e5	212		0
Renault-big-pos			465	0,1	620	EM	-	-	470	0,08	525	548	0,36	183	874	3,84	557	429	TE	2,2e4
225989	1413 (1%)	SAT	332		0	-		-	1159		1	1159		4	2,8e5		2056	1159		1,3e7
Renault-medium-pos			64	0,02	450	457	TE	582	64	0,02	650	64	0,04	150	102	TE	5432	72	73	1,3e5
9532	544 (6%)	SAT	174		0	1,9e5		3,5e5	601		1	601		0	1,5e4		3,3e6	601		9,8e6
Renault-small-pos			64	0,01	1900	64	13	1,2e4	64	0,01	1900	64	0,02	800	64	0,03	1267	64	0,03	5933
3044	347 (11%)	SAT	147		0	3e4		1,6e5	472		0	472		2	7261		22	472		143

Fonte: elaborado pelo autor (2024)

Sendo cada linha uma variante, cada coluna representa uma abordagem, e seu cruzamento mostra cada um dos 80 cenários avaliados. As abordagens aparecem abreviadas, e são descritas a seguir. NAT é a abordagem de controle, que usa a implementação nativa do Choco-Solver sobre o problema original. É a única que ignora o pré-processamento que converte inteiros em reais. APB é a abordagem baseada em predicados, sobre a implementação 4.1. AEM usa EC, através da implementação 4.2, enquanto ASE usa o algoritmo de filtro proposto em 4.3. SPB e SSE são as abordagens que usam as tabelas transformadas, sendo análogas a APB e ASE, respectivamente.

Os cenários foram preparados como tarefas de satisfação, buscando encontrar uma solução, ou provar que não é possível satisfazer o problema dentro de uma janela de 10 minutos. Devido às conversões realizadas sobre modelos de domínio finito, as CNs convertidas não representam o mesmo CSP, no entanto, para os cenários sem compactação de tabelas, as soluções devem ser compatíveis com a CN original, como descrito na seção 5.2. Dos cenários avaliados todos passaram por esta validação, incluindo os cenários das abordagens SPB e SSE, cuja transformação imposta às tabelas faz com que as CNs geradas fujam à garantia de compatibilidade com a CN original.

Durante os testes, foram identificadas possibilidades de melhoria e pontos de atenção sobre o Algoritmo 1, as quais foram consideradas no código Java usado nos testes descritos pelo Quadro 4. Para evitar a cópia da estrutura de dados `M` a cada chamada do filtro, a aferição do conjunto de valores reais permitidos via a estrutura de repetição das linhas 12 e 13 foi substituída por um novo ramo condicional `senão` ao final da estrutura condicional das linhas 5 a 10. Um vetor foi usado para representar `M`, forçando os rótulos a serem subsequentes, mas com o benefício de eliminar o custo fixo de uma estrutura de dados para mapas genéricos. E por fim, os conjuntos discretos de `M` (provenientes de TCs do tipo "$\in$") foram tratados como vetores ordenados, e a avaliação de suas respectivas interseções com x_r foi realizada através de busca binária. Este último ajuste fino, em particular, proporcionou redução de mais de 80% do tempo de busca em alguns dos cenários avaliados.

Analisando os cenários sobre tabelas não compactadas, vê-se que a abordagem de controle NAT apresenta tempos predominantemente melhores. A quantidade de falhas também se mostra bastante inferior, mostrando correlação com a melhor filtragem da GAC em relação às consistências de intervalo usada em domínios contínuos pelas outras abordagens. Dessa forma, apesar de mostrar uma taxa de visitação de nós menor associada ao maior tempo de propagação, em situações em que é possível discretizar os domínios contínuos, pode ser mais vantajoso fazer a transformação do modelo. Isso permite usufruir do benefício da consistência superior que deve reduzir a quantidade de vezes em que o usuário (ou um *solver* executando uma tarefa de satisfação, por exemplo) escolhe valores inviáveis. Essa transformação, no entanto, foge ao escopo deste trabalho.

