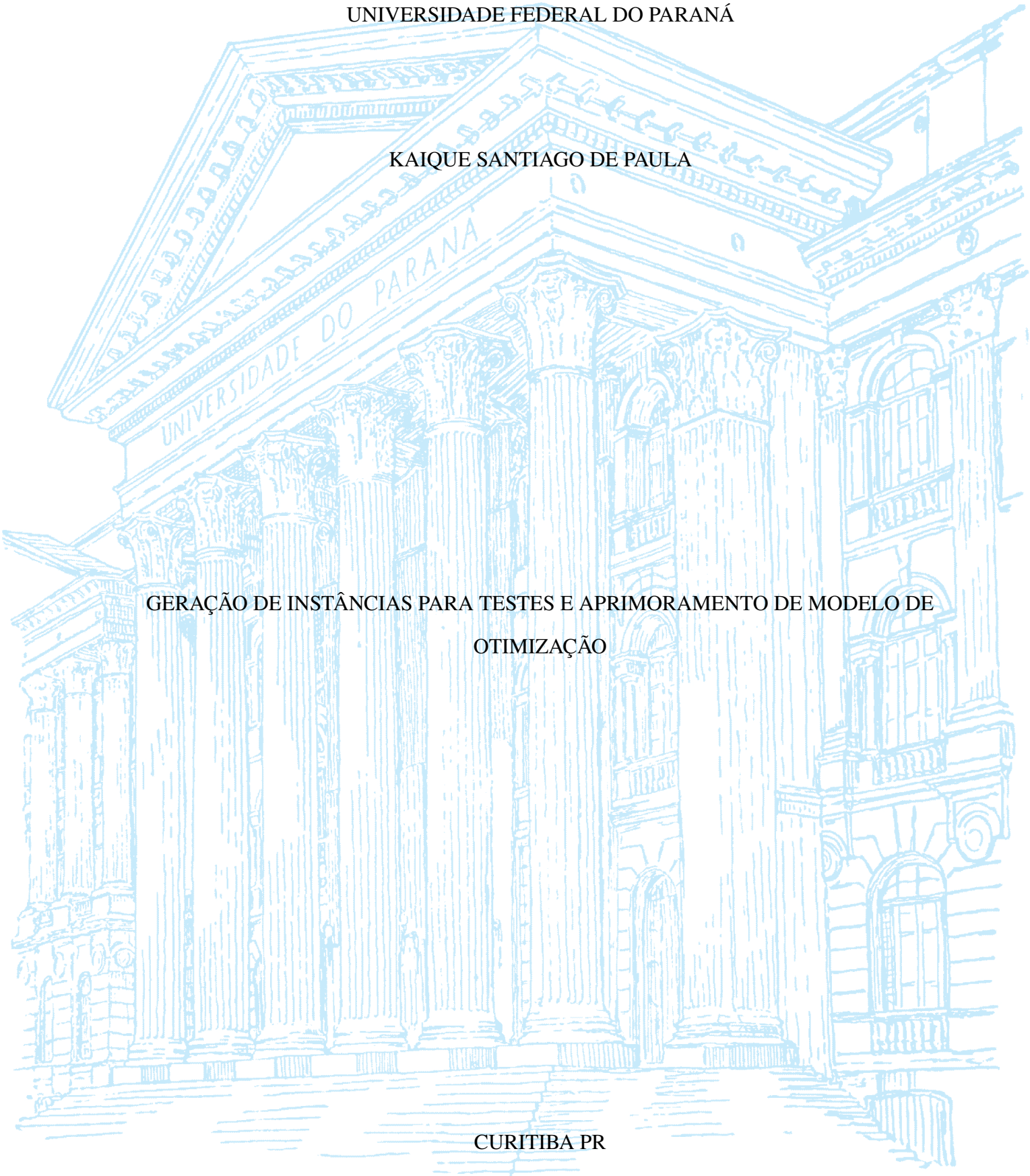
UNIVERSIDADE FEDERAL DO PARANÁ

KAIQUE SANTIAGO DE PAULA



GERAÇÃO DE INSTÂNCIAS PARA TESTES E APRIMORAMENTO DE MODELO DE OTIMIZAÇÃO

CURITIBA PR

2023

KAIQUE SANTIAGO DE PAULA

GERAÇÃO DE INSTÂNCIAS PARA TESTES E APRIMORAMENTO DE MODELO DE OTIMIZAÇÃO

Monografia apresentada como requisito parcial à obtenção do grau de Graduado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Guilherme Derenievicz.

CURITIBA PR

2023

Dedico a Ednamar Santiago de Paula, minha mãe, que não pôde viver o suficiente para poder ler essa dedicatória, mas que será para sempre lembrada por todos que a amaram. Dedico também a Manoel Júlio de Paula, meu pai, que esteve disposto a navegar comigo em muitas águas sem ter a real certeza de que chegaríamos a algum lugar.

AGRADECIMENTOS

Agradecimentos ao meu orientador, Guilherme Derenievicz, que muito me guiou ao longo do último ano com muita paciência. Agradeço, também, a Fernando Monteiro Kiotheka, veterano e amigo nessa breve trajetória pelo BCC. E por último, porém não menos importantes, agradeço a cada professor que contribuiu para o meu aprendizado neste curso.

RESUMO

O Governo Federal, por meio do Programa Nacional do Livro e Material Didático, fornece material didático para escolas públicas em toda extensão territorial brasileira. Essa tarefa de alta complexidade exige esforços logísticos de planejamento, administração e roteamento que perpassam por múltiplas etapas, que representam em conjunto um problema de otimização de larga escala. No intuito de contribuir para o projeto, modelagens matemáticas e algoritmicas foram sugeridas para melhorias em cada uma das etapas do Programa e, ao longo das análises da problemática, foi observado que quanto mais otimizada for a solução da primeira etapa, a de Paletização, mais eficientes poderiam ser as etapas subsequentes. Para o aprimoramento da modelagem fornecida, este trabalho apresenta um Gerador de Instâncias, capaz de criar entradas estruturadas, aleatorizadas de forma limitada, capazes de alimentar o modelo de paletização. A geração de instâncias desempenha um papel crucial no desenvolvimento e aprimoramento de modelos e algoritmos em diversas áreas, destacando-se especialmente em problemas de otimização. Essas instâncias, ou conjuntos específicos de dados que representam situações reais ou simuladas, são essenciais para testar a eficácia e a robustez das soluções propostas. Nesse trabalho, as instâncias criadas pelo gerador são valiadas com base na performance de sua execução nos modelos de paletização. Ainda, um modelo de Paletização Englobada é proposto como adição ao material presente na literatura e sua eficiência é comparada ao modelo original. Além dessas análises, a performance dos modelos de paletização é analisada sob o prisma da ferramenta MiniZinc e de seus múltiplos solvers disponíveis. Uma vez realizadas essas análises, conclui-se que a escolha dos solvers empregados no MiniZinc deve ser adaptada ao tamanho da entrada. Ainda, é possível notar que o modelo proposto de paletização Englobada causa impactos na eficiência da busca pela solução, se comparado ao modelo de paletização simples.

Palavras-chave: Geração de Instância. Problema de Otimização. Problema de Carregamento de Paletes.

ABSTRACT

The Federal Government, through the National Book and Teaching Material Program, provides teaching material to public schools throughout Brazil. This highly complex task requires logistical planning, administration and routing efforts that go through multiple stages, which together represent a large-scale optimization problem. In order to contribute to the project, mathematical and algorithmic modeling was suggested for improvements in each of the stages of the Program and, throughout the analysis of the problem, it was observed that the more optimized the solution in the first stage, Palletization, the more efficient could be the subsequent steps. To improve the modeling provided, this work presents an Instance Generator, capable of creating structured inputs, randomized in a limited way, capable of feeding the palletization model. Instance generation plays a crucial role in the development and improvement of models and algorithms in several areas, especially in optimization problems. These instances, or specific sets of data that represent real or simulated situations, are essential for testing the effectiveness and robustness of proposed solutions. In this work, the instances created by the generator are evaluated based on the performance of their execution in the palletization models. Furthermore, an Enclosed Palletizing model is proposed as an addition to the material present in the literature and its efficiency is compared to the original model. In addition to these analyses, the performance of the palletizing models is analyzed from the perspective of the MiniZinc tool and its multiple available solvers. Once these analyzes have been carried out, it is concluded that the choice of solvers used in MiniZinc must be adapted to the size of the input. Furthermore, it is possible to note that the proposed Encompassed palletization model impacts the efficiency of the search for a solution, compared to the simple palletization model.

Keywords: Instance Generation. Optimization Problem. Pallet Loading Problem.

LISTA DE FIGURAS

2.1 Problema de Coloração 13
2.2 Exemplo de um Sudoku 14
2.3 Backtracking 17
2.4 Enigma de Einstein 18

3.1 Esquema Visual do Problema. Imagens do Estudo Preliminar. 24

4.1 Coparação de Tempos - Geocode 31
4.2 Comparação de Tempos (Grandes Entradas) - Geocode. Também soluções satisfatórias apenas 32
4.3 Comparação de Tempos - Multiplos Solvers 33
4.4 Comparação de Tempos - Simples vs Englobado 35

LISTA DE TABELAS

4.1 Parâmetros de Entrada do Gerador de Instâncias 29
4.2 Parâmetros de Entrada do Gerador de Instâncias 34

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná
PNLD	Programa Nacional do Livro e do Material Didatico
PCP	Problema de Carregamento de Paletes
PRV	Problema de Roteamento de Veículos
PSR	Problema de Satisfação de Restrições
CABSC	Constraint Acquisition Based on Solution Counting
CSP	Constraint Satisfaction Problem
FNDE	Fundo Nacional de Desenvolvimento da Educação

SUMÁRIO

1 INTRODUÇÃO 10
1.1 JUSTIFICATIVA 10
1.2 OBJETIVOS 11
1.2.1 Objetivos Específicos 11
1.3 ESTRUTURA DO DOCUMENTO 11
2 FUNDAMENTAÇÃO TEÓRICA 12
2.1 PROBLEMAS DE SATISFAÇÃO DE RESTRIÇÕES 12
2.2 EXEMPLOS DE PSR 13
2.3 BUSCANDO SOLUÇÕES PARA OS PSR 15
2.4 PROGRAMAÇÃO POR RESTRIÇÕES 18
2.4.1 Solvers no MiniZinc 20
2.5 PROBLEMA DE CARREGAMENTO DE PALETES E DE ROTEAMENTO DE VEÍCULOS 21
2.6 REVISÃO BIBLIOGRÁFICA 21
3 APRESENTAÇÃO DA PROBLEMÁTICA 23
3.1 PROGRAMA NACIONAL DO LIVRO E DO MATERIAL DIDÁTICO - PNLD 23
3.2 DEFINIÇÕES DO PROBLEMA E MODELAGENS 24
4 SOLUÇÕES E ANÁLISES 28
4.1 GERAÇÃO DE INSTÂNCIAS 28
4.2 ANÁLISE DE EXECUÇÃO NO MINIZINC 30
4.3 RESULTADOS 31
5 CONCLUSÃO 36
REFERÊNCIAS 38
APÊNDICE A – MODELAGEM DO PROBLEMA DE EINSTEIN. 39
APÊNDICE B – MODELAGEM DA PALETIZAÇÃO ENGLOBADA . . . 41
APÊNDICE C – EXEMPLO DE ARQUIVO .DZN 44
APÊNDICE D – MODELAGEM PALETIZAÇÃO SIMPLES 45

1 INTRODUÇÃO

Problemas de otimização são desafios intrínsecos à busca pela melhor solução em uma ampla gama de contextos. Esses problemas envolvem a seleção da melhor alternativa, seja ela a mais eficiente, econômica, ou aquela que maximiza ou minimiza uma determinada função objetivo. A complexidade reside na necessidade de explorar um espaço de soluções vasto.

Embora o estudo desses artefatos possa, por si somente, sustentar interesse pela área, as aplicações práticas em problemas de agendamento e logística ampliam ainda mais o ímpeto, especialmente se atreladas a uma necessidade pedagógica. Este é o caso do Programa Nacional do Livro e Material Didático (PNLD), responsável pela disponibilização e gerenciamento de livros didáticos ao longo de todo território Brasileiro (Brasil, 2017).

O desenvolvimento de modelos computacionais capazes de encontrar soluções satisfatórias para os diversos problemas de otimização exige que testes sejam realizados no intuito de se validar a eficiência das soluções encontradas. A utilização de instâncias reais do problema a ser modelado pode não ser prática, num primeiro momento, especialmente por razões do tamanho da entrada, que pode afetar diretamente no tempo de execução do modelo de forma não desejável. A geração de instâncias, portanto, passa a ser um problema próprio já que é necessário que os responsáveis pelos testes sejam capazes de identificar rapidamente ou ter o conhecimento prévio das soluções que otimizam a instância.

Essa necessidade se faz presente na problemática do PNLD. Enquanto um programa dividido em etapas de desenvolvimento, para fins de organização e administração, o PNLD é dividido em estágios que se iniciam na publicação do edital para aquisição dos materiais e se estendendo até o processo de logística e entrega deles (FNDE, 2021). Essa dependência entre etapas se manifesta nos processos iniciais, já referentes ao planejamento da logística dos materiais, que uma vez otimizados poderiam representar melhoria no processo como um todo.

1.1 JUSTIFICATIVA

O Brasil, enquanto país de proporções continentais, representa um desafio logístico para qualquer atividade que se proponha a ser desempenhada consistentemente ao longo de todos os seus estados e cidades. No tocante ao PNLD, 130 mil escolas são contempladas pelo programa, distribuindo 153 milhões de livros anualmente (FNDE, 2021). A distribuição gratuita de livros didáticos assume um papel fundamental na promoção da educação e igualdade de oportunidades no Brasil, dadas as vastas dimensões do país. A extensão territorial e a diversidade socioeconômica brasileira destacam a importância de um acesso equitativo a materiais educacionais.

No contexto brasileiro, marcado por desigualdades regionais e socioeconômicas, a distribuição gratuita de livros didáticos emerge como uma estratégia crucial para mitigar disparidades educacionais. O tamanho continental do Brasil impõe desafios logísticos significativos, tornando essencial a implementação de políticas que garantam que todos os estudantes, independentemente de sua localização geográfica, tenham acesso a recursos educacionais de qualidade.

A distribuição gratuita de livros didáticos desempenha um papel vital no fortalecimento do sistema educacional como um todo Além de contribuir para a equidade. Esses materiais não apenas apoiam o aprendizado em sala de aula, mas também servem como ferramentas que incentivam o desenvolvimento cognitivo e crítico dos estudantes.

Ainda, para a solução dessa problemática, é necessário o emprego de ferramentas capazes de representar e resolver problemas de otimização. Em particular, faz-se necessário o

emprego do paradigma Programação por Restrições, uma vez que ele é capaz de proporcionar uma transição direta entre uma modelagem matemática de um conjunto de variáveis e restrições para uma representação computacional. E é por conta disso que esse trabalho apresenta o largo emprego da ferramenta MiniZinc (Tack, 2014).

No contexto de problemas de otimização, as instâncias de teste são cruciais a avaliação do desempenho, eficácia e robustez dos algoritmos. Elas são instrumentais para identificar possíveis pontos fracos nos algoritmos de otimização. A exposição a uma gama variada de casos desafia os modelos a se adaptarem a condições inesperadas e aperfeiçoa sua capacidade de lidar com situações imprevisíveis.

1.2 OBJETIVOS

O principal objetivo desse trabalho é apresentar o Gerador de Instâncias aleatórias desenvolvido para fins de teste e melhoria de um modelo de otimização de paletização referente ao PNLD que representa um caso particular do Problema de Carregamento de Paletes (PCP). Algumas instâncias geradas pelo algoritmo serão testadas em um ambiente controlado no intuito de se avaliar a eficácia do gerador em criar problemas, as ferramentas empregadas para a execução do modelo e a modelagem propriamente dita. Além disso, uma modelagem própria, referente à Paletização Englobada, será proposta, também no intuito de incrementar o trabalho que já tem sido realizado no projeto.

1.2.1 Objetivos Específicos

- Apresentar o Gerador de Instâncias, bem como suas estratégias.
- Apresentar a modelagem da Paletização Englobada.
- Analisar a performance de entradas de diversos tamanhos geradas pelo Gerador de Instâncias.
- Comparar a diferença de performance entre o modelo fornecido de Paletização Simples e o de Paletização Englobada.

1.3 ESTRUTURA DO DOCUMENTO

No documento a seguir, informação é estruturada da seguinte forma:

- No Capítulo 2 figura a fundamentação teórica embasando o trabalho.
- O Capítulo 3 apresenta o PNLD e as modelagens matemáticas dos problemas de Paletização.
- O Capítulo 4 detalha o funcionamento do Gerador de Instâncias e as análises de performance dos modelos.
- No Capítulo 5 há a conclusão do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 PROBLEMAS DE SATISFAÇÃO DE RESTRIÇÕES

Segundo Norvig e Russel (Russel e Norvig, 2021), Problemas de Satisfação de Restrição (PSR) são problemas nos quais o objetivo é encontrar soluções adequadas a um conjunto de variáveis pré definidas, satisfazendo um conjunto de restrições impostas. Em suma, os PSR possuem três componentes principais, X, D e C sendo:

- X é um conjunto de variáveis $\{X_1, X_2, \ldots, X_n\}$.
- D é um conjunto de domínios $\{D_1, D_2, \ldots, D_n\}$, um para cada variável.
- C é um conjunto de restrições $\{C_1, C_2, \ldots, C_m\}$ que especificam as relações entre as variáveis.

No tocante às restrições, cada uma delas é composta por um par formado por <Escopo, Relação>, sendo "escopo"o conjunto de variáveis que participam da restrição, enquanto "relação"representa as valorações possíveis das variáveis definidas por "escopo". Portanto, resolver um PSR é atribuir um conjunto de valores às variáveis do conjunto X de forma em que seus valores estejam contidos nos domínios afixados, respeitando as restrições impostas pelo problema.

Partindo dessas ideias podemos estabelecer o conceito de uma Atribuição.

Definição 1 *Uma Atribuição, A de um PSR $P = (X, D, C)$ é definida pela tupla $A = (X_A, V_A)$ pela qual:*

- *$X_A \subseteq X$ é um subconjunto das variáveis $X_A = \{X_{A_1}, \ldots, X_{A_K}\}$;*
- *$V_A = (v_{A_1}, \ldots, v_{A_k}) \in \{D_{A_1} \times \cdots \times D_{A_k}\}$ é a tupla de valores escolhidos para as variáveis valoradas.*

Definição 2 *Dada $A = (X_A, V_A)$ uma atribuição, total ou parcial, de um PSR $P = (X, D, C)$ e $C_i = (X_i, R_i)$ uma restrição de P de forma em que $X_i \subseteq X_A$, a atribuição A satisfaz a restrição C_i se e somente se todos os valores $V_{A_i} = \{v_i \in V_A | x_i \in X_i\}$ das variáveis na restrição C_i pertençam a R_i.*

Assumindo, portanto, que valorações parciais são aquelas nas quais um subconjunto das variáveis é corretamente valorada e valorações totais são aquelas nas quais todas as variáveis em X são atribuidas com valores, podemos concluir que:

Definição 3 *A solução de um PSR é atribuição total que satisfaz todas as restrições do problema.*

Durante o processo de resolução dos PSR, as soluções possíveis são buscadas através das múltiplas possibilidades de valoração presente dentro dos domínios apresentados por cada conjunto de variáveis. Pela natureza combinatória das soluções, a escala do problema pode subir de maneira exponencial, uma vez que, a princípio, cada membro de cada domínio pode ser atribuido para as variáveis elegíveis. Na verdade, as possibilidades são bem mais restritas, uma vez que as restrições surgem, justamente, como uma limitação dentro das possibilidades existentes nos domínios. Logo, uma das formas de se reduzir o espaço de busca e torná-la mais

eficiente é utilizar as restrições do problema como base para a redução dos domínios, em um processo conhecido como Propagação de Restrição.

No entanto, nem sempre encontrar uma atribuição que satisfaça todas as restrições do problema é o suficiente. Um mesmo problema de restrições pode vir a ter múltiplas soluções válidas. Existe, portanto, o conceito de "solução ótima", isto é, uma solução que, além de satisfazer as restrições do problema, é a que melhor é avaliada segundo a *função objetivo*.

Definição 4 *A função objetivo em um problema de otimização é uma expressão que atribui um valor numérico a um conjunto de variáveis de decisão de forma a avaliar a desejabilidade das variáveis. A função objetivo é geralmente representada por* $f(x_1, x_2, \ldots, x_n)$*, onde* $x_1, x_2, \ldots, x_n$ *são as variáveis de decisão do problema.*

Em suma, a função objetivo providencia valores maiores se as soluções apresentarem comportamentos mais desejáveis e menores caso contrário. Por exemplo, em um problema que busca minimizar os valores atribuidos às variáveis do PSR, uma valoração V_A, cujas variáveis em conjunto possuem valores menores que uma dada valoração V_B, teria um valor de função objetivo maior do que V_B. Os PSR que levam, também, em consideração soluções ótimas são também denominados Problemas de Otimização Restrita (Russel e Norvig, 2021).

2.2 EXEMPLOS DE PSR

A partir das definições estabelecidas anteriormente, podemos representar uma sorte de problemáticas como PSR. Para fins didáticos, podemos citar o problema da coloração de mapas. Este problema é representado por um mapa regional dividido em vários territórios menores, sendo o objetivo encontrar uma coloração de cada região do mapa de forma a garantir que duas regiões vizinhas não possuam a mesma cor.

Figura 2.1: Problema de Coloração

Considerando, portanto, o mapa da Austrália (Russel e Norvig, 2021) como exemplo, é possível estabelecer as três componentes básicas de um PSR a partir de suas características. O conjunto de variáveis, X é o conjunto das regiões que compõem o mapa da Austrália,

$\{WA, NT, Q, NSW, V, SA, T\}$. Essas variáveis, no entanto, não podem receber quaisquer valores, apenas os presentes no domínio do problema. Define-se, então, o domínio, neste exemplo $D_i = \{vermelho, azul, verde\}$. Ainda, é necessária a definição das restrições, que restringirão o espaço de soluções do problema. Anteriormente, tais restrições foram definidas de maneira informal. Uma possível formalização dessas restrições seria representada por $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$.[1] Com base nessas características e nos conceitos anteriormente citados, uma possível solução para este problema seria $\{WA = azul, NT = vermelho, Q = azul, NSW = vermelho, V = azul, SA = verde, T = azul\}$.

Outro problema bastante conhecido que pode ser visto como tal é o Sudoku. Nele, precisamos preencher espaços quadriculados com números de 1 a 9, contudo precisamos seguir um conjunto de restrições.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figura 2.2: Exemplo de um Sudoku

Em uma dada linha não podemos repetir números dentre os permitidos no intervalo. O mesmo vale para colunas e para os quadrados menores, 3x3, no interior da matriz principal. Portanto, podemos, a partir dessa definição informal do problema, estabelecer um modelo que represente a questão por meio de um PSR.

- X é representado pelas casas da matriz, sendo o intervalo $\{x_{11}, x_{12}, \ldots, x_{99}\}$.
- D é representado pelos valores possíveis $\{1, 2, \ldots, 9\}$, sendo necessário apenas um para todas as variáveis.

[1] Foi omitida parte da representação das restrições, já que, formalmente, uma restrição foi definida como um par <Escopo, Relação>, sendo escopo o conjunto de variáveis que participam da restrição. Por exemplo, a restrição $SA \neq WA$ representa a restrição formal $(\{SA, WA\}, \{(c_i, c_j) | c_i, c_j \in D, c_i \neq c_j\}), D = \{verde, vermelho, azul\}$.

- C é formado por três restrições: $Alldiff(\{x_{i1}, x_{i2}, \ldots, x_{i9}\})\forall i \in [1,9]$, $Alldiff(\{x_{1j}, x_{2j}, \ldots, x_{9j}\})\forall j \in [1,9]$ e $Alldiff(x_{ij})$ para todo x_{ij} [2] pertencente ao mesmo subquadrante.

Ainda nesse tópico, um outro exemplo de PSR é um problema conhecido como o Enigma de Einstein. Nele, somos convidados a desvendar os mistérios referentes a cinco homens de distintas nacionalidades, morando em uma mesma vizinhança. Nos são fornecidas informações referentes a locais onde moram, bebidas, hobbies, cores e animais de estimação de preferência de cada um. Contudo, essas informações são fornecidas, em sua maioria, de forma relativa nos restando à interpretação do problema que deve ser resolvido por meio de uma busca combinatória, encaixando cada opção em uma configuração que respeite as restrições impostas.

Portanto, é plenamente possível definir este problema através da representação de PSR, com um conjunto de variáveis X, um conjunto de domínios D e um conjunto de restrições C. As variáveis representam os espaços possíveis que podem ser valorados. Porém, diferentemente do problema do Sudoku, aqui as variáveis são representadas por domínios diferentes, pois aceitam e representam aspectos diferentes do Enigma de Einstein. Para identificarmos cada indivíduo, possuímos informações sobre Nacionalidade, Bebida, Cor da Casa, Animal de Estimação e Hobby, além da posição da casa na vizinhança. Logo, é de se imaginar que nos cinco campos possíveis de Nacionalidade, apenas gentílicos são aceitos, assim como no campo de cor, apenas cores fazem parte do domínio. Este problema possui diferentes conjuntos de variáveis e, para cada um deles, um domínio possível. E, por final, cada valoração adequada deve respeitar um conjunto de restrições que relacionam as variáveis e suas possibilidades.

É importante ressaltar, contudo, que encontrar as valorações que geram soluções para PSR requer a utilização de técnicas específicas de busca. Os problemas que podem ser expressados em forma de PSR em sua maioria são desafios que não apresentam solução algébrica ou facilmente expressável por funções numéricas. Na verdade, grande parte dos PSR requerem uma busca combinatória, na qual cada possível valoração deve ser testada para todo o conjunto de variáveis até que uma solução satisfaça as restrições. As técnicas empregadas na busca pelas soluções serão apresentadas, portanto, a seguir.

2.3 BUSCANDO SOLUÇÕES PARA OS PSR

Existem várias prismas pelos quais podemos analisar um PSR, quando nosso intuito é decidir a melhor forma pela qual podemos encontrar soluções apropriadas. Assim como podemos analisar as características das variáveis, com enfoque em seus domínios, podemos analisar as restrições, em particular como elas influenciam as interações entre as potenciais diversas variáveis existentes em um dado problema. Uma restrição é unária se limitar exclusivamente uma única variável do sistema (Russel e Norvig, 2021). Tendo em mente o exemplo da coloração de mapas, poderíamos ter uma restrição que definiria que, pelos habitantes da região de South Australia não gostarem da cor vermelha, não poderíamos selecionar esta cor para uma solução válida do problema. Uma restrição pode ser, também, uma restrição binária, caso venha a relacionar duas variáveis. Partindo desses conceitos, um PSR é binário se todas as suas restrições forem binárias ou unárias.

Um caso geral de restrições em termos de quantidade de variáveis relacionadas é a restrição global. Ao contrário do que a expressão sugere, uma restrição global não relaciona todas

[2] Alldiff é uma restrição global e formalizada na qual, dado um conjunto de valores, todos devem ser diferentes entre si; Assim como no caso das restrições de desigualdade do problema da coloração de mapas, uma restrição *alldiff* pode ser expressa como uma tupla de valores permitidos.

as variáveis do sistema. Na verdade, uma restrição global relaciona uma quantidade arbitrária de variáveis. Um exemplo de restrição global é a *Alldiff* que, dado um conjunto arbitrário de variáveis restringidas por ela, estipula que os valores atribuidos a cada uma delas sejam diferentes entre si, par a par.