Entre as abordagens de domínio contínuo, é possível observar que a APB leva a um consumo de memória superior, o que inclusive inviabilizou o teste em um dos cenários (identificado no Quadro 4 com EM para indicar estouro de memória). Isso se deve à necessidade de criar propagadores e variáveis auxiliares para cada predicado, em cada literal da tabela. É possível observar redução expressiva na quantidade de memória nos cenários compostos pelos modelos `Renault-big-pos` e `Renault-medium-pos`, avaliados com a abordagem SPB, onde o consumo de memória ficou pelo menos quatro vezes menor. Nos demais cenários da mesma abordagem o consumo de memória ou a taxa de compressão das tabelas não se mostrou relevante para indicar a mesma tendência. No entanto, isso demonstra que o uso da implementação 4.1 pode escalar de forma bastante precária, diferente das outras implementações, que mostraram consumo de memória mais próximo do controle.

Já para a comparação entre AEM e ASE, vê-se que a primeira traz tempos melhores para concluir a busca, justificado tanto por uma melhor taxa de visitação de nós, quanto por uma menor taxa de falhas. Analisando o código-fonte dos testes, uma das causas identificadas foi a complexidade de implementação dos propagadores usados por cada uma. Na primeira, o propagador disponibilizado pelo Choco-Solver demanda que a tabela que mapeia índices para valores seja tal que as chaves e os valores estejam em ordem, o que facilita a filtragem e propicia a preservação de informação entre uma chamada e outra do filtro pelo mecanismo de propagação. No entanto, o propagador apresentado no Algoritmo 1, que é usado pela ASE, não permite tal otimização. Visto isso, como as implementações propostas neste trabalho atuam sobre colunas individuais, esta observação justifica aplicações híbridas do *framework*, tais que

colunas reais sem operadores de STC sejam transformadas com a implementação 4.2, que é mais eficiente, e as demais colunas sobre domínio contínuo sejam transformadas com as implementações 4.1 ou a 4.3.

Ainda comparando as abordagens acima, foi observado que o número de falhas antes de encontrar a primeira solução não foi expressivamente diferente na maior parte dos cenários, e para o modelo `reg-s20-p03-c20-d10-n10-l5-08` com AEM, foi ~50% maior que com ASE. Este resultado vai de encontro com o esperado, pois AEM usa uma implementação que gera uma menor quantidade de rótulos se comparada com ASE e, portanto, menos valores para ramificar a árvore de busca, o que leva à intuição de que AEM deveria falhar menos. No entanto, este raciocínio não leva em consideração as heurísticas da estratégia de busca utilizada.

Por fim, comparando apenas as abordagens aplicadas a modelos com tabelas compactadas, é possível observar que a abordagem SSE se destaca em 8 dos 10 cenários, sendo que a SPB excede a janela de tempo pré-definida em 5 deles. Para o modelo `Renault-small-pos`, as medidas são muito pequenas, e não mostram diferença relevante, enquanto que para o cenário `Renault-big-pos` os resultados se invertem, indicando uma possível vantagem dada pelas heurísticas da estratégia de busca padrão do Choco-Solver. No entanto, na tarefa de configuração as heurísticas não são utilizadas, pois as decisões são tomadas pelo próprio usuário com base em seus requisitos e, para tal, a filtragem superior providenciada pela implementação 4.3 (usada por SSE) continua sendo mais vantajosa, quando seu uso é viável.

6. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho apresentou um *framework* para viabilizar a composição de problemas de satisfação de restrições usando restrições tabulares com escopo não limitado a variáveis de domínio finito, comuns na literatura. O *framework* proposto permitiu o uso de variáveis de domínio contínuo, em particular variáveis reais — que são comuns nas bases de conhecimento sobre os produtos configuráveis — nas tabelas através da alteração do fluxo de execução da tarefa de configuração. A alteração incluiu uma etapa intermediária de transformação à tarefa para modificar as tabelas que incluem variáveis reais, substituindo-as por variáveis de rótulo em domínio finito, e adicionando restrições auxiliares para associá-las às variáveis reais substituídas. Além disso, o *framework* foi desenhado de forma a permitir o uso parcial da representação compacta STC, que propicia uma melhor escalabilidade na manutenção da base de conhecimento.