Para encontrar uma solução válida em um PSR é necessário, muitas vezes, iterar sobre um conjunto diverso de possibilidades, uma vez que, em vários problemas, não existe um algoritmo eficiente e exato capaz de computar a solução. A solução é, portanto, buscada em um processo de tentativa e erro a partir de atribuições de valores do domínio a cada uma das variáveis existentes, no intuito de satisfazer as restrições impostas no problema. É por isso que existem técnicas para reduzir o espaço de busca das soluções e diminuir o tempo necessário para solucionar os PSR. Uma delas é a consistência local e seus diferentes tipos.

A consistência local ocorre quando há a remoção de opções inválidas dos domínios de cada variável, de forma a satisfazer as restrições e, ao mesmo tempo, garantir uma redução das possibilidades de busca de solução. De forma mais particular, consistência de nodo ocorre quando, para uma dada variável, todos os valores presentes em seu domínio satisfazem as restrições unárias impostas. Portanto, no exemplo de South Australia, a variável *SA* seria nodo-consistente se em seu domínio não figurasse a cor vermelha, já que uma das restrições exigem que a variável não pode ter atribuição $vermelho$ dado a ela.

Além da consistência de nodo, a consistência de arco é, também, uma propriedade capaz de agilizar a busca de soluções. Ela ocorre quando todos os valores do domínio de uma dada variável satisfazem suas restrições binárias (Russel e Norvig, 2021). Formalmente, uma variável X_i é arco-consistente em relação a outra variável X_j se para todos os valores no domínio D_i existe algum valor no domínio D_j que satisfaça as restrições existentes entre as variáveis X_i e X_j. Por exemplo, se definirmos que uma restrição entre duas variáveis X e Y é representada por $Y = X^3$, explicitamente a restrição poderia ser representada como

$< (X, Y), \{(0, 0), (1, 1), (2, 8), (3, 27)\} >$.

No exemplo apresentado, as variáveis X e Y serão arco-consistentes entre si se seus domínios forem representados como $D_X = \{0, 1, 2, 3\}$ e $D_Y = \{0, 1, 8, 27\}$. Em termos de computabilidade, garantir que um dado PSR se torne arco-consistente, isto é, que todas as variáveis sejam arco-consistentes entre si, pode reduzir em muito o tempo necessário para que uma solução, e por consequência uma solução ótima, seja encontrada. Ainda, se um dado PSR já não for arco-consistente, uma etapa de pré-processamento que o converta em arco-consistente pode, inclusive, entregar "gratuitamente"uma solução.

Contudo, não é incomum que a redução dos domínios, na busca pela consistência das variáveis, não seja capaz de resolver por conta o problema em toda sua complexidade. Quando isso ocorre, é necessária uma iteração efetiva das possíveis valorações das variáveis existentes no problema. Para esses casos, uma forma adequada de se enfrentar o problema faz uso do *backtracking*. Esta, por sua vez, representa uma característica de algoritmos que o tornam semelhantes a técnicas de resolução de problemas empregadas por humanos. De forma geral, algoritmos de backtracking armazenam em memória os estados a cada etapa de execução de uma busca. Cada estado possui um conjunto de estados sucessores, alcançáveis mediante ações específicas. Em um algoritmo de backtracking, são selecionadas opções dentro das possibilidades até que as combinações selecionadas não satisfaçam o problema, ou uma solução seja encontrada. Caso uma validação incorreta do problema seja detectada, o algoritmo retorna até uma etapa anterior e tenta, novamente, com outro valor possível.

Tomando o PSR de coloração de mapas como exemplo, cada estado da busca algorítmica pode ser representado pela coloração de um conjunto de territórios. Ainda, cada ação pode ser representada pela escolha e coloração de um dos territórios de uma das cores disponíveis no

domínio dele. De modo informal, a cada etapa de execução do algoritmo uma cor é elegida para colorir um dos territórios e se o novo status do mapa for uma solução para o problema, a execução é encerrada. Se, no entanto, o novo status representar uma violação das restrições, a última seleção é desfeita e uma nova é tomada no lugar, até que ou uma solução seja encontrada ou todas as possibilidades sejam exauridas, levando, então, ao retorno de uma falha por parte do algoritmo.

Ainda no contexto de coloração de mapas, na Figura 2.3 podemos observar de forma clara a execução de um algoritmo de backtrack. Na árvore representada, o estado raiz é representado pelo mapa sem cores, a partir do qual opções serão selecionadas e uma solução, buscada. As ações sucessoras do estado raiz base são representadas pela coloração do território mais a esquerda, Western Australia(WA), do mapa australiano. Sendo três cores disponíveis no problema, existem três estados sucessores. A seguir, o algoritmo seleciona um dos estados sucessores, geralmente em navegação *Inorder*[3], no exemplo representado pela ação de colorir WA de vermelho. A partir desse estado, geramos outros sucessores, no exemplo representados pela coloração de Northern Territory (NT) de verde ou azul. O algoritmo segue esse comportamento até alcançar um estado no qual encontre uma coloração completa que não viole as restrições. No entanto, caso algum dos estados represente uma violação, o algoritmo retorna até seu nodo pai e tenta outras ações a partir dele. Se todas as ações disponíveis forem exploradas e nenhuma solução encontrada, o algoritmo retorna falha.

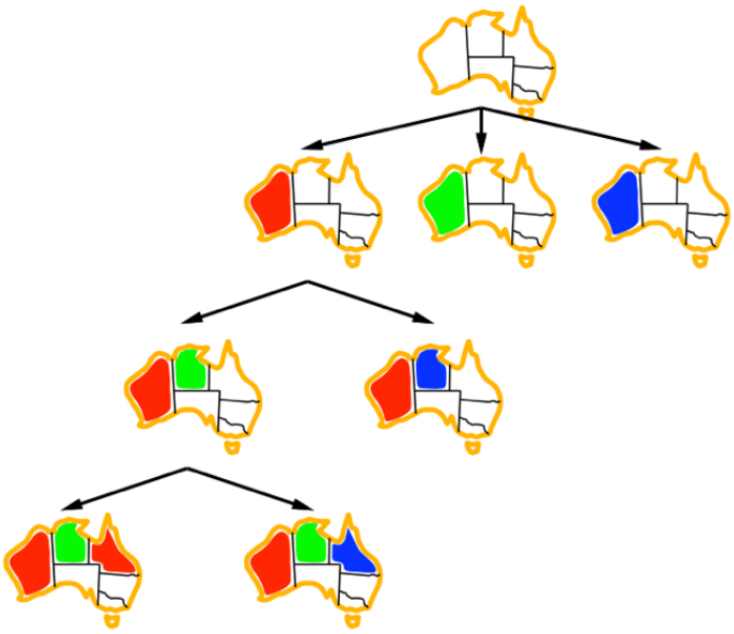


Figura 2.3: Backtracking

Dada a natureza exaustiva da busca em backtracking, reduzir possibilidades de ações reduz, também, o tempo máximo de execução do algoritmo. Portanto, ao se garantir consistência de arco nas variáveis, garante-se, também, uma menor quantidade de ações possíveis. Versões mais eficazes de algoritmos de Backtracking também são capazes de realizar uma manutenção da consistência de arco de forma dinâmica, isto é, alterar os domínios das variáveis à medida que opções anteriores forem escolhidas. Essa é, inclusive, a forma mais rudimentar possível de um mecanismo de inferência, que "imita", de certa forma, uma intuição humana na busca de uma solução para o problema.

[3]Uma das formas de se navegar uma árvore binária, seguindo a folha mais à esquerda possível, nodo pai, folha irmã à direita

2.4 PROGRAMAÇÃO POR RESTRIÇÕES

Uma outra forma de lidarmos com o problema, na busca pela solução, pode ser o uso de um paradigma de programação: A Programação por Restrições. Segundo Roman Barták (Barták, 1998), a Programação por Restrições consiste no emprego de tecnologia emergente de software para descrição declarativa e solução efetiva de problemas grandes e, particularmente combinatórios, com enfoque nas áreas de planejamento e agendamento. Esse paradigma nos permite observar os PSR através de outras lentes, que nos fornecem mais ferramentas para manipular estes problemas. De uma forma mais aprofundada, podemos empregar as técnicas já conhecidas de solução de PSR, como a Propagação de Restrições, de uma forma mais robusta em uma plataforma de programação otimizada para esse tipo de operação.

Na Programação por Restrições, os usuários instanciam de forma declarativa as restrições das variáveis de decisão. Ao contrário de linguagens imperativas, as restrições não ditarão uma ordem ou sequência de ações executadas, mas, sim, propriedades de uma solução adequada. Além disso, durante a modelagem das soluções, o usuário deverá explicitar um método pelo qual a solução deve ser buscada, como, por exemplo, a já citada Propagação de Restrições, ou alguma heurística customizada para a ramificação do problema.

Como apresentado previamente, o Paradigma de Programação por Restrições se faz presente pelo emprego de tecnologias próprias para a solução dos PSR, associadas a técnicas tradicionais de solução de problemas desse tipo. Uma dessas tecnologias é representada pela plataforma Minizinc (Tack, 2014). O Minizinc é ao mesmo tempo uma plataforma de software e uma linguagem de código aberto para otimização de problemas, capaz de rodar modelos de otimização por meio de solvers executando em *backend*. Por ele podemos realizar a criação de modelos matemáticos que representem os PSR a serem resolvidos, contudo o modelo não dita, necessariamente, o método pelo qual o problema será solucionado. Na verdade, o próprio software se encarrega da seleção do método mais adequado para solução. Uma das funcionalidades do sistema é a possibilidade de uma modelagem muito próxima da sintaxe matemática, que antecede a formulação da solução computacional do problema.

Para que possamos exemplificar o uso da linguagem, assim como apresentar uma modelagem possível de um problema, tomemos novamente o supracitado problema do Enigma de Einstein. As restrições do problema são referentes às informações de cada morador das casas, que indicam de forma relativa as possibilidades de valores de cada variável. O objetivo final aqui é descobrir quem, dos cinco indivíduos apresentados, possui como animal de estimação um peixe.

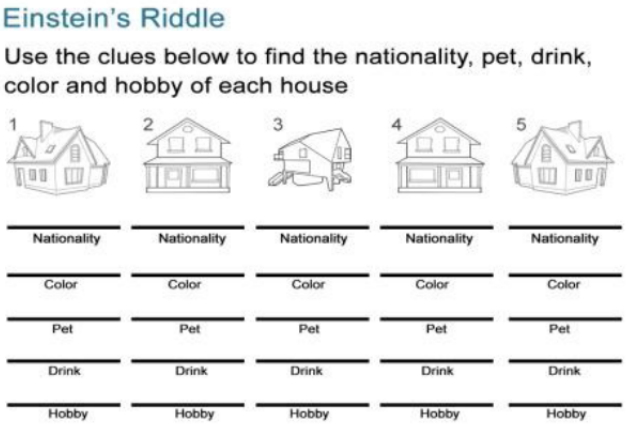


Figura 2.4: Enigma de Einstein

Problema 1 *Cada pista apresentada relaciona os moradores com a nacionalidade, animal de estimação, bebida e cor de casa.*

1. *O britânico vive na casa vermelha.*
2. *O sueco tem cachorros como animais de estimação.*
3. *O dinamarquês bebe chá.*
4. *O alemão joga futebol.*
5. *O dono da casa verde bebe café.*
6. *O morador que joga baseball tem pássaros.*
7. *O morador que joga basquete vive ao lado do que tem gatos.*
8. *O norueguês vive na primeira casa.*
9. *O morador que toca violino bebe cerveja.*
10. *O morador da casa amarela toca piano.*
11. *Aquele que vive na casa do centro bebe leite.*
12. *O que tem um cavalo vive ao lado do que toca piano.*
13. *A casa verde está imediatamente à esquerda da casa branca.*
14. *O norueguês vive ao lado da casa azul.*
15. *Aquele que joga basquete vive ao lado do que bebe água.*

Quem tem um peixe como animal de estimação?

A partir dessas restrições, podemos usar linguagem matemática e lógica para estabelecer os fatos propostos sobre os moradores. Ainda, podemos nos aproveitar da capacidade do MiniZinc de criar tipos enumerados para que a lógica do modelo se torne mais legível para olhos humanos. Isso pode ser observado no Anexo A de uma modelagem possível para a solução do problema. Cada vetor de elementos que representa os valores de hobby, nacionalidade, cor, pet e bebida é indexado a partir de um conjunto de tipos enumerados para que a leitura seja facilitada. Dessa forma, sempre que nos referirmos à cor vermelha, poderemos evocar Colors[red] para nos referirmos à opção, como visível na linha 9 da modelagem. É importante ressaltar que, ao contrário de outras linguagens e paradigmas de programação, a programação por restrições possui sutilezas próprias, por exemplo, a possibilidade de declaração de variáveis ou constantes após a sua utilização no código.

Em seguida, na linha 41, declaramos as variáveis c, n, p, d e h, que representam os campos em branco do problema e serão valoradas ao longo da execução do algoritmo que encontra a soluçao. Cada uma dessas variáveis é, também, um vetor cujos índices variam de 1 a 5. Os índices nesses vetores indicam a posição da casa na vizinhança, de forma a representar computacionalmente cada uma das casas. Essa abstração nos permite converter as restrições posicionais em restrições numéricas. Por exemplo, quando nos referirmos a uma casa estar à esquerda de outra, agora estamos nos referindo a uma casa cujo o índice é um menor do que outra.