O trabalho também apresentou três implementações para o *framework*, cada qual usando diferentes tipos de recursos e com diferentes limitações. A primeira utilizou recursos mais básicos, se limitando a construir restrições auxiliares através da composição de restrições primitivas, envolvendo variáveis reais e inteiras. Testes demonstraram que esta implementação apresenta consumo de memória elevado, e uma baixa capacidade de filtragem de valores que não levam a solução, o que pode reduzir o apoio proporcionado pelo configurador ao usuário na tomada de decisão, além de aumentar o custo da busca em tarefas de satisfação. No entanto, sua proposta simples pode ser conveniente quando uma filtragem eficiente não é mandatória. Já a segunda implementação usa restrições auxiliares do tipo EC, que atua com visão global sobre seu escopo, permitindo maior nível de filtragem. No entanto, esta implementação em particular impede que sejam usados recursos de STC sobre as variáveis reais que aparecem nas tabelas. Apesar da limitação, mostrou desempenho superior nos cenários cobertos neste trabalho aos quais se aplica, além da filtragem de alta eficiência, o que a posicionou como a mais indicada para aplicação em configuradores e em problemas de satisfação. Por fim, a terceira implementação apresentada envolveu a definição de uma nova restrição global inspirada na restrição EC, além do desenvolvimento de um propagador para a mesma. Esta nova restrição foi usada como auxiliar, propiciando uma filtragem equivalente à da segunda

implementação, mas sem a limitação sobre os recursos de STC. Como a implementação com a nova restrição apresentou desempenho levemente inferior à implementação baseada em EC, seu uso é indicado como complementar, apenas nos casos em que a baseada em EC não atende por falta de representatividade.

A busca por trabalhos correlatos encontrou bons avanços relacionados a restrições tabulares, no entanto, não foram encontrados trabalhos recentes estendendo tais avanços para domínios contínuos. Dessa forma, ficam em aberto vários pontos que demandam pesquisa mais aprofundada. Primeiramente, o *framework* limita os recursos de STC disponibilizados, não fornecendo alternativas para TCs que relacionam colunas distintas em um mesmo suporte, por exemplo. Outro ponto relevante é que os cenários avaliados produziram soluções reais compostas por intervalos de tamanho mínimo equivalente à granularidade da representação de ponto flutuante com precisão dupla (muito pequenos). Isso pode ter levado os comportamentos observados a se assimilarem excessivamente ao esperado para domínios finitos. Fica em aberto a avaliação do comportamento do *framework* em cenários de maior complexidade, envolvendo expressões aritméticas mais elaboradas que as usadas neste trabalho.

Também foram identificadas possíveis melhorias, tanto na eficácia da filtragem quanto na eficiência computacional do raciocínio. Para a filtragem, um pós-filtro sobre o resultado da tarefa de configuração, considerando o conhecimento transformado pelo *framework*, poderia identificar descontinuidades sobre os intervalos filtrados, reduzindo o espaço de valores que não levam a uma solução, apresentado como efeito colateral indesejado ao usuário. E para melhorias de eficiência, destacam-se: a) a duplicidade de variáveis de rótulo de suportes, que são recriadas a cada variável de domínio contínuo transformada no escopo da tabela; b) a duplicidade de literais para uma mesma variável na transformação; e c) a possibilidade de ordenação conveniente dos conjuntos no propagador da terceira implementação, a fim de obter otimizações similares à possível na EC, como travessia parcial da coleção de conjuntos e uso de objetos reversíveis para preservar informação entre propagações.

Não obstante, espera-se que o *framework* e suas implementações propiciem o desenvolvimento de configuradores de produtos mais flexíveis, eficientes e manuteníveis, capazes de fornecer ao usuário um maior número de opções de parametrização, de modo a permitir a oferta de produtos mais customizáveis ao cliente por parte da indústria. Além disso, como o framework está inserido em um contexto de CSP, espera-se que problemas envolvendo outras tarefas sobre restrições também sejam contemplados, como por exemplo, o controle de tráfego, o gerenciamento de armazéns, e a consistência de entrada de dados em formulários eletrônicos.

7. REFERÊNCIAS

AUDEMARD, Gilles; LECOUTRE, Christophe; MAAMAR, Mehdi. Segmented Tables: an efficient modeling tool for constraint reasoning. In: EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, 24., 2020, [S.L.]. **Frontiers in Artificial Intelligence and Applications**. [S.L.]: Ios Press, 2020. v. 325, p. 315-322. Disponível em: https://doi.org/10.3233/FAIA200108. Acesso em: 29 out. 2023.

AUDEMARD, Gilles; LECOUTRE, Christophe; PRUD'HOMME, Charles. Guiding Backtrack Search by Tracking Variables During Constraint Propagation. In: INTERNATIONAL CONFERENCE ON PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING, 29., 2023, Toronto. **Leibniz International Proceedings in Informatics**. [S.L.]: Schloss Dagstuhl – Leibniz-Zentrum Für Informatik, 2023. v. 280, p. 91-917. Disponível em: https://doi.org/10.4230/LIPIcs.CP.2023.9. Acesso em: 23 jan. 2024.