Logo, em nosso modelo, uma vez definidas as variáveis e seus respectivos domínios, chega o momento no qual devemos declarar as restrições do problema, utilizando as funções e relações lógicas entre cada uma das valorações possíveis, como visível a partir da linha 48. Assim, primeiramente devemos nos certificar de que cada uma das variáveis receba um valor único, já que cada campo possui seu próprio valor dentro dos domínios permitidos. É por conta dessa restrição que a inclusão do arquivo "alldifferent.mzn"se faz necessária na modelagem, uma vez que possui a implementação da restrição global *Alldiff*, que assegura que todos os elementos de um dado conjunto fornecido como entrada são diferentes entre si. Além dessa restrição base, devemos nos certificar que as demais restrições que indicam as relações entre os valores admitidos no problema sejam representadas. Ao rodarmos esse modelo, podemos exibir o resultado encontrado das valorações das variáveis.

Utilizando a notação numérica do modelo e considerando as posições relativas das variáveis como as mesmas da imagem original, aquele que possui o peixe é o Alemão, que vive na quarta casa. Embora exista espaço para uma melhor modelagem do problema, em termos de eficiência e de legibilidade, fica nítido que o MiniZinc enquanto ferramenta possui enorme potencial para solução de problemas de restrição e é com base nessa plataforma que será proposta uma possível modelagem de um problema de grande importância - Um problema de distribuição de Livros Didáticos pelo território brasileiro.

2.4.1 Solvers no MiniZinc

Os solvers para o Minizinc são responsáveis por receber os modelos Minizinc como entrada, interpretá-los, e então aplicar métodos e técnicas específicas para encontrar soluções que atendam aos critérios do problema modelado. Eles podem ser especializados em diferentes tipos de problemas, como programação linear, programação inteira, programação não linear, entre outros. Dentre os solvers disponíveis no MiniZinc, os que serão avaliados a posteriori são:

- Geocode: Um sistema de programação por restrições de código aberto . Ele é um solver que oferece suporte nativo a muitas das restrições globais do MiniZinc e possui capacidade para lidar com variáveis inteiras, de conjunto e de ponto flutuante. (Tack, 2014)

- Chuffed: um resolvedor de restrições baseado na geração de cláusulas preguiçosas. Esse tipo de resolvedor adapta técnicas provenientes da resolução de SAT, como aprendizado de cláusulas de conflito, propagação de literais monitorados e heurísticas de busca baseadas em atividade, e frequentemente pode ser significativamente mais rápido do que resolvedores tradicionais de programação por restrições (CP). (Tack, 2014)

- Coin-BC:um projeto de código aberto dedicado ao desenvolvimento de software para pesquisa operacional. O COIN-BC é projetado para resolver problemas de programação linear inteira (PLI) por meio do método Branch-and-Cut.[4]

- HiGHs: um software livre de alto desempenho para resolver modelos de programação linear (LP), programação linear inteira (MIP) e programação quadrática (QP) em larga escala. Ele foi desenvolvido em C++11 e é projetado para operar eficientemente tanto em ambientes de computação serial quanto paralela. Além disso, o HiGHS oferece

[4]A técnica Branch-and-Cut é uma abordagem utilizada em problemas de programação linear inteira para combinar eficientemente a busca exaustiva com a eliminação de soluções viáveis ineficazes. Ela envolve a divisão do problema em subproblemas menores, ou "branching", e a aplicação de cortes para reduzir o espaço de busca. (Forrest et al., 2019).

interfaces para as linguagens de programação C, C#, FORTRAN, Julia e Python. (Huangfu e Hall, 2018)

2.5 PROBLEMA DE CARREGAMENTO DE PALETES E DE ROTEAMENTO DE VEÍCULOS

Outros exemplos de problemas de otimização que serão relevantes ao longo deste estudo são os problemas de Carregamento de Paletes (PCP) e o Problema de Roteamento de Veículos (PRV). O primeiro deles, PCP, está relacionado com a busca de uma conformação ótima de objetos organizados em um palete, de forma a respeitar aspectos de estabilidade e peso. Este, por sua vez, pode ser subdividido em dois outros tipos de problemas PCP: O do Produtor, no qual todos os objetos a serem empilhados possuem o mesmo tamanho e formato, e o do Distribuidor, no qual há variabilidade nas cargas a serem empilhadas (Derenievicz et al., 2023).

Ainda, o PRV diz respeito ao problema de se encontrar rotas otimizadas de uma dada frota de veículos no intuito a concretizar demandas de múltiplos clientes. Assim como o PCP, o PRV pode ser representado por diferentes tipos de problema, como, por exemplo, o problema PRV com Coleta e Entrega e PRV com Janelas de Tempo. Em ambos os casos algoritmos exatos e heurísticos podem ser empregados no objetivo de se encontrar uma solução.

2.6 REVISÃO BIBLIOGRÁFICA

Ao longo das etapas iniciais desta pesquisa foram consultados materiais diversos referentes a àrea de programação por restrições e aos PSR. Em particular, dois materiais foram vistos com mais profundidade: A documentação Referente ao MiniZinc e, também, a pesquisa referente ao CABSC (Coulombe e Quimper, 2022).

O CABSC (Constraint Acquisition Based on Solution Counting) é um projeto de Coulombe e Quimper, cujo objetivo é desenvolver modelos de PSR indiretamente através de um algoritmo de branch and bound que explora possíveis modelos finais a partir de um conjunto de modelos fornecidos como exemplos na entrada. O software resolve um Meta-PSR, isto é, um PSR cuja solução segundo as restrições será um PSR, sendo as variáveis valoradas com parâmetros do problema resultante.

Para explorar essa solução, os criadores desenvolveram uma ferramenta que denominaram "MiniZinc Aprimorado"(Augmented MiniZinc), que incorporou as características do problema que denominaram Meta-CSP (Ou Meta-PSR). Além dos componentes tradicionais de um PSR, o Meta ainda incorpora uma variável de ativação, que define quais restrições serão consideradas pelo CSP otimizado ao final, além das entradas para as variáveis de parâmetro, que farão parte do CSP resultado, e uma entrada para uma matriz de soluções. Essa matriz de soluções, E armazena os resultados que resolvem o CSP resultante como uma solução possível. Ao longo das iterações do CABSC, múltiplos CSP serão produzidos com diferentes parâmetros. Ao final do algoritmo, será escolhido como solução o CSP que, ao mesmo tempo, aceitar todas as soluções da matriz E como resultado e que aceite a menor quantidade de soluções dentre os CSP gerados até então. Daí a principal característica do projeto: Contagem de soluções. A cada nova iteração do algoritmo as soluções aceitas pelo CSP da etapa são contadas; ao final o intuito é minimizar esse valor, ainda respeitando as restrições de aceitação das soluções da matriz E.

A principal aplicação de um projeto como o CABSC é a possibilidade de criação de modelos de otimização por pessoas que não necessariamente possuam expertise computacional ou matemática para a implementação de um modelo PSR. Munido de uma matriz composta por algumas soluções viáveis previamente conhecidas, um usuário poderia rodar o Meta-CSP e

obter um modelo capaz de organizar e otimizar algum problema de restrição de interesse. Um exemplo citado pelos próprios autores foi a organização de uma escala de enfermeiras de um hospital. Contudo, essa possibilidade traz consigo vários desafios. Um deles, por exemplo, seria a necessidade da criação de uma matriz com as soluções possíveis, o que implicaria na execução prévia de algum modelo de otimização ou planejamento, ou mesmo na organização manual dos dados. Isso significa que, na melhor das hipóteses, apenas deslocaríamos o custo intelectual da criação do modelo para uma etapa anterior e, na pior delas, poderíamos despender um esforço manual que possivelmente inviabilizaria o emprego dessa ferramenta.

Esse projeto embora de muito interesse e promissor, acabou por não ser escolhido como base para o desenvolvimento desse trabalho, mas serviu como fonte de estudo para exemplo das múltiplas capacidades de solução que os problemas PSR podem fornecer. Além desse, o problema de distribuição de livros (Brasil, 2017), bem como uma modelagem de solução no MiniZinc para o problema de paletização, disponível em (Derenievicz et al., 2023), foram o pontapé para a decisão da direção geral das próximas etapas, tanto no aspecto teórico quanto prático.

3 APRESENTAÇÃO DA PROBLEMÁTICA

3.1 PROGRAMA NACIONAL DO LIVRO E DO MATERIAL DIDÁTICO - PNLD

No Brasil, atualmente, os livros didáticos a serem utilizados pelas escolas públicas são requisitados mediante a necessidade de cada rede de ensino. Assim, não existe um padrão exato de livros a ser seguido igualmente por todas as instituições. Isso significa que é necessário a requisição periódica de novos livros didáticos e sua distribuição é variada, dependendo de fatores logísticos referentes às editoras e distribuidoras do material. Na verdade, a logística representa um problema de vários níveis diferentes, que se combinam em um macro-problema de distribuição.

Em princípio a distribuição dos livros depende da distância entre os produtores e distribuidoras responsáveis, o que é um grande desafio, em decorrência das proporções continentais do território brasileiro. Ainda, é necessária uma logística capaz de organizar adequadamente as demandas em consonância com as distribuidoras, de forma a abastecer a maior quantidade de escolas possíveis para cada região. No entanto, este problema vai além, pois será necessário que esses livros sejam organizados fisicamente, não apenas dentro dos caminhões, para que caibam adequadamente, como também em termos de ordem de entrega e capacidade dos paletes que sustentarão as entregas. Por fim, o roteamento desses caminhões de entrega deve levar em consideração essa organização de livros, de forma a entregar os livros da maneira mais eficiente possível em termos de tempo e custo de combustível.

O Programa Nacional do Livro e do Material Didático (Brasil, 2017) é executado pelo Fundo Nacional de Desenvolvimento da Educação-FNDE com o objetivo de garantir aos estudantes das escolas públicas brasileiras o direito constitucional ao material didático (CF Art. 208, inciso VII). Para participar de tal Programa as redes de ensinos devem manifestar seu interesse por meio de uma adesão prévia. Todas as redes de ensino estaduais, grande parte das escolas da rede federal e a grande maioria das redes municipais de educação estão aderidas hoje ao PNLD. Assim, a cada edição, são beneficiados mais de 29 milhões de estudantes de aproximadamente 120 mil escolas. Para que este objetivo seja concretizado, são estabelecidas as seguintes etapas:

- Paletização virtual
- Postagem (Coleta e recebimento)
- Transporte (contratações de veículos e frota própria)
- Armazenamento (Otimização da carga em centros de distribuição)
- Monitoramento
- Entrega (Domiciliar e com Aviso de Recebimento/AR).

Esse problema de distribuição pode ser decomposto em diversos problemas, em especial Problemas de Satisfação de Restrições, individualmente complexos e de interesse. Em princípio, podemos destacar a etapa da Paletização Virtual, na qual há a organização, mediante algoritmo, das encomendas de cada escola, levando em consideração aspectos referentes à localização geográfica das editoras, assim como das centralizadoras responsáveis pelas distribuições dos

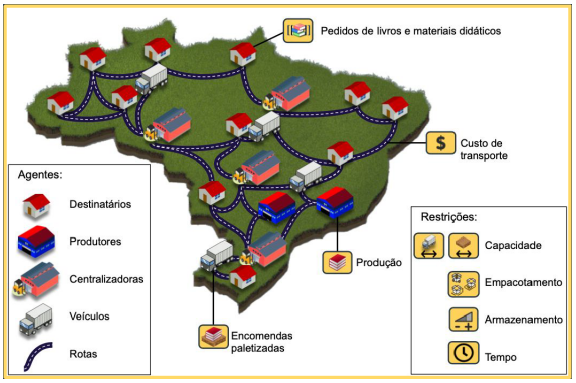


Figura 3.1: Esquema Visual do Problema. Imagens do Estudo Preliminar.

livros. No tocante à Paletização Virtual, além do planejamento dos destinos, centralizadoras e editoras, aspectos referentes à capacidade física de cada palete também interfere no processo. Ainda, a organização geométrica dessas encomendas também deve ser levada em conta, uma vez que desequilíbrios nos paletes podem vir a danificar as cargas, ocasionando prejuízos financeiros às instituições competentes, além da possibilidade do atraso na entrega do material didático.

A etapa de Paletização Virtual, determina a organização dos paletes a serem entregados, tarefa que usa como base para sua concretização os destinatários de cada encomenda a ser embalada no palete, suas respectivas produtoras e centralizadoras, no intuito de que se otimize a roteirização de entregas e que haja a maior redução possível de custos financeiros e temporais. Uma vez realizada essa etapa, a Postagem realizada a seguir é a etapa na qual as empresas, aqui discriminadas como produtoras, embalarão seus itens, montarão suas encomendas e enviarão os paletes montados para as centralizadoras, que armazenarão os paletes recebidos e serão responsáveis pela reorganização dos paletes e, em seguida, encaminhá-los para seus respectivos destinatários.

Todo o processo de organização de encomendas, produção de itens e entregas aos destinatários possui uma forte interação desde os princípios do processo até a entrega final para cada destinatário. Portanto, a otimização inicial realizada durante o processo de paletização virtual tem impactos ao longo de toda cadeia de processos da entrega, já que paletes mais otimizados requerirão menos recursos para a distribuição na fase de postagem e, em seguida, na fase de entregas. Com isso em mente, modelagens com o enfoque na etapa inicial de paletização foram implementadas, de forma a viabilizar uma análise posterior das demais etapas de otimização, uma vez que os modelos e práticas de paletização virtual realizados atualmente no processo não estão disponíveis para análise.

3.2 DEFINIÇÕES DO PROBLEMA E MODELAGENS

No intuito de produzir soluções para a problemática anteriormente citada, modelagens referente aos processos de logística de organização e entrega dos livros didáticos foram produzidas (Derenievicz et al., 2023). Cada etapa presente na distribuição do PNLD foi abstraída em um conjunto de problemas de otimização interconectados tanto para fins de melhor compreensão dos detalhes do problema geral quanto para que cada fragmento do macro problema pudesse ser visualizado individualmente. Neste problema, é considerado $I = \{1, \ldots, |I|\}$ o conjunto de itens que categorizam as encomendas e o conjunto $D = \{1, \ldots, |D|\}$ representa o conjunto de destinatários. O conjunto E representa o conjunto de encomendas necessárias para se concretizar as demandas. O conjunto E é ainda fracionado em conjuntos $S_1, \ldots, S_{|I|}$, sendo

cada S_i representante de encomendas do i-ésimo item. Ainda, cada produtor q possui um limite máximo de produção como restrição, que pode ser representado pela função $qtd(q, i)$.

Como citado anteriormente, a otimização da etapa referente a Paletização Virtual influencia nas demais etapas da distribuição. No que diz respeito as representações apresentadas previamente, uma paletização $\mathcal{P}$ é uma função definida para $P \to 2^E$ que relaciona o conjunto de paletes P aos subconjuntos possíveis de encomendas, sendo uma paletização válida apenas se satisfizer as restrições. Em suma buscamos a minimização da quantidade total de paletes utilizados. Para concretizar esse objetivo, foram criadas modelagens referentes aos processos de Paletização. Além dos conjuntos previamente citados, outros dados de entrada se fazem necessários para a realização do modelo. Além dos conjuntos I e E, o conjunto $C = \{1, \ldots, |C|\}$ representa as centralizadoras enquanto $D = \{1, \ldots, |D|\}$ representa os destinatários. Além disso, o conjunto $Q = \{1, \ldots, |Q|\}$ representa o conjunto dos produtores que, assim como todos os outros conjuntos anteriores, são indexados por valores únicos.

Ainda, cada encomenda presente no conjunto E possui associada um destinatário, $dest : E \longrightarrow D$, peso $peso : E \to \mathbb{N}$, além das suas dimensões $dim : E \to \mathbb{N}^3$; cada destinatário possui uma centralizadora $cent : D \to C$. A partir desses parâmetros, o algoritmo de otimização deve minimizar a quantidade de paletes utilizados na paletização. Dessa forma, as seguintes restrições devem ser satisfeitas para que uma solução seja válida:

$$\forall p_1, p_2 \in P: \quad p_1 \neq p_2 \implies \mathcal{P}(p_1) \cap \mathcal{P}(p_2) = \emptyset \tag{3.1}$$

$$\bigcup_{p \in P} \mathcal{P}(p) = E \tag{3.2}$$

$$\forall p \in P: \quad C(p) = cent(dest(e)), \qquad \forall e \in \mathcal{P}(p) \tag{3.3}$$

$$\forall q \in Q, \forall i \in I: \quad \sum_{p \in P, Q(p) = q} |\mathcal{P}(p) \cap S_i| \leq qtd(q, i) \cdot |T| \tag{3.4}$$

$$\forall p \in P: \quad \mathcal{P}(p) \neq \emptyset, C(p) \neq \emptyset, Q(p) \neq \emptyset, \mathcal{P}(p) \in \Pi \tag{3.5}$$

A partir dessas restrições, é possível observar que 3.1 e 3.2 garantem que não pode haver mais de um palete com uma mesma encomenda e todas as encomendas estão acomodadas em paltes. Ainda, a restrição 3.3 garante que todas as encomendas pertencentes a um mesmo palete devem compartilhar a mesma centralizadora, sendo $C(p)$ representativo da centralizadora responsável pelo palete p. A restrição 3.4 estabelece que a produção de paletes de um dado produtor qualquer não pode exceder a capacidade total de produção de cada item, sendo $qtd(q, i)$ a quantidade máxima de exemplares do item i produzíveis por q a cada unidade de tempo e $|T|$ a quantidade total de unidades de tempo. Finalmente, a restrição 3.5 garante que a produção dos paletes deve respeitar as restrições físicas de peso e tamanho, sendo o conjunto Π representante de todos os subconjuntos das encomendas acomodáveis nos paletes. O conjunto Π pode ser tão rigoroso quanto a modelagem exigir, no entanto. Dependendo da modelagem, pode representar restrições referentes ao PCP, citado no capítulo anterior e fundamental para a análise do atual problema; dependendo da modelagem, pode representar a restrição de peso máximo do palete, abordagem atual escolhida para fins de simplicidade e análise de outros aspectos do modelo.

Além dos paletes comuns, produzidos durante a etapa de paletização virtual, os paletes englobados também prepresentam outra camada do problema de logística enfrentado. A partir dessa abordagem, é definido *palete hospedado* aquele que abriga encomendas de um mesmo item, destinados a uma mesma centralizadora. Ainda, *palete hospedeiro* é aquele capaz de abrigar paletes de quaisquer itens destinados a uma mesma centralizadora. Com essa nova

abstração, operações logísticas são facilitadas, assegurando que uma vez entregues os paletes, as centralizadoras fiquem responsáveis apenas pela logística das encomendas até os destinatários. Contundo, uma nova camada de abstração requer uma nova restrição, capaz de adaptar a modelagem apresentada anteriormente às necessidades da modelagem do palete englobado. Essa e as outras figuram a seguir:

$$\forall p \in P, \exists i \in I : \qquad \mathcal{P}(p) \subseteq S_i \tag{3.6}$$

$$\forall p'_1, p'_2 \in P_e : \qquad p'_1 \neq p'_2 \implies \mathcal{E}(p'_1) \cap \mathcal{E}(p'_2) = \emptyset \tag{3.7}$$

$$\forall p' \in P_e : \qquad |\{C(p) \mid p \in \mathcal{E}(p')\}| = 1 \tag{3.8}$$

$$\forall p' \in P_e : \qquad |\{Q(p) \mid p \in \mathcal{E}(p')\}| = 1 \tag{3.9}$$

$$\bigcup_{p' \in P_e} \mathcal{E}(p') = P \tag{3.10}$$

$$\forall p' \in P_e : \qquad \mathcal{E}(p') \neq \emptyset, \mathcal{E}(p') \in \Omega \tag{3.11}$$

A restrição 3.6 é uma adição na modelagem anterior e garante que em cada paletização haja apenas encomendas de um mesmo item. Considerando a função $\mathcal{E} : P_e \to 2^P$, $\mathcal{E}$ relaciona o conjunto de paletes englobados P_e ao subconjunto de todos os englobamentos possíveis de paletes simples, sendo o englobamento válido somente se o conjunto de paletes hospedados sastisfizer as restrições; a restrição 3.7, então, garante que um mesmo palete p_i não possa pertencer a dois englobamentos válidos simultaneamente. As restrições 3.8 e 3.9 garantem que haja uma única centralizadora para cada palete englobado e também uma única produtora para cada um deles. A restrição 3.10 garante que todos os paletes hospedados estarão contemplados nos englobamentos e a restrição 3.11, assim como a 3.5 garante que todas as restrições particulares de peso e formato dos paletes englobados sejam respeitadas, restrições essas representadas pelo conjunto Ω.

Além disso, os modelos de Paletização são, também, considerados Problemas de Otimização Restrita, isto é, possuem uma solução ótima e, consequentemente, uma função objetivo. A função objetivo selecionada para a otimização das soluções foi a quantidade de paletes. Em ambos modelos, a função objetivo é:

$$Minimize \quad |\mathcal{P}| \tag{3.12}$$

Essas modelagens foram representadas por meio da ferramenta Minizinc, previamente citada em capítulos anteriores. Esta, por sua vez, além de suas características próprias referentes a declaração de variáveis, também possui sintaxe apropriada para a fácil representação da linguagem matemática apresentada. A exemplo da modelagem no Minizinc do Enigma do Einsten no capítulo 2, o modelo respectivo da Paletização Englobada figura no anexo B.

Esse modelo recebe como valores de entrada os parâmetros na primeira seção do código, até a linha 33 recebidos por meio de um arquivo de extensão ".dzn"(exemplo de arquivo em C). A possibilidade de carregamento de dados via arquivo externo estruturado foi fundamental para o desenvolvimento de possibilidades aleatoriezadas de instâncias de entrada para fins de teste e análise de eficiência do modelo.

As restrições do modelo matemático previamente apresentado foram replicadas por meio da sintaxe do Minizinc, que possui como característica a presença de funções *built-in* que

representam operadores matemáticos. Essa versão do modelo representada é um incremento de uma modelagem previamente fornecida (Derenievicz et al., 2023), que originalmente atacava o problema de paletização simples do PNLD. Este incremento, por sua vez, inclui restrições necessárias para a realização da paletização englobada. Em ambas as versões do modelo, é importante ressaltar que as restrições representantes da 3.5 e 3.11 da modelagem matemática foram simplificadas para representar parâmetros de peso máximo de paletes, no Minizinc representadas pela "CapacidadePalete"e "CapacidadePaleteEnglobado", na linha 9. É importante frisar que na prática, as restrições referentes às capacidades físicas são mais elaboradas do que as que foram empregadas, contudo é possível a utilização de outros modelos de otimização, de natureza opaca ao presente modelo em questão, capazes de retornar a validade dos paletes montados.

4 SOLUÇÕES E ANÁLISES

4.1 GERAÇÃO DE INSTÂNCIAS

Em decorrência da abordagem de modelagem previamente exposta, torna-se evidente a crucial necessidade de incorporar uma diversidade significativa de instâncias ao processo de validação de modelos algorítmicos de otimização. A robustez e confiabilidade de qualquer algoritmo de otimização dependem diretamente da capacidade de lidar eficientemente com uma gama abrangente de cenários e complexidades inerentes aos problemas em questão.

A inclusão de instâncias diversas não apenas amplia o espectro de desafios enfrentados pelo modelo, mas também proporciona uma avaliação mais abrangente de sua eficácia. Diferentes conjuntos de dados de teste podem representar variações nas condições do problema, refletindo nuances do mundo real que um único conjunto de dados não seria capaz de abranger. Isso é especialmente crucial em ambientes complexos e dinâmicos, nos quais a validação de modelos deve ser holística e abrangente.

Ao submeter o modelo a uma variedade de instâncias, é possível identificar melhor suas limitações, pontos fortes e áreas de aprimoramento. Essa abordagem permite que o algoritmo seja ajustado para lidar de maneira mais eficiente com uma ampla gama de circunstâncias, contribuindo assim para a generalização e aplicabilidade real do modelo. Além disso, a diversidade nas instâncias de teste é fundamental para mitigar o risco de sobreajuste do modelo a um conjunto específico de dados.

Foi partindo dessa perspectiva que foi desenvolvido, para fins desta pesquisa e, também, por conta da problemática supracitada, um gerador de instâncias capaz de produzir entradas aleatoriezadas para a modelagem no Minizinc, seguindo o formato disponível em C. O criador de instâncias, disponível em (Santiago, 2023), foi projetado em c e recebe de entrada cinco parâmetros, números inteiros, em linha de comando, e monta a partir dela uma instância limitadamente aleatorizada. Os parâmetros de entrada são, respectivamente, número de paletes, número de produtores, número de itens, número de centralizadoras e número de destinos. Além dos parâmetros de entrada, o algortimo também conta com valores constantes, ajustáveis no código fonte, que delimitam o funcionamento interno e ajustam a precisão da instância, como veremos a seguir. A principal ideia do criador de instâncias é que independente da precisão ajustada do modelo, sempre haverá alguma solução válida para o conjunto de valores gerados.

O primeiro parâmetro representa o número de paletes de uma solução ótima possível a ser encontrada na execução da instância gerada [1]. O algoritmo, então, busca preencher a quantidade de paletes estipulada com a maior quantidade possível de encomendas, com base no peso de cada uma delas e na capacidade máxima dos paletes, definidas na linha 6 do gerador. O peso das encomendas é decidido antes de seus respectivos identificadores, produtores e destinatários, estabelecido de forma aleatória entre um valor mínimo, definido na linha 8, e um valor máximo, definido na linha 7.

Os paletes são preenchidos até que o peso de encomendas no total seja no máximo a capacidade de cada palete e, só então, são definidos os valores de ID de cada encomenda figurando nos paletes. Essa decisão foi tomada pois só é possível determinar a quantidade total

[1] O criador de instâncias foi criado para alimentar, originalmente, o modelo de Paletização Simples. O modelo de Paletização Englobada é mais restrito do que o de Paletização Simples e, por isso, uma solução encontrada nele provavelmente será menos otimizada do que a que serviu de base para o criador de instâncias, isto é, usará mais paletes do que o número fornecido na entrada do algoritmo gerador.

numPalete	Quantidade de Paletes
numProdutores	Quantidade de Produtoras
numItens	Quantidade de Itens no Total
numCentralizadoras	Quantidade de Centralizadoras
numDest	Quantidade de Destinatários
MIN_WEIGHT	Peso mínimo de cada encomenda
MAX_WEIGHT	Peso máximo de cada encomenda
MAX_CARGO	Capacidade do palete
K_ADJ	Gera paletes finais com mais ou menos espaço sobrando

Tabela 4.1: Parâmetros de Entrada do Gerador de Instâncias

de encomendas uma vez que o processo de preenchimento dos paletes tenha sido concretizado, uma vez que os pesos de cada encomenda não são fixos, havendo uma variabilidade dentro de um espectro específico que pode ser aumentado ou diminuído à preferência daquele que rodar o algoritmo. É também por conta dessa característica do gerador de instâncias que o tamanho da entrada, a quantidade de encomendas, está diretamente relacionado com a quantidade de paletes requisitados ao rodar o gerador de instâncias. De forma mais exata, a quantidade de encomendas é proporcional à quantidade de paletes, a capacidade de peso neles definidas e o peso máximo estipulado para cada encomenda.

Além da relação de proporcionalidade entre a quantidade de encomendas e a quantidade de paletes, podemos destacar que o preenchimento dos paletes para a geração da instância ocorre de forma bastante justa, de forma a aproveitar a capacidade do palete. Por conta disso as chances de que a solução gerada seja a ótima são aumentadas. Dessa forma, podemos validar a eficácia do modelo com base em seu comportamento na busca por uma solução - Soluções melhores indicariam que o modelo não está completo em suas restrições, enquanto soluções piores poderiam indicar que ou o ótimo não foi obtido ou as restrições podem estar limitando mais do que o necessário.