BELDICEANU, Nicolas; CONTEJEAN, Évelyne. Introducing Global Constraints in CHIP. **Mathematical And Computer Modelling**, [S.L.], v. 20, n. 12, p. 97-123, 1994. Elsevier BV. Disponível em: http://doi.org/10.1016/0895-7177(94)90127-9. Acesso em: 11 fev. 2024.

BELDICEANU, Nicolas *et al*. On the Reification of Global Constraints. **Constraints**, [S.L.], v. 18, n. 1, p. 1-6, 2013. Springer Science and Business Media LLC. Disponível em: https://doi.org/10.1007/s10601-012-9132-0. Acesso em: 12 fev. 2024.

BENHAMOU, Frédéric. Interval Constraint Logic Programming. In: PODELSKI, Andreas (ed.). **Constraint Programming**: basics and trends. [S.L.]: Springer Berlin Heidelberg, 1995. p. 1-21. (Lecture Notes in Computer Science). Disponível em: https://doi.org/10.1007/3-540-59155-9_1. Acesso em: 13 jun. 2024.

BENHAMOU, Frédéric; OLDER, William J.. Applying Interval Arithmetic to Real, Integer, and Boolean Constraints. **The Journal Of Logic Programming**, [S.L.], v. 32, n. 1, p. 1-24, 1997. Elsevier BV. Disponível em: https://doi.org/10.1016/S0743-1066(96)00142-2. Acesso em: 13 jun. 2024.

BENNAI, Soufia *et al*. An Efficient Heuristic Approach Combining Maximal Itemsets and Area Measure for Compressing Voluminous Table Constraints. **The Journal Of Supercomputing**, [S.L.], v. 79, n. 1, p. 650-676, 2023. Springer Science and Business Media LLC. Disponível em: https://doi.org/10.1007/s11227-022-04667-1. Acesso em: 1 nov. 2023.

BESSIÈRE, Christian. Constraint Propagation. In: ROSSI, Francesca; VAN BEEK, Peter; WALSH, Toby. **Handbook of Constraint Programming**. Amsterdan: Elsevier, 2006. Cap. 3. p. 29-83. Disponível em: https://doi.org/10.1016/S1574-6526(06)80007-6. Acesso em: 25 fev. 2024.

BESSIÈRE, Christian *et al*. Decompositions of All Different, Global Cardinality and Related Constraints. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 21., 2009, Pasadena. **Proceedings [...]**. [S.L.]: Ijcai Organization, 2009. v. 4, p. 419-424. Disponível em: https://www.ijcai.org/Proceedings/09/Papers/077.pdf. Acesso em: 11 fev. 2024.

BESSIÈRE, Christian; RÉGIN, Jean-Charles. Arc Consistency for General Constraint Networks: preliminary results. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 15., 1997, Nagoya. **Proceedings [...]**. [S.L.]: Morgan Kaufmann, 1998. p. 398-404. Disponível em: https://www.researchgate.net/profile/Jean-Charles-Regin/publication/220813650_Arc_Consistency_for_General_Constraint_Networks_Preliminary_Results/links/551e99bc0cf29dcabb0445a2/Arc-Consistency-for-General-Constraint-Networks-Preliminary-Results.pdf. Acesso em: 26 fev. 2024.

CARVALHO, Christopher de. Desenvolvimento de uma PoC para Recomendação de Diagnósticos em Configuradores de Produtos Baseados em Restrições. 2021. 95 f. TCC (Graduação) - Curso de Graduação em Engenharia de Controle e Automação, Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, 2021. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/226098/TCC.pdf. Acesso em: 23 dez. 2021.

CHMEISS, Assef; SAIS, Lakhdar. Constraint Satisfaction Problems: backtrack search revisited. In: INTERNATIONAL CONFERENCE ON TOOLS WITH ARTIFICIAL INTELLIGENCE, 16., 2004, Boca Raton. **Proceedings [...]**. [S.L.]: IEEE, 2005. p. 252-257. Disponível em: https://doi.org/10.1109/ICTAI.2004.43. Acesso em: 24 fev. 2024.