Após definirmos os IDs das encomendas de cada palete, se torna necessária a definição dos destinos e de suas respectivas centralizadoras. Essa ordem de definições foi escolhida pois cada palete, seguindo o modelo de Paletização Simples, deve ser entregue exclusivamente a um único centralizador. Ainda, cada destino deve ser contemplado por um único centralizador, enquanto cada centralizador, por sua vez, pode administrar um destinatário ou mais. Portanto, ao definirmos as centralizadoras em conjunto de seus respectivos destinos contemplados, podemos garantir que uma vez que os paletes sejam distribuídos a uma centralizadora, os destinos de cada encomenda presente no referido palete sejam apenas aqueles contemplados pela centralizadora escolhida. Essa característica também exige que a quantidade de destinatários, recebida de parâmetro de entrada, seja igual ou maior que a quantidade de centralizadoras, também parâmetro, já que cada centralizadora tem pelo menos um destinatário.

Uma vez definidas as centralizadoras, seus destinatários e os paletes que serão entregues, é necessário estabelecer quais produtoras acolherão cada palete, assim como os itens produzidos por elas. Os itens, cujos ids vão de 1 até o número estipulado no parâmetro de entrada do executável, são embaralhados e distribuidos de maneira uniforme entre as produtoras. Em seguida, os paletes são divididos entre as produtoras, para que a mesma lógica de se atribuir itens pertencentes a uma produtora às encomendas de cada palete se aplique novamente. Se todos os processos de distribuição e alocação forem bem sucedidos, o algoritmo imprime, na saída padrão, uma instância estruturada no formato do ".dzn"de entrada para o modelo de Paletização,

assim como imprime, na saída de erros, os paletes de solução gerados para a instância impressa. Caso contrário, há apenas a impressão do erro na saída de erros.

Além dos pontos observados sobre o Gerador de Instâncias, é possível destacar o papel das constantes ajustáveis, citadas previamente. As constantes `MIN_WEIGHT` e `MAX_WEIGHT` definem o peso mínimo e o peso máximo das encomendas, respectivamente. A constante `MAX_CARGO`, por sua vez, define a capacidade máxima do palete. Contudo, a constante `K_ADJ` de ajuste não representa um valor palpável mas uma forma de tornar a solução base construída pelo Gerador de Instância menos eficiente. Isso ocorre pois a constante de ajuste é subtraída da capacidade do palete considerado de forma a fazer o algoritmo tratar a capacidade efetiva do palete como se fosse menor. Com mais espaço disponível no palete a solução base para a construção da instância é menos otimizada. Essa escolha foi tomada no algoritmo para fins de analisar a capacidade do MiniZinc de encontrar soluções mais eficientes do que as propostas pelo gerador de instâncias, além da eficácia do próprio modelo de paletização.

4.2 ANÁLISE DE EXECUÇÃO NO MINIZINC

As modelagens de Paletização, tanto simples quanto englobada, abrem espaço para até a quantidade de encomendas em quantidade de paletes totais. De fato, na modelagem apresentada a "pior"solução possível, isto é, a solução com a maior quantidade possível de paletes é aquela que utiliza um palete para cada encomenda. Assim como temos uma "pior"solução possível, temos, também, uma "melhor"solução possível. Embora nenhuma das restrições definam a necessidade de que cada produtora possua um item, o gerador de instâncias considera que todo produtor gera ao menos 1 item, no intuito de garantir que cada produtora e item seja contemplada na solução final. Portanto, a menor quantidade de paletes possíveis na solução da Paletização Simples é:

$$min(|\mathcal{P}(p)|) = max(|Q|, |C|). \tag{4.1}$$

Isso significa que mesmo na Paletização Simples, cuja paletização consome menos paletes do que a Paletização Englobada, temos um valor mínimo possível para o total de paletes utilizados. Consequentemente, a Paletização Englobada apresenta valores mínimos possíveis maiores ou iguais ao de Paletização Simples. Ter o conhecimento da existência de valores mínimos pode ser útil em processos de estimativa ou planejamento de custo de recursos durante o Processo do PNLD.

No entanto, o Minizinc apresenta mais sutilezas no processo de busca intensiva por soluções. Além das diferenças para com os paradigmas de linguagem tradicionais, como citado em 2.4, o Minizinc disponibiliza a possibilidade do uso de múltiplos *solvers* diferentes na solução de problemas de otimização e satisfação. Ao rodarmos os modelos, podemos selecionar quais deles podemos empregar e cada um deles, em geral, apresenta funcionalidades próprias, como a impressão de soluções parciais, impressão de todas as soluções possíveis ou a representação gráfica da árvore de buscas gerada durante o processo de otimização. Portanto, ao analisarmos a performance do MiniZinc no processo de busca de soluções, precisamos não apenas da avaliação de diferentes tamanhos de entrada do modelo, como, também, de execuções nos diferentes *solvers* disponíveis.

Por conta disso, foram realizados testes de execução em diferentes *solvers*, mantendo-se fixos os tamanhos de entrada para que apenas a eficiência do percorrer da árvore de buscas e da otimização algoritmica do *solver* pudesse ser comparada entre os demais. Além disso, também foram realizados testes mantendo fixo o *solver* e variando entre as instâncias os tamanhos de

entrada, em específico a quantidade de encomendas. Por último, também se avaliou a diferença de eficiência entre os modelos de Paletização Simples e de Paletização Englobada, no intuito de mensurar o impacto na solução que a necessidade de mais restrições nos paletes pode vir a ter no resultado.

4.3 RESULTADOS

Os primeiros testes a serem realizados foram os de execução do modelo de Paletização Simples, todos concluidos a partir do *solver* Geocode. Com base nas constantes, como visto na seção 4.1, e também na quantidade de paletes especificados nos parâmetros de entrada, podemos gerar instâncias com valores dentro de um certo alcance. Os testes realizados com *solver* fixo foram realizados com entradas de tamanho variando de 11 encomendas até 25. O tamanho de paletes nas instâncias foi configurado para 5 unidades, com encomendas de peso variante entre 1 e 2 unidades.

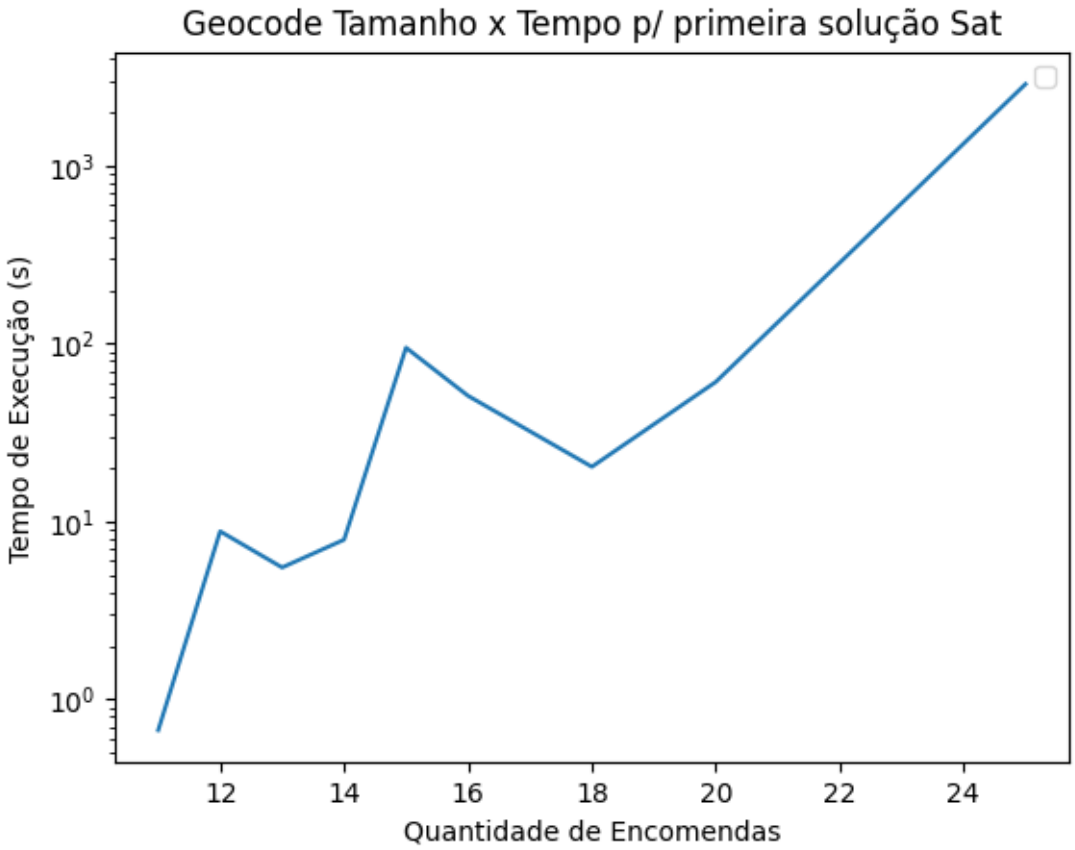


Figura 4.1: Coparação de Tempos - Geocode

Os tempos de execução foram adquiridos a partir de uma modelagem de satisfação, isto é, para o primeiro resultado encontrado capaz de satisfazer as restrições. Ao longo da execução desse modelo, a abordagem de selecionar a primeira opção encontrada se mostrou vantajosa por duas razões principais. A primeira delas é que, ao longo dos diversos testes realizados no Geocode, observou-se que a primeira solução encontrada no espaço de busca foi, em todas os testes realizados, a solução ótima por não haver outra com quantidade inferior de paletes. A segunda é que testar se uma solução é ótima requer a certeza de que qualquer outra solução

possível é no máximo tão boa quanto a melhor encontrada. Portanto, a busca pela solução ótima requer uma exploração muito mais profunda da árvore de estados. Além disso, o gerador de instâncias, em sua execução, nos retorna duas saídas: a instância propriamente dita e os paletes que serviram de origem para ela. A referência de comparação de resultados passa a ser, logo, o resultado encontrado pelo modelo no MiniZinc e os paletes gerados pelo gerador de instâncias. Ainda, além do problema em si escalar de forma exponencial com base na entrada, a certeza de uma solução minimizadora só ocorre após a total exploração das outras possibilidades de valoração. Foi a partir desse obstáculo que decidiu-se analisar o tempo para se encontrar uma solução satisfatória, e não exatamente uma solução ótima. Por termos em nosso alcance a solução ótima gerada pelo gerador de instâncias, podemos compará-la com a solução satisfatória encontrada pelo MiniZinc e no caso do GeoCode, ela coincidiu todas as vezes que testada.

Dentre os diversos algoritmos existentes na literatura para a solução desse tipo de problema combinatório, existem alternativas que nos permitem cortar ramos das árvores de estado sem que precisamos explorar suas folhas com detalhes, no entanto o problema da Paletização, tanto Simples quanto englobada, apresenta uma quantidade consideravelmente grande de estados, de forma que mesmo solvers bem otimizados requerem horas de execução para instâncias suficientemente grandes. Como exemplo, podemos destacar os seguintes resultados:

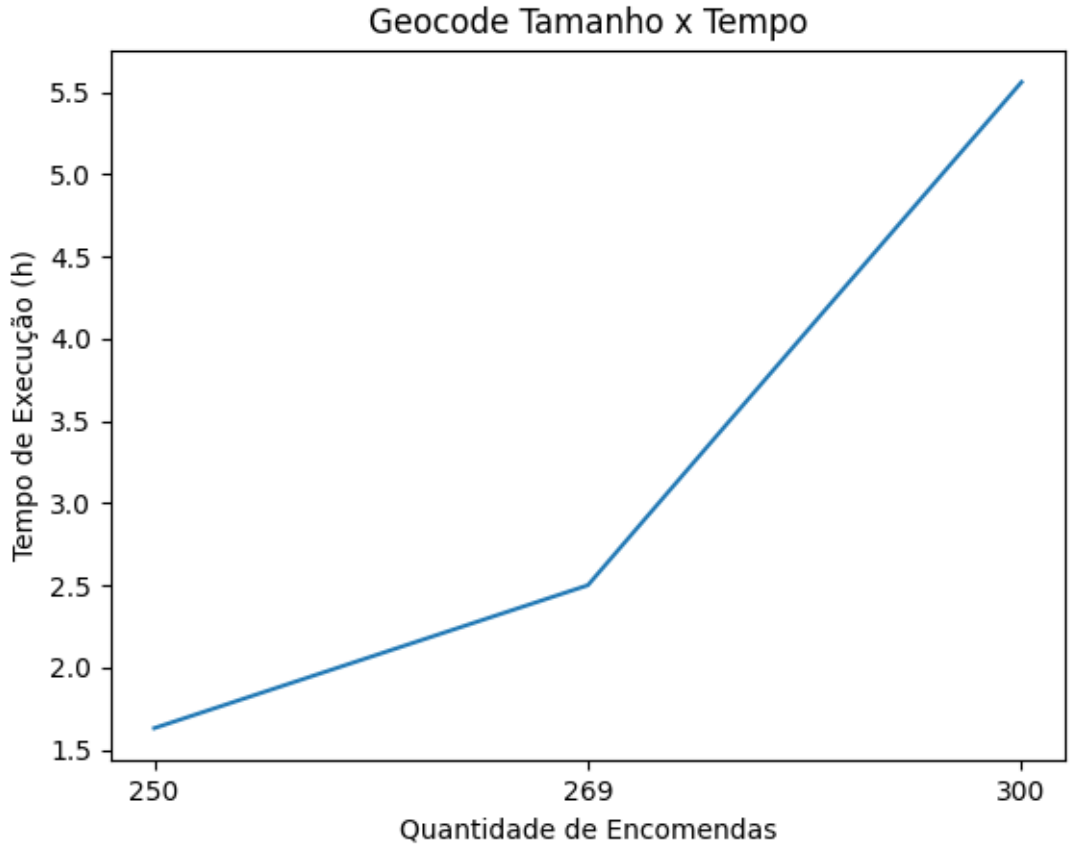


Figura 4.2: Comparação de Tempos (Grandes Entradas) - Geocode. Também soluções satisfatórias apenas

Como observável na figura 4.2, para entradas de 250 encomendas e acima, o tempo para se encontrar uma solução viável se encontra na escala de horas. O aumento de tamanho de entrada se reflete em um rápido aumento exponencial do tempo de execução para valores maiores do que 25 encomendas. De fato, a natureza combinatória do problema torna a busca por

solução exponencialmente custosa, não apenas em tempo, mas também em espaço, uma vez que, idealmente, o modelo deve ser rodado não apenas para uma satisfação de resultados, mas para a total otimização do problema. A forma com a qual o Geocode percorre a árvore se mostrou promissora, com a solução satisfatória encontrada também coincidindo com a solução ótima na maior parte das vezes.