CRUZ, Jorge; BARAHONA, Pedro. Maintaining Global Hull Consistency With Local Search for Continuous CSPs. In: INTERNATIONAL WORKSHOP ON GLOBAL CONSTRAINTS OPTIMIZATION AND CONSTRAINT SATISFACTION, 1., 2002, Valbonne. **Global Optimization and Constraint Satisfaction**. Heidelberg: Springer Berlin, 2003. p. 178-193. Disponível em: https://doi.org/10.1007/978-3-540-39901-8_14. Acesso em: 10 jan. 2024.

DECHTER, Rina. Learning While Searching in Constraint Satisfaction Problems. In: AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 5., 1986, Philadelphia. **Proceedings [...]**. [S.L.]: Aaai Press, 1986. p. 178-183. Disponível em: https://cdn.aaai.org/AAAI/1986/AAAI86-029.pdf. Acesso em: 23 jan. 2024.

DEMEULENAERE, Jordan. **Efficient Algorithms for Table Constraints**. 2015. 49 f. Dissertação (Mestrado) - Curso de Computer Science And Engineering, Uclouvain, Louvain-La-Neuve, 2015. Cap. 3. Disponível em: https://dial.uclouvain.be/downloader/downloader.php?pid=thesis%3A440&datastream=PDF_01. Acesso em: 05 jun. 2024.

DONG, Liang *et al*. A Novel Smart Product-Service System Configuration Method for Mass Personalization Based on Knowledge Graph. **Journal Of Cleaner Production**, [S.L.], v. 382, p. 135270, 2023. Elsevier BV. Disponível em: https://doi.org/10.1016/j.jclepro.2022.135270. Acesso em: 28 ago. 2023.

FAGES, Jean-Guillaume; CHABERT, Gilles; PRUD'HOMME, Charles. Combining Finite and Continuous Solvers Towards a Simpler Solver Maintenance. In: INTERNATIONAL CONFERENCE ON PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING, 19., 2013, Uppsala. **TRICS'13 workshop: Techniques for implementing constraint programming systems**. [S.L.]: Hal, 2013. p. 1-9. Disponível em: https://hal.science/hal-00904069. Acesso em: 11 fev. 2024.

FAGES, Jean-Guillaume; PRUD'HOMME, Charles. Making the First Solution Good! In: INTERNATIONAL CONFERENCE ON TOOLS FOR ARTIFICIAL INTELLIGENCE, 29., 2017, Boston. **Proceedings [...]**. [S.L.]: IEEE, 2018. p. 1073-1077. Disponível em: https://doi.org/10.1109/ICTAI.2017.00164. Acesso em: 23 jan. 2024.

FALKNER, Andreas *et al*. Solver Requirements for Interactive Configuration. **Journal Of Universal Computer Science**, [S.L.], v. 26, n. 3, p. 343-373, 2020. Disponível em: https://www.researchgate.net/profile/Richard-Comploi-Taupe/publication/340526513_Solver_Requirements_for_Interactive_Configuration/links/5e8ec43292851c2f528d3ff8/Solver-Requirements-for-Interactive-Configuration.pdf. Acesso em: 05 set. 2023.

FALKNER, Andreas *et al*. Twenty-Five Years of Successful Application of Constraint Technologies at Siemens. **AI Magazine**, [S.L.], v. 37, n. 4, p. 67-80, 2016. Wiley. Disponível em: https://doi.org/10.1609/aimag.v37i4.2688. Acesso em: 27 fev. 2024.

FELFERNIG, Alexander *et al*. Benefits of Configuration Systems. In: ____. **Knowledge-Based Configuration**: from research to business cases. [S.L.]: Elsevier, 2014. Cap. 4. p. 29-33. Disponível em: https://doi.org/10.1016/B978-0-12-415817-7.00004-9.

FREUDER, Eugene C.; MACKWORTH, Alan K.. Constraint Satisfaction: an emerging paradigm. In: ROSSI, Francesca; VAN BEEK, Peter; WALSH, Toby (ed.). **Handbook of Constraint Programming**. Amsterdan: Elsevier, 2006. Cap. 2. p. 13-27. Disponível em: https://doi.org/10.1016/S1574-6526(06)80006-4. Acesso em: 25 fev. 2024.

GOLOMB, Solomon W.; BAUMERT, Leonard D.. Backtrack Programming. **Journal Of The ACM**, [S.L.], v. 12, n. 4, p. 516-524, 1965. Association for Computing Machinery. Disponível em: https://doi.org/10.1145/321296.321300. Acesso em: 24 fev. 2024.