Contudo, o Geocode não é o único *solver* disponível para a execução do modelo no MiniZinc. Foram avaliadas as eficiências dos diferentes *solvers* apresentados, em especial Chuffed, Coin-BC, HiGHs além do próprio Geocode.

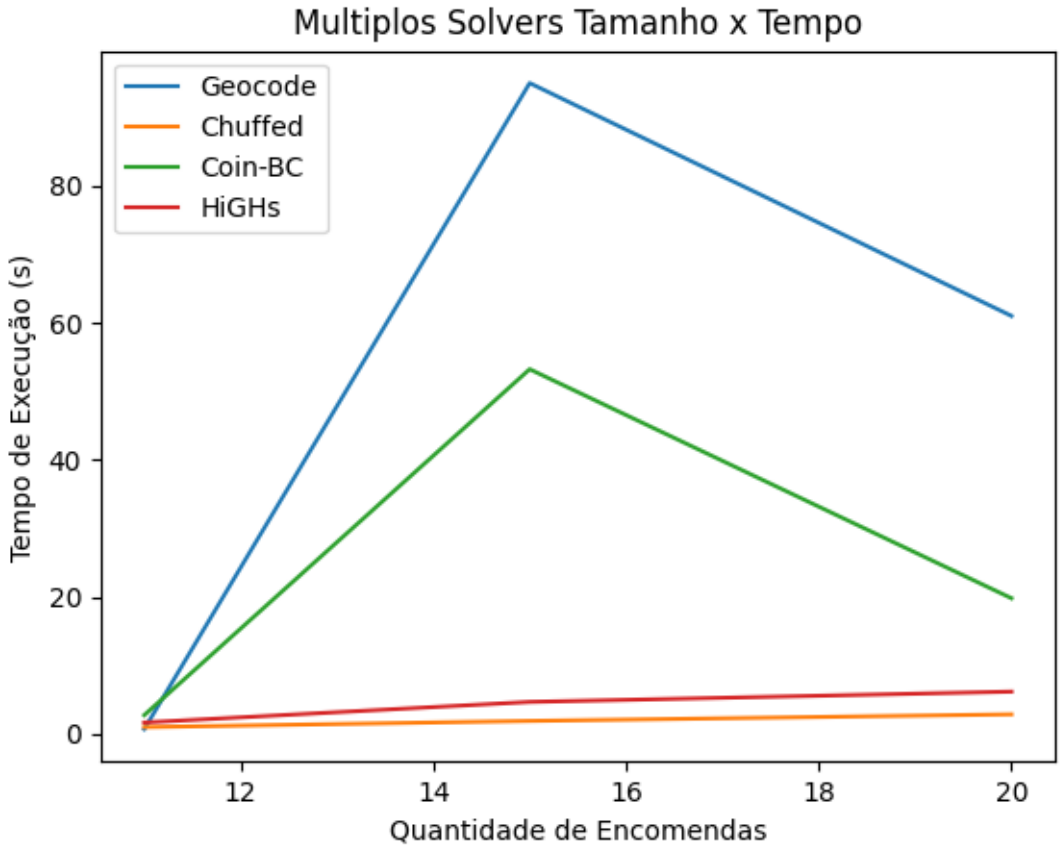


Figura 4.3: Comparação de Tempos - Multiplos Solvers

Ao observarmos gráfico 4.3, é possível ter a impressão de que GeoCode é o menos eficiente de todos os outros apresentados. No entanto, existem considerações a serem feitas. Primeiramente, os solvers foram empregados de forma a encontrar uma solução que satisfaça o problema. Isso significa que todos eles retornam o primeiro resultado que encontram, e não necessariamente esse resultado é o resultado ótimo. Na verdade, os resultados entregue pelos solvers que não são o GeoCode em múltiplas observações não foi o resultado ótimo e o próprio GeoCode apresentou soluções ótimas como as primeiras a serem geradas em todas as entradas testadas. Isso é possível de ser afirmado, uma vez que, durante a geração da instância pelo gerador, além da entrada referente à instância, também é gerada uma solução ótima para o problema, já que buscamos a minimização da quantidade de paletes. Qualquer solução que apresente um número maior de paletes do que o estipulado durante a geração da instância é, portanto, menos eficiente do que o desejado e não a solução ótima.

Número de Encomendas	Geocode	Chuffed	coin-BC	HiGHs
11	0.673s	0.971s	2.719s	1.643s
15	95s	1.873s	53.225s	4.648s
20	61s	2.836s	19.781s	6.152s

Tabela 4.2: Parâmetros de Entrada do Gerador de Instâncias

De um modo geral, podemos ter uma melhor percepção da eficiência dos *solvers* ante aos modelos ao buscarmos minimizar a solução e não apenas satisfazê-la. A partir dessa lente podemos observar que houve opções mais interessantes que o Geocode para entradas de diferentes tamanho, mesmo em um ambiente de minimização. Podemos destacar, por exemplo, os *solvers* Chuffed e Coin-BC foram capazes de retornar uma solução ótima em consideravelmente menos tempo do que o Geocode, respectivamente 18 segundos e 3min23s contra 19 minutos do GeoCode. Ainda, o Chuffed utiliza uma estratégia de limitação de profundidade durante o processo de busca, a partir de um certo número de filhos, determinado internamente, o Chuffed encerra a execução e retorna a melhor resposta encontrada. Essa estratégia de poda, para valores pequenos, se mostrou muito eficiente; para entrada de tamanho 11 Chuffed encontrou uma solução ótima 60 vezes mais rápido que o Geocode. Em ambos os casos, a solução ótima encontrada são seis paletes hospedados e o GeoCode explora toda a árvore de fato, o que nos garante que o valor retornado pelo Chuffed é correto.

Além dos comportamentos observados, foi possível notar, também, que para a entrada de tamanho 25, enquanto o Geocode demora 48 minutos para retornar um resultado que apenas satisfaz as restrições, o Chuffed e o Coin-BC são capazes de gerar uma solução, também de satisfação apenas, em menos de cinco minutos. A solução encontrada pelo Coin-BC é um palete menor do que a solução do Chuffed.

É relevante, também, analisarmos as diferenças de execução entre os modelos de Paletização Simples e Englobada. Ao compararmos ambos os modelos em seus tempos de execução, podemos observar que, para o alcance selecionado de entradas, o modelo de Paletização Englobada leva mais tempo para uma solução do que o modelo de Paletização Simples.

Embora esse comportamento fosse esperado, considerando que o modelo de Englobados introduz mais restrições ao modelo Simples, é possível observar comportamentos que fogem dessa expectativa - Para uma entrada de tamanho 18, o modelo de paletização simples rodou por 9 minutos sem encontrar uma solução, enquanto o modelo de paletização englobado devolveu uma resposta em vinte segundos. Ainda, ao rodar o modelo de paletização simples, para uma entrada de 20, o modelo levou 27 minutos sem encontrar uma solução enquanto o modelo de paletização englobado encontrou uma solução em menos de dois minutos. Salienta-se, contudo, que esses testes comparativos foram realizados através do *Geocode*. Ao utilizarmos outra ferramenta, como o Chuffed, por exemplo, uma solução é retornada e menos de dois segundos para entrada 20 e paletização simples, mas é garantidamente uma solução não ótima, uma vez que a solução detectada pelo Geocode possui menos paletes finais (6) do que a detectada pelo Chuffed (7).

A partir dos resultados apresentados aqui, é possível constatar que a modelagem de paletização simples se mostrou adequada para o fim a qual se propõe, uma vez que as soluções encontradas pelo GeoCode, mesmo que resultadas do processo de satisfação das restrições e não necessariamente de otimização, coincidem com as soluções propostas pelo gerador de instâncias, ao menos no que tange o número de paletes gerados.

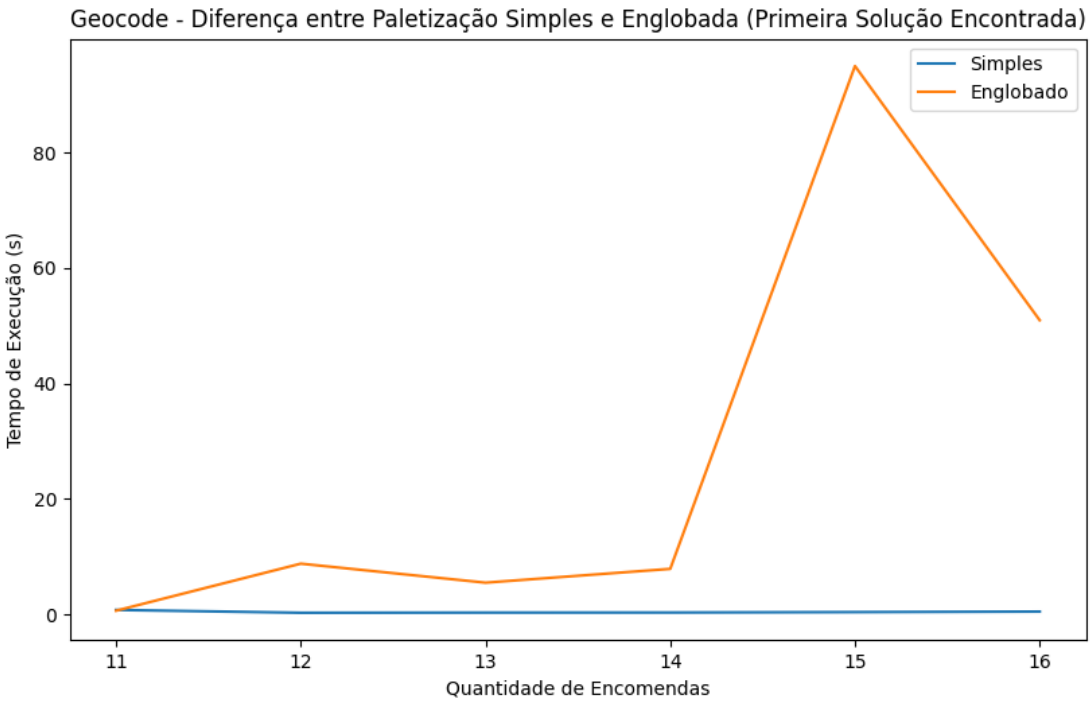


Figura 4.4: Comparação de Tempos - Simples vs Englobado

5 CONCLUSÃO

Após os testes realizados nos modelos a partir das instâncias geradas e pelos *solvers* empregados, foi possível observar que o Gerador de Instâncias desenvolvido pode representar uma ferramenta útil para a criação dos modelos de otimização. Conforme os trabalhos referentes ao PNLD prosseguem, um obstáculo na produção de conhecimento ao projeto é a não disponibilização dos métodos empregados pelas empresas responsáveis pela logística dos livros a serem entregues. Conforme citado em (Derenievicz et al., 2023), a otimização da etapa inicial de produção de paletes interfere progressivamente nas demais etapas da modelagem, o que é um incentivo para a busca de modelos mais eficientes de otimização.

Com base nos dados apresentados, os testes revelam insights valiosos sobre a resolução de problemas de paletização utilizando diversos solvers e modelos. Inicialmente, o solver Geocode demonstrou eficácia ao adotar uma abordagem que prioriza a primeira solução encontrada, frequentemente resultando em soluções ótimas, especialmente em instâncias de problemas menores.

Contudo, à medida que o tamanho do problema aumenta, o Geocode revela sua limitação, demandando tempos de execução significativos, na ordem de horas para instâncias com mais de 250 encomendas. Esta característica ressalta a complexidade exponencial associada à busca por soluções ótimas em problemas combinatórios extensos.

A comparação entre diferentes solvers, como Chuffed, Coin-BC, HiGHs, e o próprio Geocode, destaca que o Geocode pode não ser a opção mais eficiente em termos de tempo de execução, especialmente quando a otimização é crucial. Chuffed e Coin-BC emergem como alternativas mais rápidas, entregando soluções ótimas em lapsos temporais substancialmente menores.

Ao explorar os modelos de paletização simples e englobada, a expectativa de que o modelo englobado demande mais tempo se confirma, devido à introdução de restrições adicionais. No entanto, algumas instâncias revelam resultados inesperados, sugerindo que a escolha entre os modelos pode depender da natureza específica do problema.

É crucial considerar não apenas o tempo de execução, mas também a qualidade da solução ao avaliar a eficiência dos solvers. Enquanto o Geocode muitas vezes retorna soluções ótimas, solvers mais rápidos, como Chuffed e Coin-BC, podem oferecer eficiência notável, embora por muitas vezes empregando mais paletes que o GeoCode, considerando todas as buscas sendo realizadas para a satisfação do problema e não necessariamente otimização. A escolha do solver e do modelo deve ser uma decisão ponderada, levando em conta tanto o tempo de execução quanto a qualidade da solução desejada. Compreender as nuances individuais de cada solver e modelo é fundamental para uma escolha informada, adaptando-se às demandas específicas de cada problema de paletização.