GRANVILLIERS, Laurent; BENHAMOU, Frédéric. Progress in the Solving of a Circuit Design Problem. **Journal Of Global Optimization**, [S.L.], v. 20, n. 2, p. 155-168, 2001. Springer Science and Business Media LLC. Disponível em: https://doi.org/10.1023/A:1011266226870. Acesso em: 13 jun. 2024.

HOTZ, Lothar *et al*. Configuration Knowledge Representation and Reasoning. In: FELFERNIG, Alexander *et al*. (ed.). **Knowledge-Based Configuration**: from research to business cases. [S.L.]: Elsevier, 2014. Cap. 6. p. 41-72. Disponível em: https://doi.org/10.1016/B978-0-12-415817-7.00006-2.

HOUGH, David G.. The IEEE Standard 754: one for the history books. **Computer**, [S.L.], v. 52, n. 12, p. 109-112, 2019. Institute of Electrical and Electronics Engineers (IEEE). Disponível em: https://doi.org/10.1109/mc.2019.2926614. Acesso em: 20 out. 2023.

HULGAARD, Henrik. **Virtual Tabulation**: the 3. generation configuration technology. 2020. Disponível em: https://go.configit.com/l/265662/2017-07-25/2crk2/265662/16643/Virtual_Tabulation.pdf. Acesso em: 27 fev. 2024.

JAULIN, Luc *et al*. Interval Analysis. In: ____. **Applied Interval Analysis**: with examples in parameter and state estimation, robust control and robotics. [S.L.]: Springer-Verlag, 2001. Cap. 2. p. 11-43. Disponível em: https://doi.org/10.1007/978-1-4471-0249-6_2. Acesso em: 10 jan. 2024.

JUNKER, Ulrich. Configuration. In: ROSSI, Francesca; VAN BEEK, Peter; WALSH, Toby (ed.). **Handbook of Constraint Programming**. Amsterdan: Elsevier, 2006. Cap. 24. p. 837-873. Disponível em: https://doi.org/10.1016/S1574-6526(06)80028-3. Acesso em: 28 fev. 2024.

KATSIRELOS, George; WALSH, Toby. A Compression Algorithm for Large Arity Extensional Constraints. In: PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING, 13., 2007, Providence. **Proceedings [...]**. [S.L.]: Springer Berlin Heidelberg, 2007. p. 379-393. Disponível em: https://doi.org/10.1007/978-3-540-74970-7_28. Acesso em: 08 maio 2024.

KUMAR, Vipin. Algorithms for Constraint Satisfaction Problems: a survey. **AI Magazine**, USA, v. 13, n. 1, p. 32-44, 1992. American Association for Artificial Intelligence. Disponível em: https://doi.org/10.1609/aimag.v13i1.976. Acesso em: 23 jan. 2024.

LE, Viet-Man *et al*. FASTDIAGP: an algorithm for parallelized direct diagnosis. In: AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 37., 2023, Washington, Dc. **Proceedings [...]**. [S.L.]: Association For The Advancement Of Artificial Intelligence, 2023. p. 6442-6449. Disponível em: https://doi.org/10.1609/aaai.v37i5.25792. Acesso em: 25 set. 2023.

LECOUTRE, Christophe. STR2: optimized simple tabular reduction for table constraints. **Constraints**, [S.L.], v. 16, n. 4, p. 341-371, 2011. Springer Science and Business Media LLC. Disponível em: https://doi.org/10.1007/s10601-011-9107-6. Acesso em: 13 fev. 2024.

LECOUTRE, Christophe; SZYMANEK, Radoslaw. Generalized Arc Consistency for Positive Table Constraints. In: PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING, 12., 2006, Nantes. **Proceedings [...]**. [S.L.]: Springer Berlin Heidelberg, 2006. p. 284-298. Disponível em: https://doi.org/10.1007/11889205_22. Acesso em: 13 fev. 2024.

LEITNER, Gerhard *et al*. User Interfaces for Configuration Environments. In: FELFERNIG, Alexander *et al*. (ed.). **Knowledge-Based Configuration**: from research to business cases. [S.L.]: Elsevier, 2014. Cap. 8. p. 89-106. Disponível em: https://doi.org/10.1016/B978-0-12-415817-7.00008-6.

LHOMME, Oliver. Consistency techniques for numeric CSPs. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 13., 1993, Chambery. **Proceedings [...]**. Chambery, France: Morgan Kaufmann Publishers, 1993. p. 232-238. Disponível em: https://www.ijcai.org/Proceedings/93-1/Papers/033.pdf. Acesso em: 02 jan. 2024.