Além de examinar minuciosamente as múltiplas execuções dos modelos de Paletização, este trabalho não apenas ofereceu uma análise, mas também buscou auxiliar o projeto do PNLD ao incrementar o modelo de Paletização Simples e integrá-lo ao modelo de Paletização Englobada. Mais do que isso, pode representar uma ferramenta de análise indireta dos intricados processos logísticos adotados por empresas responsáveis pela distribuição de materiais didáticos.

Ainda, é relevante destacarmos a inclusão da paletização englobada na geração de instâncias. Ao gerarmos as instâncias com base nos argumentos de entrada na linha de comando, os paletes utilizados como base para a saída ".dzn"não levam em consideração a otimização dos paletes englobados, considerando apenas a paletização simples. Para que isso fosse possível,

seria necessário, primeiro, definir o estado da paletização englobada antes do processo que leva à criação dos paletes simples. Talvez, uma das possibilidades para a realização dessa tarefa seria o encapsulamento das funções apresentadas dentro de uma outra função, que poderia distribuir os paletes simples ao longo dos paletes englobados, de forma a gerar uma paletização englobada também otimizada como base para a geração de instâncias. Além disso, o comportamento dos *solvers* empregados em termos de como as árvores de buscas são construídas e navegadas pode ajudar a compreender melhor os reais desafios relacionados à modelagem do problema de distribuição de livros do PNLD, representando, assim como a geração de instâncias para a Paletização Englobada, oportunidades de estudos futuros.

Podemos ressaltar, inclusive, o emprego do MiniZinc e do paradigma de Programação por Restrições para a modelagem dos problemas de paletização. De fato, a proximidade entre o paradigma do formalismo matemático tornou a transição direta para a representação computacional do problema, tarefa que teria sido mais complexa de ser realizada por meio de outros paradigmas. Essa escolha também impactou o gerador de instâncias, uma vez que a saída pôde ser estruturada em uma entrada ".dzn"empregada múltiplas vezes nos modelos.

Por último, o gerador de instâncias é capaz de criar entradas tão complexas e próximas da realidade de entregas quanto for necessário para aqueles que fizerem uso dele. Seu código permanecerá aberto e disponível a todos aqueles que se interessarem em contribuir para o projeto e também para melhorias em funcionamento e organização.

REFERÊNCIAS

Barták, R. (1998). Online guide to constraint programming. http://kti.ms.mff.cuni.cz/~bartak/constraints/index.html. Acessado em Junho de 2023.

Brasil (2017). Decreto nº 9.099, de 18 de julho de 2017. http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2017/decreto/D9099.htm. Acessado em Novembro de 2023.

Coulombe, C. e Quimper, C.-G. (2022). Constraint Acquisition Based on Solution Counting. Em Solnon, C., editor, *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, volume 235 de *Leibniz International Proceedings in Informatics (LIPIcs)*, páginas 15:1–15:16, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Derenievicz, G., Pereira, R., Peres, L., Castilho, M., Rodrigues, N. e da Cruz, S. (2023). Análise e modelagem da logística do programa nacional do livro e do material didático. Em *LV Simpósio Brasileiro de Pesquisa Operacional*, São José dos Campos - Brasil.

FNDE (2021). Programa nacional do livro e do material didático – pnld: Entendendo a distribuição. Relatório técnico.

Forrest, J., Ralphs, T., Vigerske, S. e Santos, H. G. (2019). Coin-or. https://github.com/coin-or/Cbc.

Huangfu, Q. e Hall, J. A. J. (2018). Parallelizing the dual revised simplex method. *Math. Program. Comput.*, 10(1):119–142.

Russel, S. e Norvig, P. (2021). *Artificial Intelligence, A Modern Approach*. Pearson.

Santiago, K. (2023). Criador de instâncias - git. https://github.com/kSantiagoP/TCC. Atualizado em Novembro de 2023.

Tack, G. (2014). Minizinc. https://www.minizinc.org/index.html. Acessado em Junho de 2023.

APÊNDICE A – MODELAGEM DO PROBLEMA DE EINSTEIN

```
1
2  include "alldifferent.mzn";
3
4  array[colors] of int: Colors = [1, % red
5                          2, % blue
6                          3, % yellow
7                          4, % green
8                          5 % white
9                          ];
10 array [nations] of int: Nations = [1, %Danish
11                          2, % British
12                          3, % German
13                          4, % Norwegian
14                          5 % Swedish
15                          ];
16 array [pets] of int: Pets = [1, %dog
17                          2, % fish
18                          3, % cat
19                          4, % bird
20                          5 % horse
21                          ];
22 array [drinks] of int: Drinks = [1, % milk
23                          2, % beer
24                          3, % coffee
25                          4, % tea
26                          5 % water
27                          ];
28 array [hobby] of int: Hobby = [1, % piano
29                          2, % violin
30                          3, % beseball
31                          4, % basketball
32                          5 % soccer
33                          ];
34 enum colors = {red, blue, yellow, green, white};
35 enum nations = {Danish, British, German, Norwegian, Swedish};
36 enum pets = {dog, fish, cat, bird, horse};
37 enum drinks = {milk, beer, coffee, tea, water};
38 enum hobby = {piano, violin, baseball, basketball, soccer};
39
40
41 array [1..5] of var 1..5: c;
42 array [1..5] of var 1..5: n;
43 array [1..5] of var 1..5: p;
44 array [1..5] of var 1..5: d;
45 array [1..5] of var 1..5: h;
46
47
48 %%cada campo da casa deve possuir apenas uma unica solucao
49 constraint forall (i in [1..5])(
50    alldifferent (c[i]) /\
51    alldifferent (n[i]) /\
52    alldifferent(p[i]) /\
53    alldifferent(d[i]) /\
```

```
54     alldifferent(h[i])
55 );
56 
57 constraint forall (i in 1..5)(
58     if n[i] = Nations[British] then c[i] = Colors[red] else true endif /\
59     if n[i] = Nations[Swedish] then p[i] = Pets[dog] else true endif /\
60     if n[i] = Nations[Danish] then d[i] = Drinks[tea] else true endif /\
61     if n[i] = Nations[German] then h[i] = Hobby[soccer] else true endif /\
62     if c[i] = Colors[green] then d[i] = Drinks[coffee] else true endif /\
63     if h[i] = Hobby[baseball] then p[i] = Pets[bird] else true endif /\
64     if h[i] = Hobby[basketball] then i > 0 /\
65     (p[i-1] = Pets[cat] \/ p[i+1] = Pets[cat]) else true endif /\
66     n[1] = Nations[Norwegian] /\
67     if h[i] = Hobby[violin] then d[i] = Drinks[beer] else true endif /\
68     if c[i] = Colors[yellow] then h[i] = Hobby[piano] else true endif /\
69     d[3] = Drinks[milk] /\
70     if p[i] = Pets[horse] then i > 0 /\
71     (h[i-1] = Hobby[piano] \/ h[i+1] = Hobby[piano]) else true endif /\
72     if c[i] = Colors[green] then c[i+1] = Colors[white] else true endif /\
73     if n[i] = Nations[Norwegian] then i > 0 /\
74     (c[i-1] = Colors[blue] \/ c[i+1] = Colors[blue]) else true endif /\
75     if h[i] = Hobby[basketball] then i > 0 /\
76     (d[i-1] = Drinks[water] \/ d[i+1] = Drinks[water]) else true endif
77      );
78 
79 solve satisfy;
```

APÊNDICE B – MODELAGEM DA PALETIZAÇÃO ENGLOBADA

```
% Entrada

int: qtd_Itens    ;
int: qtd_Encomendas ;
int: qtd_Destinatarios ;
int: qtd_Centralizadoras;
int: qtd_Produtores ;
int: T            ;
int: CapacidadePalete;
int: CapacidadePaleteEnglobado;

set of int: Itens    = 1..qtd_Itens;
set of int: Encomendas = 1..qtd_Encomendas;
set of int: Destinatarios = 1..qtd_Destinatarios;
set of int: Centralizadoras = 1..qtd_Centralizadoras;
set of int: Produtores = 1..qtd_Produtores;

% Considerando que no pior caso tera um palete por encomenda
% Mas a funcao objetivo busca minimizar o numero de paletes
set of int: Paletes  = 1..qtd_Encomendas;
set of int: PaletesEnglobados = 1..qtd_Encomendas;
set of int: IdPaleteSimples = 1..qtd_Encomendas;

array[Itens] of set of int: S;

array[Encomendas] of int: dest;
array[Encomendas] of int: peso;




array[Destinatarios] of set of int: cent;

array[Produtores, Itens] of int: qtd;

% Variaveis

int: qtd_Paletes == qtd_Encomendas;

array[Paletes] of var Produtores: Q;
array[Paletes] of var Centralizadoras: C;
array[Paletes] of var set of Encomendas: P;
array[PaletesEnglobados] of var set of IdPaleteSimples: E;

% Restricoes

constraint forall(p1, p2 in Paletes)(
           if p1 != p2 then P[p1] intersect P[p2] = {}
           else true endif );


constraint forall(p1, p2 in PaletesEnglobados)(
            if p1 != p2 then E[p1] intersect E[p2] = {}
            else true endif
```

```
);



constraint forall(p in Paletes)(
           exists(i in Itens)(
             P[p] subset S[i]
           )
);


constraint forall(p in Paletes)(
           forall (e in P[p])(
             C[p] in cent[dest[e]]
           )
         );

constraint array_union([E[p_e] | p_e in PaletesEnglobados]) == IdPaleteSimples;

constraint array_union([P[p] | p in Paletes]) == Encomendas;

constraint forall(q in Produtores)(
           forall(i in Itens)(
             sum(p in Paletes where Q[p] == q)
               (card(P[p] intersect S[i])) <= qtd[q,i]*T
           )
         );

constraint forall(p in Paletes)(
              %P[p] != {} /\ permitindo paletes vazios pra min
              sum(e in P[p])(peso[e]) <= CapacidadePalete
          );

constraint forall(p_e in PaletesEnglobados)(
            sum(p in E[p_e])(
               sum(e in P[p])(peso[e])
            ) <= CapacidadePaleteEnglobado
);


constraint forall(p in Paletes)(
   exists(i in Itens)(
      P[p] subset S[i]
   )
);


constraint forall(p_e in PaletesEnglobados)( %conferir
          forall(p1, p2 in E[p_e])(
            C[p1] = C[p2]
          )
);

constraint forall(p_e in PaletesEnglobados)(
          forall(p1, p2 in E[p_e])(
            Q[p1] = Q[p2]
          )
);
```

```
112
113
114 % funcao objetivo
115
116 array[Paletes] of var int: palete_usado;
117 constraint forall(p in Paletes)(
118                palete_usado[p] == if P[p] == {} then 0
119                                   else           1 endif
120             );
121 solve minimize sum(palete_usado);
```

APÊNDICE C – EXEMPLO DE ARQUIVO .DZN

```
% MAX_WEIGHT 3, MIN_WEIGHT 1, MAX_CARGO 6, K_ADJ 0
qtd_Itens      = 3;
qtd_Encomendas     = 16;
qtd_Destinatarios    = 3;
qtd_Centralizadoras  = 2;
qtd_Produtores     = 1;
T        = 16;
CapacidadePalete    = 6;
CapacidadePaleteEnglobado = 13;
dest = [1,3,3,1,2,3,1,2,1,1,1,3,1,1,2,1];
peso = [2,1,3,1,3,1,1,1,1,1,3,1,1,1,2,1];
S = [{5,8,4,9},{11,7,10,15,2,12,14},{13,3,6,16,1}];
cent = [{2},{1},{1}];
qtd = [|1,1,1|];
```

APÊNDICE D – MODELAGEM PALETIZAÇÃO SIMPLES

```
include "ide/vis.mzn";
% Entrada

int: qtd_Itens    ;
int: qtd_Encomendas ;
int: qtd_Destinatarios ;
int: qtd_Centralizadoras;
int: qtd_Produtores ;
int: T            ;
int: CapacidadePalete;

set of int: Itens    = 1..qtd_Itens;
set of int: Encomendas = 1..qtd_Encomendas;
set of int: Destinatarios = 1..qtd_Destinatarios;
set of int: Centralizadoras = 1..qtd_Centralizadoras;
set of int: Produtores = 1..qtd_Produtores;

% Considerando que no pior caso tera um palete por encomenda
% Mas a funcao objetivo busca minimizar o numero de paletes
set of int: Paletes  = 1..qtd_Encomendas;

array[Itens] of set of int: S;

array[Encomendas] of int: dest;
array[Encomendas] of int: peso;




array[Destinatarios] of set of int: cent;

array[Produtores, Itens] of int: qtd;

% Variaveis

% qtd_Paletes nao vai ser variavel
%var int: qtd_Paletes;
int: qtd_Paletes == qtd_Encomendas;

array[Paletes] of var Produtores: Q;
array[Paletes] of var Centralizadoras: C;
array[Paletes] of var set of Encomendas: P;

% Restricoes

constraint forall(p1, p2 in Paletes)(
           if p1 != p2 then P[p1] intersect P[p2] = {}
           else true endif );




constraint forall(p in Paletes)(
          forall (e in P[p])(
```

```
54              C[p] in cent[dest[e]]
55            )
56          );
57
58  constraint array_union([P[p] | p in Paletes]) == Encomendas;
59
60  constraint forall(q in Produtores)(
61            forall(i in Itens)(
62              sum(p in Paletes where Q[p] == q)
63                 (card(P[p] intersect S[i])) <= qtd[q,i]*T
64            )
65          );
66
67  constraint forall(p in Paletes)(
68                %P[p] != {} /\ permitindo paletes vazios pra min
69                sum(e in P[p])(peso[e]) <= CapacidadePalete
70           );
71
72  % funcao objetivo
73
74  array[Paletes] of var int: palete_usado;
75  constraint forall(p in Paletes)(
76                 palete_usado[p] == if P[p] == {} then 0
77                                    else          1 endif
78              );
79  solve minimize sum(palete_usado);
80
81  output [ show(P) ] ++ [show(sum(palete_usado))];
```