LHOMME, Olivier. Arc-Consistency Filtering Algorithms for Logical Combinations of Constraints. In: INTEGRATION OF AI AND OR TECHNIQUES IN CONSTRAINT PROGRAMMING FOR COMBINATORIAL OPTIMIZATION PROBLEMS, 1., 2004, Nice. **Proceedings [...]**. [S.L.]: Springer Berlin Heidelberg, 2004. p. 209-224. Disponível em: https://doi.org/10.1007/978-3-540-24664-0_15. Acesso em: 13 fev. 2024.

LI, Zhe *et al*. A Novel Multi-Thread Parallel Constraint Propagation Scheme. **Ieee Access**, [S.L.], v. 7, p. 167823-167835, 2019. Institute of Electrical and Electronics Engineers (IEEE). Disponível em: https://doi.org/10.1109/access.2019.2951027. Acesso em: 02 nov. 2023.

LIKITVIVATANAVONG, Chavalit; XIA, Wei; YAP, Roland Hock Chuan. Higher-Order Consistencies through GAC on Factor Variables. In: PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING, 20., 2014, Lyon. **Proceedings [...]**. [S.L.]: Springer Cham, 2014. p. 497-513. Disponível em: https://doi.org/10.1007/978-3-319-10428-7_37. Acesso em: 14 jun. 2024.

MACKWORTH, Alan K.. Consistency in Networks of Relations. **Artificial Intelligence**, [S.L.], v. 8, n. 1, p. 99-118, 1977. Elsevier BV. Disponível em: https://doi.org/10.1016/0004-3702(77)90007-8. Acesso em: 11 fev. 2024.

MAIRY, Jean-Baptiste; DEVILLE, Yves; LECOUTRE, Christophe. Domain k-Wise Consistency Made as Simple as Generalized Arc Consistency. In: CPAIOR, 11., 2014, Cork. **Integration of AI and OR Techniques in Constraint Programming**. [S.L.]: Springer Cham, 2014. p. 235-250. Disponível em: https://doi.org/10.1007/978-3-319-07046-9_17. Acesso em: 14 jun. 2024.

MAIRY, Jean-Baptiste; DEVILLE, Yves; LECOUTRE, Christophe. The Smart Table Constraint. In: CPAIOR, 12., 2015, Barcelona. **Integration of AI and OR Techniques in Constraint Programming**. [S.L.]: Springer Cham, 2015. p. 271-287. Disponível em:http://dx.doi.org/10.1007/978-3-319-18008-3_19. Acesso em: 14 jun. 2024.

MOON, Junyean; CHADEE, Doren; TIKOO, Surinder. Culture, Product Type, and Price Influences on Consumer Purchase Intention to Buy Personalized Products Online. **Journal Of Business Research**, [S.L.], v. 61, n. 1, p. 31-39, 2008. Elsevier BV. Disponível em: https://doi.org/10.1016/j.jbusres.2006.05.012. Acesso em: 25 set. 2023.

NETTO, Paulo Oswaldo Boaventura. **Grafos**: teoria, modelos, algoritmos. 5. ed. São Paulo: Blücher, 2022. 311 p.

NIGHTINGALE, P. *et al*. Short and Long Supports for Constraint Propagation. **Journal Of Artificial Intelligence Research**, [S.L.], v. 46, p. 1-45, 2013. AI Access Foundation. Disponível em: https://doi.org/10.1613/jair.3749. Acesso em: 13 jun. 2024.

PAPARRIZOU, Anastasia; STERGIOU, Kostas. Strong Local Consistency Algorithms for Table Constraints. **Constraints**, [S.L.], v. 21, n. 2, p. 163-197, 2015. Springer Science and Business Media LLC. Disponível em: https://doi.org/10.1007/s10601-014-9179-1. Acesso em: 31 out. 2023.

PESANT, Gilles. A Constraint Programming Primer. **Euro Journal On Computational Optimization**, [S.L.], v. 2, n. 3, p. 89-97, 2014. Elsevier BV. Disponível em: https://doi.org/10.1007/s13675-014-0026-3. Acesso em: 10 set. 2023.

PRUD'HOMME, Charles; FAGES, Jean-Guillaume. Choco-Solver: a Java library for constraint programming. **Journal of Open Source Software**, [S.L.], v. 7, n. 78, p. 4708, 2022. The Open Journal. Disponível em: https://doi.org/10.21105/joss.04708. Acesso em: 13 mar. 2024.

PRUD'HOMME, Charles; FAGES, Jean-Guillaume; LORCA, Xavier. **Choco Solver Documentation**: release 4.0.8. TASC, INRIA Rennes, LINA CNRS UMR, 2016. 36 p. Disponível em: https://www.dcs.gla.ac.uk/~pat/cpM/choco4/user_guide-4.0.8.pdf. Acesso em: 14 fev. 2024.

ROSSI, Francesca; VAN BEEK, Peter; WALSH, Toby. Introduction. In: ____. **Handbook of Constraint Programming**. Amsterdan: Elsevier, 2006. Cap. 1. p. 3-12. Disponível em: https://doi.org/10.1016/S1574-6526(06)80005-2. Acesso em: 25 fev. 2024.

SCHNEIDER, Anthony; CHOUEIRY, Berthe Y.. PW-CT: Extending Compact-Table to Enforce Pairwise Consistency on Table Constraints. In: PRINCIPLES AND PRACTICE OF CONSTRAINT PROGRAMMING, 24., 2018, Lille. **Proceedings [...]**. [S.L.]: Springer International Publishing, 2018. p. 345-361. Disponível em: https://doi.org/10.1007/978-3-319-98334-9_48. Acesso em: 01 nov. 2023.

SCHULTE, Christian; STUCKEY, Peter J.. Efficient constraint propagation engines. **Acm Transactions On Programming Languages And Systems**, [S.L.], v. 31, n. 1, p. 1-43, 2008. Association for Computing Machinery (ACM). Disponível em: https://doi.org/10.1145/1452044.1452046. Acesso em: 05 mar. 2024.

STOLZENBURG, Frieder. Membership-Constraints and Complexity in Logic Programming with Sets. In: BAADER, Frans; SCHULZ, Klaus U. (ed.). **Frontiers of Combining Systems**: first international workshop. Munich: Springer, Dordrecht, 1996. p. 285-302. Disponível em: https://doi.org/10.1007/978-94-009-0349-4_15. Acesso em: 12 fev. 2024.

ULLMANN, Julian R.. Partition Search for Non-Binary Constraint Satisfaction. **Information Sciences**, [S.L.], v. 177, n. 18, p. 3639-3678, 2007. Elsevier BV. Disponível em: https://doi.org/10.1016/j.ins.2007.03.030. Acesso em: 13 fev. 2024.

VAN HENTENRYCK, Pascal; CARILLON, Jean-Philippe. Generality versus specificity: an experience with ai and or techniques. In: AAAI NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 7., 1988, Saint Paul. **Proceedings [...]**. [S.L]: Aaai Press, 1988. p. 660-664. Disponível em: https://cdn.aaai.org/AAAI/1988/AAAI88-117.pdf. Acesso em: 01 maio 2024.

WANG, Ruiwei *et al*. Optimizing Simple Tabular Reduction with a Bitwise Representation. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 25., 2016, New York. **Proceedings [...]**. Palo Alto, California: AAAI Press, 2016. p. 787-793. Disponível em: https://www.ijcai.org/Proceedings/16/Papers/117.pdf. Acesso em: 13 fev. 2024.

WATTEZ, Hugues *et al*. Refining Constraint Weighting. In: INTERNATIONAL CONFERENCE ON TOOLS WITH ARTIFICIAL INTELLIGENCE, 31., 2019, Portland. **Proceedings [...]**. [S.L.]: IEEE, 2020. p. 71-77. Disponível em: https://doi.org/10.1109/ICTAI.2019.00019. Acesso em: 23 jan. 2024.

YAP, Roland H. C.; XIA, Wei; WANG, Ruiwei. Generalized Arc Consistency Algorithms for Table Constraints: a summary of algorithmic ideas. In: AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 34., 2020, New York. **Proceedings [...]**. [S.L.]: AAAI, 2020. v. 9, p. 13590-13597. Disponível em: https://doi.org/10.1609/aaai.v34i09.7086. Acesso em: 28 jan. 2024.

YOUNSI, Zineb *et al.* HSJ-Solver: a new method based on GHD for answering conjunctive queries and solving constraint satisfaction problems. **Applied Intelligence**, [S.L.], v. 53, n. 13, p. 17226-17239, 2023. Springer Science and Business Media LLC. Disponível em: https://doi.org/10.1007/s10489-022-04361-y. Acesso em: 13 jun. 2024.